

**Learning by Game-Building
in
Theoretical Computer Science Education**

Laura Hutchins-Korte



Doctor of Philosophy
Institute for Communicating and Collaborative Systems
School of Informatics
University of Edinburgh
2008

Abstract

It has been suggested that theoretical computer science (TCS) suffers more than average from a lack of intrinsic motivation. The reasons provided in the literature include the difficulty of the subject, lack of relevance to the informatics degree or future careers, and lack of enjoyment of the learning experience. This thesis presents evidence of these claims derived from the results of an expert survey.

Increasing the students' perceived control of the learning experience has been shown to increase student motivation in numerous different settings. A few of those also showed increased student performance. This thesis proposes that game-building can be such a setting for the area of TCS. Within the area of TCS, the focus will be on the modelling skills (finite state automata, push-down automata, Turing machines, CCS, etc.) since they form the majority of the curriculum at undergraduate level.

It will be demonstrated how arbitrary TCS modelling skills can be mapped onto a game-building framework and allow the students to learn about the former by using the latter. It is hypothesized that the success of the approach depends on the amount of control given to the student. To test this claim, two experimental conditions were used in a repeated-measures design: (1) own-game and (2) pre-defined game. In the former, students are asked to write a game of their own, whereas in the latter, they are asked to copy a pre-defined game.

A large demand for the own-game context was observed and results of its effect on performance and enjoyment are presented. Although no main effect of the own-game condition versus the pre-defined game condition was found in terms of either enjoyment or performance, some interesting interaction effects between condition and motivational type were unveiled.

Acknowledgements

First of all, I would like to thank my supervisor Helen Pain, for taking me on as a student. At the time, I was two years into my PhD and had reached a dead end in my research on type-theory. My heart had always been in education and I wanted to use the opportunity to do research on a topic that I really enjoy. Taking me on under those circumstances was a risk, and I am extremely grateful for the opportunity Helen offered me.

Judith Good and Stuart Anderson have both provided valuable input as my second supervisors. Despite the distance between Sussex and Edinburgh, Judith has always provided me with the help I needed and her experience with research on game-building tasks proved very helpful. Stuart Anderson has been my link between theoretical computer science (TCS) and teaching practice. I am grateful for the high-level perspective he brought to my research, and for his never-ending enthusiasm and encouragement.

I would also like to thank Frank Keller for sharing his knowledge of statistics with me; Jim Buckley, Hamish McCloud, Judy Robertson, Keith Stenning, and John Lee for their input at various stages of my research; and Guido Rößling for giving me an invaluable piece of advice on writing a PhD-Thesis.

Stuart Anderson, Bonnie Webber, Frank Keller, and Stratis Viglas all allowed me to conduct my research in context of their courses, for which I am extremely grateful, while DeLesley Hutchins did an amazing job writing the game-engine used in Study 2, which allowed the students to create a rich array of diverse games.

Furthermore, I really appreciate the time, energy, and enthusiasm that all the students who participated in Studies 2 and 3 put into writing such wonderfully inventive games (which I probably spent too much time playing!). Without their hard work and creativity, I would not have been able to write this thesis.

I also really appreciate the time and effort that the lecturers who participated in Study 1 put into sharing their experience and opinions on TCS education. Filling out surveys is a thankless job, but it makes an important contribution to studies like this one.

Financially, I am indebted to the Laboratory for the Foundations of Computer Science (LFCS) for awarding me a scholarship and for extending it despite my change of topic. The Institute for Communicating and Collaborative Systems (ICCS) and the Informatics Graduate School (IGS) together made it financially possible to have Judith Good as my supervisor; I am thankful for their support as well.

Last, but certainly not least, I would like to thank my family. My parents for always encouraging me to do my best in everything I do; without their love and support, I never would have made it to where I am now. And my husband DeLesley, for sharing with me both the burdens (incl. PhDs!) and the joys of life:

*Samen zijn is samen lachen, samen huilen, leven door dichtbij elkaar te zijn.
Samen zijn is sterker dan de sterkste storm, gekleurder dan het grauwe om ons heen.
Want samen zijn, ja met z'n twee, te beleven één te zijn.*
(Willeke Alberti)

Declaration

I declare that this thesis was composed by myself, that the work contained herein is my own except where explicitly stated otherwise in the text, and that this work has not been submitted for any other degree or professional qualification except as specified.

(Laura Hutchins-Korte)

Table of Contents

1	Introduction	1
1.1	Introduction	1
1.2	Theoretical Computer Science	2
1.3	Learning by Game-Building	2
1.4	Thesis Questions	3
1.5	Overview	5
2	Study I: Motivation in TCS Education	7
2.1	Introduction	7
2.1.1	Design	8
2.1.2	The Questionnaire	9
2.1.3	The Interface	11
2.1.4	Pilot Study	11
2.1.5	Hypotheses	17
2.2	Method	18
2.2.1	Participants	18
2.2.2	Materials	18
2.2.3	Procedure	18
2.3	Results	19
2.4	Discussion	30
2.4.1	The Importance of TCS	30
2.4.2	Lecturers' Thoughts on Student Opinion	31
2.4.3	Suggested Improvements	33
2.5	Summary	34
3	Background	35
3.1	Introduction	35

3.2	Learning Objectives in Theoretical Computer Science	35
3.2.1	Bloom’s Taxonomy	36
3.2.2	A Younger Audience	37
3.3	Constructionism	38
3.3.1	Logo and its Relatives	39
3.3.2	Programming in Other Subject Areas	41
3.3.3	The Overhead Problem	44
3.4	Motivation	45
3.4.1	Intrinsic vs. Extrinsic Motivation	45
3.4.2	Scales and Measures	46
3.4.3	Choice	52
3.4.4	Programming	54
3.5	Game-Based Learning	55
3.6	A Word on Transfer	63
3.6.1	Is Programming the New Latin?	64
3.7	Summary	64
4	Learning by Game-Building in Theoretical Computer Science	66
4.1	Introduction	66
4.2	Examples	67
4.2.1	Finite State Automata (FSAs) and Regular Expressions (REs)	67
4.2.2	Context-Sensitive Grammars (CSGs)	69
4.2.3	Turing Machines (TMs)	76
4.2.4	Concurrency: Value-Passing CCS	79
4.3	Practicalities	83
4.3.1	Playing the Games	83
4.3.2	Guiding the Building Process	84
4.3.3	The Overhead Problem	85
4.3.4	Assessing the Resulting Games	86
4.4	Summary	86
5	Study II: Motivation and Preference	88
5.1	Introduction	88
5.1.1	Design	89
5.1.2	Hypotheses	95
5.2	Method	98

5.2.1	Design	98
5.2.2	Participants	98
5.2.3	Materials	98
5.2.4	Procedure	99
5.3	Results	100
5.3.1	Experimental Condition Choice	100
5.3.2	Motivation (Achievement Goals)	102
5.3.3	Achievement Goals and Choice	103
5.4	Discussion	111
5.4.1	Choice	111
5.4.2	Achievement Goals	112
5.4.3	Achievement Goals and Choice	113
5.4.4	Design	114
5.4.5	Validity	115
5.5	Summary	117
6	Study III: Affect and Performance	119
6.1	Introduction	119
6.2	Design	120
6.2.1	Empirical Design	120
6.2.2	Measuring Enjoyment and Performance	122
6.2.3	Measuring Motivation and Time	123
6.2.4	Derived Motivation Measures	126
6.3	The Game-Building Tasks	129
6.3.1	Study 3a: Finite State Automata	129
6.3.2	Study 3b: Turing Machines	131
6.4	Hypotheses	134
6.4.1	Performance	134
6.4.2	Enjoyment	137
6.5	Method	140
6.5.1	Design	140
6.5.2	Participants	140
6.5.3	Materials	140
6.5.4	Procedure	143
6.6	Results	145

6.6.1	Raw Data	145
6.6.2	Pre-Defined Game vs. Own Game: Performance	151
6.6.3	Pre-Defined Game vs. Own Game: Enjoyment	156
6.7	Discussion	164
6.7.1	Pre-Defined Game vs. Own Game: Performance	164
6.7.2	Pre-Defined Game vs. Own Game: Enjoyment	167
6.7.3	Validity	170
6.7.4	Ethical Considerations	171
6.8	Summary	172
7	Conclusions	173
7.1	Contributions and Findings	173
7.2	Discussion	175
7.3	Future Work	179
7.4	Conclusions	182
	Bibliography	184
A	Questionnaire 1	194
B	Questionnaire 2	197
C	Communicating Objects Assignment (2005–2006)	200
D	Communicating Objects Assignment (2006–2007)	216
E	Finite State Automata Assignments	227
F	API for FSA.py	242
G	Finite State Automata Assignment (Marking Guide)	247
H	Turing Machine Assignments	253
I	API for tm.py	272
J	Turing Machine Assignment (Marking Guide)	278

Chapter 1

Introduction

[...] Greater attention to the individualization of learning environments on motivational, as well as instructional, grounds may ultimately pay large educational benefits. (Malone and Lepper, 1987)

1.1 Introduction

It is claimed in the literature (e.g. Rodger *et al.*, 2006; Verma, 2005; White and Way, 2006) that the area of theoretical computer science (TCS) suffers more than average from a lack of intrinsic motivation among students, although no form of evidence is provided. Chapter 2 elaborates on this. The literature provides three reasons for the decreased motivation in TCS:

1. difficulty of the subject matter
2. lack of enjoyment, frequently referred to as ‘boring’.
3. lack of relevance to the university degree and future career

Students are not generally lacking in motivation in educational contexts though. They gladly spend large amount of their time at student societies or virtual communities, many of which are centred around learning (Kuh, 1995). They learn anything from karate and wine tasting to a new language and how to fix a bike. The question is: *how do we tap into this well of motivation?*

1.2 Theoretical Computer Science

Before we can attempt to answer this question, we need to elaborate on the problem itself. First of all, the statement that TCS suffers more than usual from the lack of intrinsic motivation is merely a claim, based on anecdotal evidence. In our first study (Chapter 2), we will put this claim to the test and determine the validity of the reasons which the literature provides for this larger-than-usual lack of motivation.

Second, TCS is a very large area and addressing all of it would be beyond the scope of this thesis. Instead, we decided to focus on the *modelling skills*, because they form the most common topics at undergraduate level, a claim which is backed up by Study 1 described in Chapter 2. Some of example of TCS modelling skills are: Finite State Automata (FSAs), Regular Expressions (REs), Context-Free Grammars (CFGs), Turing Machines (TMs), and the Calculus of Communicating Systems (CCS).

1.3 Learning by Game-Building

Of the three reasons provided for the decreased motivation in TCS, the first two – difficulty and lack of enjoyment – are often discussed in the literature in different contexts. For example, Papert (1980) claims in his theory of *constructionism* that students learn more and more easily when they are engaged in creating a public entity, whether this entity is a Lego-car or a PhD thesis.

Numerous publications have also demonstrated that adding personalization and choice can increase a person's enjoyment of a task. For example, DeCharms (as cited in Pfaffman, 2003) describes a study in which he asked college students to play with Tinkertoys either in a 'pawn' condition, where they had to follow instructions step-by-step, or in an 'origin' condition, where they were allowed to create their own model. DeCharms (1971) found increased affect for the students in the 'origin' condition. The same group was also more likely to continue working on the model, completed the task more elegantly, and recalled more of the names associated with the model.

In the proposed approach students will learn a new modelling skill by completing a game-building assignment in which there is a direct and transparent mapping between the game that the student is building and the model in the framework they are trying to master. We claim that game-building is the embodiment of the two paradigms mentioned above: constructionism and personalisation and choice.

The approach is *constructionist* in that the students learn by *constructing* an entity

which should therefore improve the student's understanding of the modelling framework (Gee, 2003; Papert, 1980; Harel and Papert, 1991). Because the students are free to design their own game, the approach also contains a significant degree of personalisation and choice, which can have beneficial effects on motivation (DeCharms, 1971; Zuckerman *et al.*, 1978; Cordova and Lepper, 1996) and even on performance (Cordova and Lepper, 1996). Each student can choose a context which is relevant to them personally. By allowing students to choose their own context, they are given an opportunity to draw on their interests and to ground the abstract concepts in a context which is meaningful for them personally. The benefits of this contextualization are discussed in theories of situated learning (Lave, 1988; Lave and Wenger, 1991).

In addition to this, the approach also has a social aspect, as games are written to be played by others. This could potentially help us to tap in to the well of student motivation, as most of their preferred activities also have a social aspect.

The approach has one more advantage, which is that it makes a very abstract subject matter, much more concrete. TCS is traditionally presented in an abstract manner, using small illustrative examples which are generally only marginally relevant to either the rest of the informatics degree or an informatics career. This fact is demonstrated by leading textbooks on the subject like Sipser (1997); Kozen (1997). Although they are both leading textbooks, they may not necessarily be appreciated by all informatics students. Research has shown the drawbacks of the traditional approach taken in these books, as described in Section 3.6. For example the Cognition and Technology Group at Vanderbilt (as cited in Squire, 2002) found that students who learn algebra through problem solving are more likely to use algebra in solving problems than students who learn algebra through the traditional means. We propose that game-building could be that problem-solving environment for the modelling skills in TCS.

1.4 Thesis Questions

This brings us to the six questions at the heart of this thesis and the three studies designed to answer them.

Question 1: *Are students really less motivated for TCS than they are for their other informatics subjects? If so, why?*

This first question tries to establish the validity of the problem this thesis aims to address. Additionally, it also tries to determine possible causes of the problem. Study 1

attempts to answer Question 1 using an online survey of university lecturers teaching TCS.

Question 2: *Can we use game-building as an approach to TCS education? If so, how?*

Although learning by game-building seems to have educational potential on paper, it is far from trivial how one would go about implementing the approach. Chapter 4 presents a detailed overview of the approach including several examples. This will answer Question 2.

Question 3: *Is there a demand for personalizable tasks among informatics undergraduates?*

One of the two major potential advantages of learning by game-building is the availability of personalization. However, for the approach to be successful, we need at the very least for students to prefer a personalizable task over a non-personalizable one. Therefore, by slight abuse of terminology, ‘demand’ is defined as follows:

There is a demand for personalizable tasks among informatics undergraduates, if at least some of them choose these tasks over non-personalizable tasks when given a choice between the two. The size of the demand depends on the number of students preferring personalizable tasks over non-personalizable tasks.

Study 2 attempts to answer Question 3 by presenting a large group of undergraduate students with exactly such a choice on a game-building task. In the personalizable version students are asked to write their own game, whereas in the non-personalizable version they are asked to copy a pre-defined game.

Question 4: *Is there a motivational difference between students who choose personalization and those who do not?*

Besides preference, Study 2 also records the students’ motivational type in an attempt to find a relationship between the two. For example, the motivation measure identifies a person’s affinity with *performance approach*, a desire to outperform one’s peers. We would expect students who choose to write their own game to generally exhibit more affinity with *performance approach* than those who prefer to copy a pre-defined game.

Question 5: *What is the effect of increased personalizability on enjoyment and performance?*

Study 3 attempts to answer this question using the two conditions introduced in Study 2, own game and pre-defined game, in a counter-balanced repeated measures design. Besides enjoyment and performance, several other data were collected including time spent on the tasks, motivational type, and gender.

Question 6: *For which kinds of students do these factors compare favourably and for which do they compare unfavourably?*

The success of the approach in terms of its effect on enjoyment and performance is expected to depend largely on individual differences. The differences themselves are expected to be at least partially based on motivational type and possibly gender. Therefore, we do not expect our approach to show an improvement in all students, since different students thrive in different environments, as shown in studies such as (Stenning, 2002; Lepper and Malone, 1987; Squire, 2004). For example, there are students who thrive in the traditional abstract setting. We do not expect them to show increased motivation or improved understanding in the proposed approach. Another example of a kind of student we do not necessarily expect to show any improvement is described by Brophy (1983); Pieper (2003). They suggest that there is a group of students whose main motivation is to do as little work as possible to obtain their desired grade. These *work-avoiding* students could potentially be put off by the design aspect of tasks with increased personalisation, as it may be perceived as ‘extra work’.

This expectation is less obvious than the previous one though. The reason being that some work-avoiding students could also classify designing a game as fun, rather than work, in which case their enjoyment and performance would not necessarily be influenced by the perceived amount of work needed to complete a game-building task.

Study 3 attempts to answer Question 6 by determining the nature of the relationship between the experimental condition and other variables like motivation type and gender.

1.5 Overview

The remainder of this thesis is organized as follows. Chapter 2 presents Study 1, which investigates the validity of the claim that students are less motivated for TCS than they are for their other informatics subjects. Chapter 3 presents an overview of previous and related research. Chapter 4, describes the suggested approach and also contains several examples of possible applications. Chapter 5 presents Study 2, which is an

investigation into the demand for personalisation among undergraduate informatics students. Chapter 6 presents Study 3, which is an investigation into the effect of the suggested approach on enjoyment and performance. Chapter 7 concludes.

Chapter 2

Study I: Motivation in TCS Education

In the previous chapter, the problem at the centre of our research was presented and an outline of the route to be taken to address this problem was described. Before we start on this route, we will take a step back and take a closer look at one of the premisses of this thesis: the claim that students are less motivated for theoretical computer science (TCS) than they are for their other informatics/computing subjects. Chapter 1 mentions this claim, but no evidence is provided in the literature. The current chapter takes a first step towards providing such evidence of this broad claim. Using an open-ended questionnaire addressed to international TCS educators, this chapter tries to determine the validity of the problem, locate its possible causes, and identify potential solutions to the problem.

Furthermore, an attempt will be made to create a short-list of the most important undergraduate topics within TCS, which will serve as a focus of the approach described in Chapter 4.

2.1 Introduction

In the literature it is often claimed that TCS is not a popular subject with students. Reasons given include the difficulty of the subject, lack of relevance to the informatics degree or future careers, and lack of enjoyment of the learning experience. For example, Rodger *et al.* (2006) writes the following about enjoyment of automata theory:

“[Many automata theory problems] are tedious to solve using pencil and paper.”

Verma (2005) talks about the issues of difficulty and relevance:

“Teaching Theory of Computation and learning it are both challenging tasks. Moreover, students are not sufficiently interested/motivated to learn this material since: (i) they believe that the material is dated and of little use and (ii) it is too abstract and difficult.”

White and Way (2006) also mentions the issue of difficulty:

“Within a computer science curriculum, courses in theory are often perceived by students as the most difficult to understand and do well in. Some survive the experience with only minimal understanding of the formal languages and automata (FLA) material that is vital to subsequent courses in programming languages and compilers. Computer science students generally enjoy writing computer programs, a reality that can be used to improve learning in FLA courses.”

This is in fact almost exactly the approach that is taken in our research as described in Chapter 4. However, instead of writing just any computer program, students will be writing simple computer games.

Despite this abundance of claims that TCS is difficult and not enjoyable or relevant, there is surprisingly little evidence to support them. In fact we were unable to find any. This brings us to the purpose of the study described in this chapter.

The purpose is two-fold: (1) to identify any possible negative influences on student motivation in the area of TCS and possible remedies which counter-act these influences and (2) to find an answer to the questions: (why) should TCS be in the undergraduate TCS curriculum and if so, which topics should we focus on? A survey-based study was designed with these questions in mind.

2.1.1 Design

A design decision was made to use a questionnaire with open questions to avoid leading the participants. A direct and fairly large disadvantage of this approach is the nature of the resulting data.

A choice for lecturers rather than students was made for several reasons. First of all, lecturers are able stand back and observe how student motivation changes on the same material using several different approaches. Second, they possess enough knowledge on the subject to judge its relevance and its place in the informatics curriculum. Third, we were able to ask lecturers about their experience with TCS as a student and last, but not least, a questionnaire for lecturers enabled us to use a more international group of participants. This prevented our conclusions from being tied to one particular institute or even to one particular country.

2.1.2 The Questionnaire

The questions on the questionnaire that was used can be divided into two sections: (A) Background Information and (B) Theoretical Computer Science Education. A list of all the questions used on the questionnaire can be found in Appendix B. In the first section participants were asked to provide some information regarding their education, research interests, and their teaching portfolio to allow for their answers to the question in the second section to be interpreted in context. For example, it is possible that students are generally more motivated in a course on algorithms than they are in a course on logic. If context-data like those described above is available, we will be able to better interpret the data.

In the second section, there are five general topics on which the participants' opinion is sought. We will discuss each one in turn:

- (a) the *participant's* enjoyment and perceived relevance of TCS;
- (b) the participants' informed opinion of the *students'* enjoyment, difficulty, and perceived relevance of TCS;
- (c) ideas for improving student motivation on TCS subjects and, where appropriate, the participant's experience with them;
- (d) the participant's experience with TCS as a student and possible differences with the current situation;
- (e) the participant's experience with and opinion on learning by game-building.

Starting with (a), we are interested in the level of enjoyment the participant experiences when teaching TCS topics, how relevant they perceive TCS to be in an undergraduate degree in computer science, and why. The answers to these question will provide some insight into the factors which cause lecturers to (dis)like teaching TCS and whether or not the students' alleged lack of motivation is one of these factors. They will also provide an array of educated opinions on the importance of TCS in an undergraduate TCS curriculum together with accompanying arguments. Topic (a) therefore fulfills the second purpose of this study: to find an answer to the question: (why) should TCS be in the undergraduate TCS curriculum? Topic (a) is represented by Questions 8–11.

In addition to their own enjoyment and perceived relevance of TCS, we are also interested in what the lecturers believe the influences of these factors are on the motivation of their students. These questions together with questions on the additional factor of *difficulty*, are presented in Topic (b). The answers to these questions will be used to analyse the learning by game-building approach and possibly improve it. The analysis will consider, for each influence in turn, whether or not learning by game-building addresses it, and if not, if it can be adapted to do so in future applications. Topic (b) is represented by Questions 12–15.

Staying with the same topic, Topic (c) prompts the participant for ideas on improving student motivation for TCS subjects on a factor-by-factor basis and, where appropriate, the participant's experience with them. Some of the ideas provided on this topic can potentially be used in learning by game-building to achieve the improvements mentioned in the previous paragraph. Topic (c) is represented by Questions 18–21.

Topic (d) touches on the slightly different matter of the participant's own experience being a TCS student and they are invited to identify any possible differences with the current situation. Ideally, we would like to find at least one participant who did not like theory as a student, but who ended up enjoying teaching it later in life. This would be an extreme example of a *raison d'être* for our approach learning by game-building, because it illustrates that it is not a futile exercise to attempt to motivate people, because their opinions are not set in stone. Furthermore, we might also find a participant who had trouble with the TCS material as a student, but ended up teaching TCS material later in life, thus proving that even struggling students can become extremely adept at TCS and refuting anyone trying to argue a TCS equivalent of Bornat's argument that teaching programming is a futile exercise (Dehnadi and Bornat, 2004). Topic (d) is represented by Questions 16–17.

Topic (e) enquires after the participant's experience with learning by game-building in TCS, to further ensure that the approach is original. The participant is also given the option of reading a paper on the subject (Korte *et al.*, 2007) and giving their educated opinion on it. Topic (e) is represented by Question 22.

One last question (Question 23) is added to the set of *data questions* in which the participant is invited to provide additional remarks or comments.

2.1.3 The Interface

An online version of the questionnaire was used in an effort to reach an international group of participants. This not only allows for the questionnaire to be easily distributed via mailing lists, but participants might also be tempted to write longer answers if they can type them and they might be more likely to submit, because it all requires less effort. Furthermore, participants cannot easily misplace their copy of the questionnaire and sending out reminders for people to finish their questionnaire is straightforward.

The interface was written in Python, because it has good support for CGI or Common Gateway Interface, which allows for communication through HTML-forms. The layout of the interface is fairly standard except for the ‘SAVE’- and ‘EDIT’ buttons next to each individual question. This design-decision was based on our personal experience with interfaces and our frustration carefully filling out long forms, only to lose all your data when unexpectedly the browser crashes. However, the use of the individual buttons is not mandatory, because there is also a traditional ‘SAVE ALL’ button at the bottom of the questionnaire.

Figures 2.1 and 2.2 show four screen shots of the interface of the questionnaire. Screen shots 1 and 2 show the top and bottom of the main questionnaire of the pilot study, screen shot 3 shows the login page, and screen shot 4 shows the exit page which is displayed after the participant presses the ‘SAVE ALL & EXIT’ button.

In order to identify any potential problems with the questionnaire, a pilot study was designed.

2.1.4 Pilot Study

In the pilot study, our participants were university lecturers who are currently teaching theoretical computer science at the undergraduate level, or have done so in the past. In particular, they are members of the Laboratory of the Foundations of Computer Science at the University of Edinburgh and people who are on one of their mailing lists. The mailing lists were used for recruitment of participants.

The pilot questionnaire consisted of two parts: (a) the *data questions* and (b) the *meta-questions*, i.e. questions about the questionnaire itself. The former formed the basis for the questionnaire used in the main study, whereas the answers to the latter were used to improve the data questions and design of the interface. There were two general areas for which feedback was required: (a) the questions and (b) the interface. On the questionnaire, one question was formulated for each area, plus one more ques-

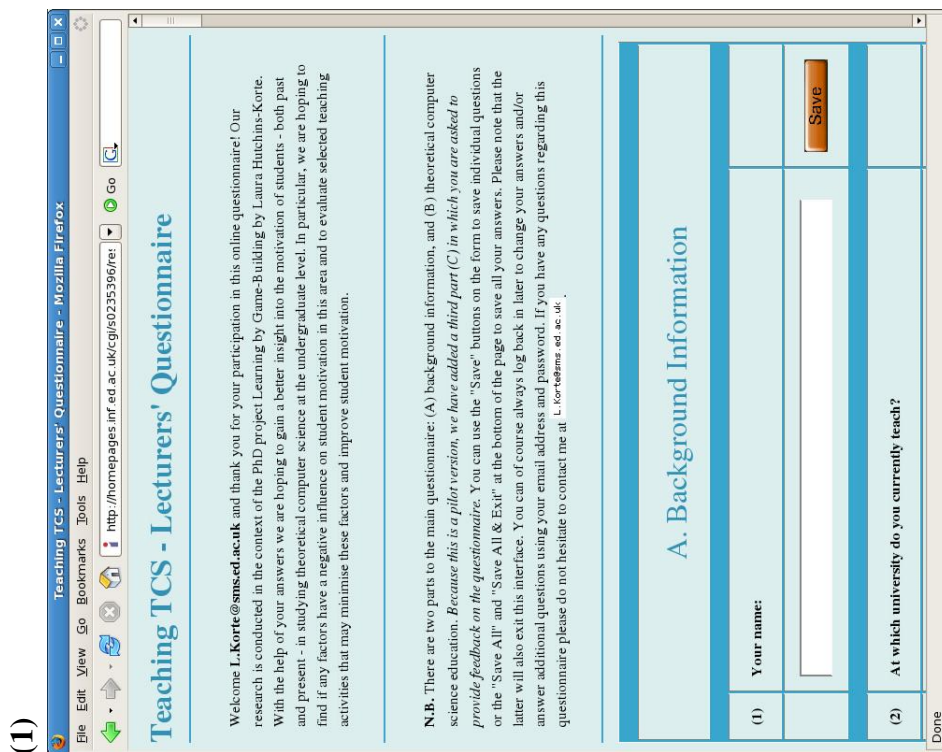
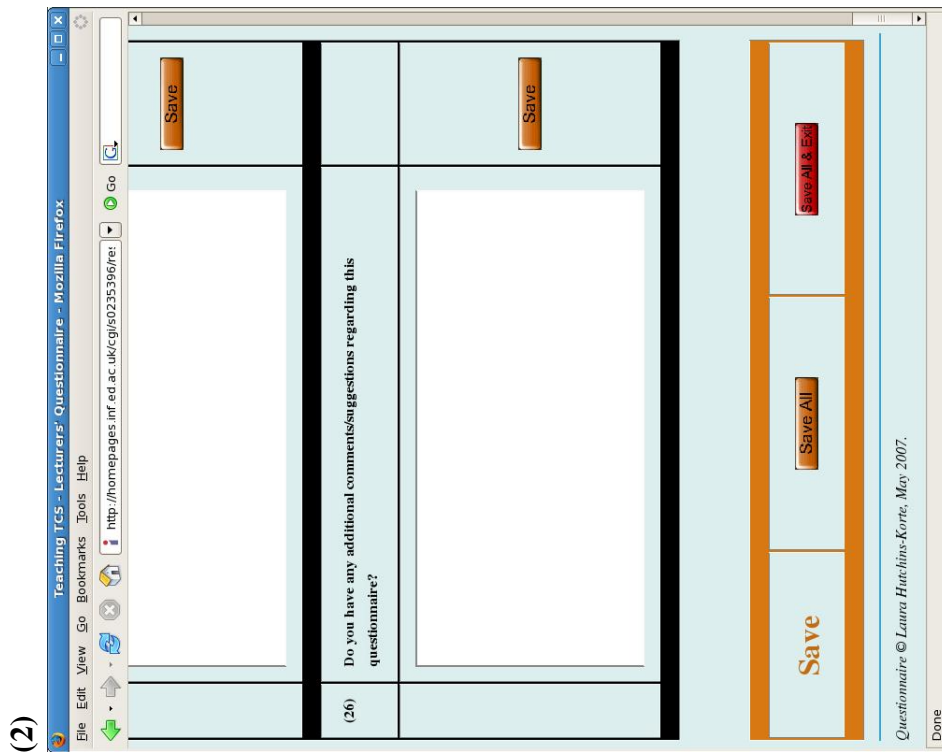


Figure 2.1: The Pilot Questionnaire: Screen Shots 1 and 2

(3)

Teaching TCS - Lecturers' Questionnaire - Login

This questionnaire is aimed at university lecturers who are teaching theoretical computer science (TCS) topics at an undergraduate level or have done so in the past.

Our research is conducted in the context of the PhD project Learning by Game-Building by Laura Hutchins-Korte. With the help of your answers we are hoping to gain a better insight into the motivation of students - both past and present - in studying theoretical computer science at the undergraduate level. In particular, we are hoping to find if any factors have a negative influence on student motivation in this area and to evaluate selected teaching activities that may minimise these factors and improve student motivation.

Email Address: Request Password

Password: Login

Questionnaire © Laura Hutchins-Korte, May 2007.

Done

(4)

Teaching TCS - Lecturers' Questionnaire - Mozilla Firefox

Your answers have been saved.

You have not yet answered question(s) 4, 11, 18 and 26.

Please remember that you can [log back in](#) at any time to fill in the questions you have not answered yet, or change your answer to any of the questions.

Thank you for participating!

Questionnaire © Laura Hutchins-Korte, May 2007.

Done

Figure 2.2: The Pilot Questionnaire: Screen Shots 3 and 4

tion for any additional feedback the participants might want to provide. Appendix A shows the complete questionnaire used in the pilot of Study 1.

As a result of the two emails that were sent out to the mailing lists, 9 passwords were requested, 5 participants started filling out the questionnaire and completed some of the questions. Passwords were requested in the 5 days following the sending out of the emails and participants were answering questions on the questionnaire up to a week after the emails were sent out. This information was used to inform the design of a timeline for the study described in this chapter.

After two weeks, 3 participants out of the 9 who had requested a password had completed the questionnaire. Figure 2.3 shows all the answers that were received on the three evaluation questions. The first and second column contain the evaluation question and the answers provided by the participants; the third and fourth column contain our comments on the participants' answers and any actions that needed to be taken as a results of the answers. The table shows that the feedback that was received was very diverse.

2.1.4.1 Feedback

First of all, one participant mentioned that it was unclear what the purpose of the questionnaire was. It was argued that since it contains only open-ended questions, it will not be possible to do any statistical analysis on the data collected. The options for quantitative analysis are indeed very limited when working with open-ended questions, but the possibilities of qualitative analysis are endless. We intend to pursue the latter kind of analysis. Furthermore, a short statement of purpose is included at the top of the questionnaire and the login screen:

With the help of your answers we are hoping to gain a better insight into the motivation of students - both past and present - in studying theoretical computer science at the undergraduate level. In particular, we are hoping to find if any factors have a negative influence on student motivation in this area and to evaluate selected teaching activities that may minimise these factors and improve student motivation.

A second comment we received was that Question 6 requires the participant to remember exactly what they were teaching 10–20 years ago. This is a valid comment and a full teaching portfolio is not strictly necessary for our research, so we decided to ask for the past 5 years only. This should give us an idea of the kind of teaching

Question	Participant's Answer	Researcher's Comments	Action
<p>24. Were there any questions you found particularly difficult to answer? If so, please specify which questions and explain why you found them difficult to answer.</p>	<p>a. I wasn't clear what the purpose was. Since these are open-form questions, it won't be possible to do any statistical analysis to verify or refute a hypothesis.</p> <p>b. Questions requiring me to remember what I was teaching 10 or 20 years ago.</p> <p>c. Not really, but TCS is a broad area and many of the questions have different answers, depending on the topic.</p>	<p>Participant is not familiar with qualitative research.</p> <p>Valid.</p> <p>We anticipate as much.</p>	<p>None.</p> <p>Change Question 6 to cover only education in the past 5 years.</p> <p>None.</p>
<p>25. Did you have any trouble using the interface? Please describe any problems you had.</p>	<p>a. I was put off by the need to identify myself by e-mail and verify this by password. Why do you want my email address and name? And if you don't trust me to provide a correct e-mail, why are you asking me questions?</p> <p>b. It is unusual to have a save button next to every field, so you might explain this.</p>	<p>Valid.</p> <p>Information already present at top of questionnaire.</p>	<p>Provide ground for requesting email address on the login page.</p> <p>None.</p>
<p>26. Do you have any additional comments/suggestions regarding this questionnaire?</p>	<p>a. The questionnaire is too long. You should either drop some questions, or at the very least, give people an indication of the amount of time they will be spending completing the questionnaire.</p>	<p>Valid.</p>	<p>(1) Question 6 has already been shortened, (2) shorten Question 4, and (3) add time estimate of 20–30 minutes at top of questionnaire.</p>

Figure 2.3: Feedback from Pilot Study

material the participant is most familiar with, without participants having to spend too much time trying to remember what all they have taught in the past 20 or more years.

A third comment pointed out the fact that TCS is a very broad area and many of the questions will have different answers depending on the topic within TCS. This is a valid comment. However, we anticipated this and this is exactly the reason we are asking participants to give us their education portfolio (Question 6), research interests (Question 5), and the universities they have taught TCS at the undergraduate level (Question 4). This last question is important because there may be a difference in research-oriented universities versus teaching-oriented universities.

One participant was put off by the fact that one needs to identify oneself by email and verify this by password. It seems they were under the impression this functionality was set up to verify the identity of the participant, which is a misunderstanding. In fact, the functionality is set up so that participants are not required to fill in all their answers in one session, but can log back in later to complete any questions they had not had time for the first time around. To prevent any further participants from having the same misunderstanding, we decided to add the following paragraph to the login page:

"Please provide a valid email address so that we can send you a password, which will allow you to log back in at a later time to answer additional questions or change any of your answers. If you have any questions regarding this questionnaire please do not hesitate to contact me at L.Korte@sms.ed.ac.uk."

With regard to the interface, we received a comment that it is unusual to have a save button next to each field and that we might explain this. It is indeed not standard practice to have a save-button next to each field. The decision to go with this non-standard feature was based on our personal experience with interfaces and our frustration when the browser crashes after having filled in half or even the entire form. However, it is not mandatory to use this feature and a short explanation of its use *is* included at the top of the questionnaire:

You can use the 'SAVE' buttons on the form to save individual questions or the 'SAVE ALL' and 'SAVE ALL & EXIT' at the bottom of the page to save all your answers. Please note that the latter will also exit this interface.

The final complaint we received was that the questionnaire was too long. The participant suggested we either drop some questions and at the very least give future

participants an indication of the amount of time they will need to spend filling out the questionnaire. This is a valid criticism. The questionnaire is indeed fairly long and we have not given any indication of how long it takes to fill it out. To remedy this shortcoming, we decided to include an estimate of 20–30 minutes at the top of the questionnaire. The participant is also reminded that it is not necessary to fill all the questions out at once, and they are encouraged to log back in at their convenience to complete the questionnaire.

Furthermore, after carefully considering each question for deletion – using the answers to the 23 data questions provided by our pilot participants as an aid – we found that none of the questions were less than essential. We did however bring down the question on research interests (Question 4) from 10 lines to a mere 3 lines and the question on education portfolio (Question 6) had previously been shortened to address one of the other comments.

2.1.5 Hypotheses

Based on the literature we expect to find that students are less motivated on TCS than they are on their other informatics subjects. In particular, we expect to be able to validate the alleged reasons for this lack of motivation: (a) difficulty, (b) lack of enjoyment, and (c) lack of relevance.

We also expect participants to consider it important to have TCS on the undergraduate curriculum. This expectation is based on the ACM Curriculum (Joint Task Force for Computing Curricula 2005, 2005), the fact that many universities offer one or more undergraduate modules on the topic, and the self-selecting nature of our participants. Most people consider their job to be of importance, so we would naturally expect those who teach TCS to value TCS education. The interesting part of their answer will be the *reasons* they provide for the importance of TCS.

Because an open mind is imperative in analysing open-ended questions, we decided not to hypothesize over the nature of these reasons. Instead, we will let the results of the survey speak for themselves.

Finally, the most important topics at undergraduate level are expected to include some the following:

- Finite State Automata / Regular Expressions
- Grammars / Push-Down Automata

- Turing Machines / Computability / Halting Problem / Rice's Theorem
- Logic
- Complexity Theory / Cook-Levin Theorem / NP-Completeness
- Concurrency / CCS

This expectation is based on informal examination of undergraduate courses and the ACM curriculum (Joint Task Force for Computing Curricula 2005, 2005).

2.2 Method

2.2.1 Participants

Participants are university lecturers who are currently teaching TCS topics at the undergraduate level or have done so in the past.

2.2.2 Materials

An online survey with 23 open-ended questions was used. See Appendix B for the complete survey and Figures 2.1 and 2.2 for screen shots of the interface.

2.2.3 Procedure

For the main study two new mailing lists were chosen as a target for the revised questionnaire: (1) the general mailing list of the ACM Special Interest Group on Computer Science Education (SIGCSE) and (2) the discussion mailing list of the Psychology of Programming Interest Group (PPIG). Both these lists are international and reach a wide range of academics from diverse settings.

Based on the pilot study which revealed that there was no activity on the questionnaire site after one week, a decision was made to send out three successive emails. The first one contained the initial request to lecturers teaching TCS at the undergraduate level to fill out the questionnaire. The second email was sent out one week after the first one and contained a further request to fill out the questionnaire. The third and final email was sent out a week after the second one and mainly targeted those people who had started the questionnaire, but had not yet managed to finish it. The email stated that participants are given one more week to complete the questionnaire after which

the site was closed. This brought the total data collection time of the study to three weeks.

At the end of those three weeks, the online questionnaire was closed down and the participants' answers consolidated. The results presented in the next section are the result of careful analysis of the participants' answers based on techniques for qualitative data analysis described in Miles and M. (1994). The analysis was performed on a question-by-question basis and comprised four stages:

1. Read through all the answers to identify common factors.
2. Read through all the answers again to identify factors that cannot be classified under any of the common ones.
3. Sort the factors into groups and subgroups, and assign each of the (sub)groups a unique label.
4. Annotate the answers with the unique labels.
5. Determine the frequency of each label to form a single section in the table on Page 21–30.

2.3 Results

In total 25 people participated in our study, but only 20 of them filled in all the questions on the questionnaire. We have used the data from all 25 participants where available, but some of the later answers are based on as little as 20 participants. The results of the survey are shown in the table on Page 21–30. In this table, there is exactly one section devoted to each of the question on the survey. Quantities are given in numbers, rather than percentages, to make the data set transparent. Moreover, quantities are given out of the total number of answers received to that particular question. For example, in Question 9, 25 answers were received in total, 20 of those said that it is important to teach theoretical computer science to undergraduates; 4 of those said that it depends; one of those said that it is moderately important, and nobody said that it is unimportant.

For some questions, including Question 9, these answers are then divided into more specific subcategories. For example, in Question 9 the answer 'It depends ...' is divided into two subcategories: (a) '... on the exact course students are taking' and

(b) ‘...on the topics within TCS’. Quantities are given out of the number of responses in the parent category, so out of the 4 people who answers ‘It depends ...’, 2 said it depends on the exact course students are taking and 2 said it depends on the topics within TCS. Nested subdivision is used for some of the questions, e.g. Question 8.

It is possible for the sum of the quantities of the subcategories to be larger than the number of responses of the parent category. This means that at least one participant gave more than one specification or justification for that particular answer. For example, the sum of the quantities of the subcategories of the first item of Question 9 is $10 + 7 + 7 + 1 = 25$, which is larger than the number of responses of the parent category, 20. This will have happened because at least three participants gave more than one reason for TCS being important to undergraduates.

For some of the questions, answers have not been displayed. Reason for this include: (a) the privacy of our participants, (b) unusable answers due to diverse interpretation of a question, and (c) no information relevant to the study was provided by participants.

1.	<i>Your name:</i>																									
Personal information undisclosed.																										
2.	<i>At which university do you currently teach?</i>																									
10 / 25	European University																									
14 / 25	North American University																									
1 / 25	Other University																									
3.	<i>Which undergraduate/postgraduate degrees do you hold? For each degree please specify the subject, kind of degree (BSc, MA, PhD, etc.), and the university it was received from.</i>																									
Personal information undisclosed.																										
4.	<i>Please describe your research interests in a maximum of 3 lines.</i>																									
13 / 25	<p>Formal Methods</p> <table border="1"> <tr> <td>8 / 13</td> <td>Maths</td> </tr> <tr> <td>4 / 13</td> <td>discrete maths/graph theory</td> </tr> <tr> <td>1 / 13</td> <td>constructive mathematics</td> </tr> <tr> <td>2 / 13</td> <td>computational geometry</td> </tr> <tr> <td>1 / 13</td> <td>numerical analysis</td> </tr> <tr> <td>1 / 13</td> <td>history/philosophy of ...</td> </tr> <tr> <td>3 / 13</td> <td>logic</td> </tr> <tr> <td>2 / 13</td> <td>lambda calculus</td> </tr> <tr> <td>2 / 13</td> <td>domain theory/category theory</td> </tr> <tr> <td>2 / 13</td> <td>proof theory</td> </tr> <tr> <td>1 / 13</td> <td>cryptology</td> </tr> <tr> <td>1 / 13</td> <td>general</td> </tr> </table>		8 / 13	Maths	4 / 13	discrete maths/graph theory	1 / 13	constructive mathematics	2 / 13	computational geometry	1 / 13	numerical analysis	1 / 13	history/philosophy of ...	3 / 13	logic	2 / 13	lambda calculus	2 / 13	domain theory/category theory	2 / 13	proof theory	1 / 13	cryptology	1 / 13	general
8 / 13	Maths																									
4 / 13	discrete maths/graph theory																									
1 / 13	constructive mathematics																									
2 / 13	computational geometry																									
1 / 13	numerical analysis																									
1 / 13	history/philosophy of ...																									
3 / 13	logic																									
2 / 13	lambda calculus																									
2 / 13	domain theory/category theory																									
2 / 13	proof theory																									
1 / 13	cryptology																									
1 / 13	general																									
10 / 25	Education																									
	<table border="1"> <tr> <td>7 / 10</td> <td>computer science</td> </tr> <tr> <td>1 / 10</td> <td>robots</td> </tr> <tr> <td>1 / 10</td> <td>gender issues</td> </tr> <tr> <td>1 / 10</td> <td>programming</td> </tr> <tr> <td>1 / 10</td> <td>digital</td> </tr> </table>		7 / 10	computer science	1 / 10	robots	1 / 10	gender issues	1 / 10	programming	1 / 10	digital														
7 / 10	computer science																									
1 / 10	robots																									
1 / 10	gender issues																									
1 / 10	programming																									
1 / 10	digital																									
8 / 25	Programming Languages																									
	<table border="1"> <tr> <td>4 / 8</td> <td>program verification/logic</td> </tr> <tr> <td>3 / 8</td> <td>declarative programming (functional/logic)</td> </tr> <tr> <td>2 / 8</td> <td>type theory/specification</td> </tr> <tr> <td>1 / 8</td> <td>semantics</td> </tr> <tr> <td>1 / 8</td> <td>non-linear</td> </tr> <tr> <td>1 / 8</td> <td>general</td> </tr> <tr> <td>1 / 8</td> <td>storage</td> </tr> </table>		4 / 8	program verification/logic	3 / 8	declarative programming (functional/logic)	2 / 8	type theory/specification	1 / 8	semantics	1 / 8	non-linear	1 / 8	general	1 / 8	storage										
4 / 8	program verification/logic																									
3 / 8	declarative programming (functional/logic)																									
2 / 8	type theory/specification																									
1 / 8	semantics																									
1 / 8	non-linear																									
1 / 8	general																									
1 / 8	storage																									

6	/	25	Applications	<table border="1"> <tbody> <tr> <td>2</td> <td>/</td> <td>6</td> <td>Games</td> </tr> <tr> <td>2</td> <td>/</td> <td>6</td> <td>scientific computing (incl. mathematical sw)</td> </tr> <tr> <td>1</td> <td>/</td> <td>6</td> <td>software engineering</td> </tr> <tr> <td>1</td> <td>/</td> <td>6</td> <td>image processing</td> </tr> <tr> <td>1</td> <td>/</td> <td>6</td> <td>Natural Language Processing</td> </tr> <tr> <td>1</td> <td>/</td> <td>6</td> <td>interplay with theory</td> </tr> </tbody> </table>	2	/	6	Games	2	/	6	scientific computing (incl. mathematical sw)	1	/	6	software engineering	1	/	6	image processing	1	/	6	Natural Language Processing	1	/	6	interplay with theory				
2	/	6	Games																													
2	/	6	scientific computing (incl. mathematical sw)																													
1	/	6	software engineering																													
1	/	6	image processing																													
1	/	6	Natural Language Processing																													
1	/	6	interplay with theory																													
5	/	25	Algorithms	<table border="1"> <tbody> <tr> <td>2</td> <td>/</td> <td>5</td> <td>general</td> </tr> <tr> <td>1</td> <td>/</td> <td>5</td> <td>approximation</td> </tr> <tr> <td>1</td> <td>/</td> <td>5</td> <td>complexity</td> </tr> <tr> <td>1</td> <td>/</td> <td>5</td> <td>randomized</td> </tr> <tr> <td>1</td> <td>/</td> <td>5</td> <td>visualization</td> </tr> <tr> <td>1</td> <td>/</td> <td>5</td> <td>data structures and ...</td> </tr> <tr> <td>1</td> <td>/</td> <td>5</td> <td>graph algorithms</td> </tr> </tbody> </table>	2	/	5	general	1	/	5	approximation	1	/	5	complexity	1	/	5	randomized	1	/	5	visualization	1	/	5	data structures and ...	1	/	5	graph algorithms
2	/	5	general																													
1	/	5	approximation																													
1	/	5	complexity																													
1	/	5	randomized																													
1	/	5	visualization																													
1	/	5	data structures and ...																													
1	/	5	graph algorithms																													
4	/	25	AI	<table border="1"> <tbody> <tr> <td>2</td> <td>/</td> <td>4</td> <td>machine learning</td> </tr> <tr> <td>2</td> <td>/</td> <td>4</td> <td>automated reasoning/problem solving</td> </tr> <tr> <td>2</td> <td>/</td> <td>4</td> <td>bioinformatics</td> </tr> <tr> <td>1</td> <td>/</td> <td>4</td> <td>evolutionary algorithms</td> </tr> <tr> <td>1</td> <td>/</td> <td>4</td> <td>robotics</td> </tr> </tbody> </table>	2	/	4	machine learning	2	/	4	automated reasoning/problem solving	2	/	4	bioinformatics	1	/	4	evolutionary algorithms	1	/	4	robotics								
2	/	4	machine learning																													
2	/	4	automated reasoning/problem solving																													
2	/	4	bioinformatics																													
1	/	4	evolutionary algorithms																													
1	/	4	robotics																													
3	/	25	Social Computing	<table border="1"> <tbody> <tr> <td>2</td> <td>/</td> <td>3</td> <td>ethics</td> </tr> <tr> <td>1</td> <td>/</td> <td>3</td> <td>accessibility</td> </tr> </tbody> </table>	2	/	3	ethics	1	/	3	accessibility																				
2	/	3	ethics																													
1	/	3	accessibility																													
5.	<i>At which universities have you taught TCS topics at the undergraduate level?</i>			Personal information undisclosed.																												
6.	<i>Education Portfolio: Which TCS topics have you taught at the undergraduate level in the past 5 years? Which other computing topics have you taught? Which other subjects have you taught?</i>																															
20	/	25	Theory of Computation																													
17	/	25	Algorithms																													
11	/	25	Programming																													
11	/	25	Mathematical Foundations of CS																													
10	/	25	Data Structures / Knowledge Representation																													
8	/	25	Foundations of Programming Languages																													
7	/	25	Transkills																													
6	/	25	Logic																													

4	/	25	CS1 or CS2																																								
4	/	25	AI																																								
3	/	25	Compilers																																								
3	/	25	Databases																																								
3	/	25	Concurrent/Parallel Programming (Distributed Systems)																																								
3	/	25	Cryptography																																								
2	/	25	Ethics / History																																								
2	/	25	Computer Organisation																																								
2	/	25	Software Engineering																																								
2	/	25	Operating Systems																																								
2	/	25	Games and Simulations																																								
2	/	25	Formal methods																																								
1	/	25	NLP																																								
1	/	25	Networking																																								
1	/	25	Vision/Image Processing																																								
1	/	25	Bio-Informatics																																								
1	/	25	HCI																																								
7.	<i>Of the subjects that you have taught, which do you prefer teaching? (Please give any reasons for this preference)</i>																																										
21	/	25	Mention affinity with material as a reason.																																								
7	/	25	Mention importance / usefulness of the material as a reason.																																								
7	/	25	Mention student reactions as a reason.																																								
7 / 7 Positive Reactions.																																											
2	/	25	Mention games, because of their explanatory power, strong student engagement, and social relevance as a reason.																																								
8.	<i>Do you like teaching TCS topics? If so, why? If not, why not?</i>																																										
25	/	25	Yes, ...																																								
<table style="width: 100%; border-collapse: collapse;"> <tbody> <tr> <td style="width: 10%;">20</td> <td style="width: 5%;">/</td> <td style="width: 10%;">25</td> <td>...because of affinity with subject / close to research interests.</td> </tr> <tr> <td>7</td> <td>/</td> <td>25</td> <td>...and mention student reactions.</td> </tr> <tr> <td colspan="4" style="text-align: center; border: 1px solid black; padding: 5px;"> <table style="width: 100%; border-collapse: collapse;"> <tbody> <tr> <td style="width: 10%;">5</td> <td style="width: 5%;">/</td> <td style="width: 10%;">7</td> <td>Negative Reactions (e.g. difficulty, enjoyment)</td> </tr> <tr> <td colspan="4" style="text-align: center; border: 1px solid black; padding: 5px;"> <table style="width: 100%; border-collapse: collapse;"> <tbody> <tr> <td style="width: 10%;">3</td> <td style="width: 5%;">/</td> <td style="width: 10%;">5</td> <td>Consider this a teaching challenge.</td> </tr> <tr> <td>2</td> <td>/</td> <td>5</td> <td>Like teaching TCS despite the reactions.</td> </tr> </tbody> </table> </td> </tr> <tr> <td colspan="4" style="text-align: center;">2 / 7 Positive Reactions</td> </tr> </tbody> </table> </td> </tr> <tr> <td>2</td> <td>/</td> <td>25</td> <td>...but only when students are motivated / take it as an elective.</td> </tr> <tr> <td>1</td> <td>/</td> <td>25</td> <td>...because of affinity with teaching in general.</td> </tr> </tbody> </table>				20	/	25	...because of affinity with subject / close to research interests.	7	/	25	...and mention student reactions.	<table style="width: 100%; border-collapse: collapse;"> <tbody> <tr> <td style="width: 10%;">5</td> <td style="width: 5%;">/</td> <td style="width: 10%;">7</td> <td>Negative Reactions (e.g. difficulty, enjoyment)</td> </tr> <tr> <td colspan="4" style="text-align: center; border: 1px solid black; padding: 5px;"> <table style="width: 100%; border-collapse: collapse;"> <tbody> <tr> <td style="width: 10%;">3</td> <td style="width: 5%;">/</td> <td style="width: 10%;">5</td> <td>Consider this a teaching challenge.</td> </tr> <tr> <td>2</td> <td>/</td> <td>5</td> <td>Like teaching TCS despite the reactions.</td> </tr> </tbody> </table> </td> </tr> <tr> <td colspan="4" style="text-align: center;">2 / 7 Positive Reactions</td> </tr> </tbody> </table>				5	/	7	Negative Reactions (e.g. difficulty, enjoyment)	<table style="width: 100%; border-collapse: collapse;"> <tbody> <tr> <td style="width: 10%;">3</td> <td style="width: 5%;">/</td> <td style="width: 10%;">5</td> <td>Consider this a teaching challenge.</td> </tr> <tr> <td>2</td> <td>/</td> <td>5</td> <td>Like teaching TCS despite the reactions.</td> </tr> </tbody> </table>				3	/	5	Consider this a teaching challenge.	2	/	5	Like teaching TCS despite the reactions.	2 / 7 Positive Reactions				2	/	25	...but only when students are motivated / take it as an elective.	1	/	25	...because of affinity with teaching in general.
20	/	25	...because of affinity with subject / close to research interests.																																								
7	/	25	...and mention student reactions.																																								
<table style="width: 100%; border-collapse: collapse;"> <tbody> <tr> <td style="width: 10%;">5</td> <td style="width: 5%;">/</td> <td style="width: 10%;">7</td> <td>Negative Reactions (e.g. difficulty, enjoyment)</td> </tr> <tr> <td colspan="4" style="text-align: center; border: 1px solid black; padding: 5px;"> <table style="width: 100%; border-collapse: collapse;"> <tbody> <tr> <td style="width: 10%;">3</td> <td style="width: 5%;">/</td> <td style="width: 10%;">5</td> <td>Consider this a teaching challenge.</td> </tr> <tr> <td>2</td> <td>/</td> <td>5</td> <td>Like teaching TCS despite the reactions.</td> </tr> </tbody> </table> </td> </tr> <tr> <td colspan="4" style="text-align: center;">2 / 7 Positive Reactions</td> </tr> </tbody> </table>				5	/	7	Negative Reactions (e.g. difficulty, enjoyment)	<table style="width: 100%; border-collapse: collapse;"> <tbody> <tr> <td style="width: 10%;">3</td> <td style="width: 5%;">/</td> <td style="width: 10%;">5</td> <td>Consider this a teaching challenge.</td> </tr> <tr> <td>2</td> <td>/</td> <td>5</td> <td>Like teaching TCS despite the reactions.</td> </tr> </tbody> </table>				3	/	5	Consider this a teaching challenge.	2	/	5	Like teaching TCS despite the reactions.	2 / 7 Positive Reactions																							
5	/	7	Negative Reactions (e.g. difficulty, enjoyment)																																								
<table style="width: 100%; border-collapse: collapse;"> <tbody> <tr> <td style="width: 10%;">3</td> <td style="width: 5%;">/</td> <td style="width: 10%;">5</td> <td>Consider this a teaching challenge.</td> </tr> <tr> <td>2</td> <td>/</td> <td>5</td> <td>Like teaching TCS despite the reactions.</td> </tr> </tbody> </table>				3	/	5	Consider this a teaching challenge.	2	/	5	Like teaching TCS despite the reactions.																																
3	/	5	Consider this a teaching challenge.																																								
2	/	5	Like teaching TCS despite the reactions.																																								
2 / 7 Positive Reactions																																											
2	/	25	...but only when students are motivated / take it as an elective.																																								
1	/	25	...because of affinity with teaching in general.																																								

0	/	25	No.
9.	<i>In undergraduate computing degrees, how important do you think it is to teach TCS topics? Please explain your answer.</i>		
20	/	25	Important, because ...
10	/	20	... it teaches general reasoning skills.
7	/	20	... of its historical importance / because it is part of informatics field.
7	/	20	... knowledge is needed for degree / career.
1	/	20	... no reason provided.
4	/	25	It Depends ...
2	/	4	... on the exact course students are taking.
2	/	4	... on the topics within TCS.
1	/	25	Moderately Important ...
1	/	1	... teach only material directly relevant to degree / career.
0	/	25	Unimportant
10.	<i>Which TCS topics do you feel are the most important to teach at the undergraduate level? Please explain your answer.</i>		
12	/	23	Chomsky's Hierarchy
8	/	12	Automata / Regular Expressions
8	/	12	Turing Machines / Computability / Halting Problem / Rice's Theorem
7	/	12	Grammars / Push-Down Automata
2	/	12	Relations between Classes of Hierarchy
12	/	23	Computation/Maths/Logic
7	/	12	Algorithms
4	/	12	Logic
4	/	12	Complexity Theory/Cook-Levin theorem/NP-Completeness
2	/	12	Foundation of Programming Languages, Compilers, Typing, Concurrency
1	/	12	Data Structures
1	/	12	Discrete Math
1	/	12	Asymptotic Analysis
1	/	12	Timely Topics: Genetic Algorithms & Schema Theorem, Quantum Computing, etc.

3	/	23	Programming	<table border="1"> <tbody> <tr> <td>1</td> <td>/</td> <td>3</td> <td>Functional Programming</td> </tr> <tr> <td>1</td> <td>/</td> <td>3</td> <td>Logic Programming</td> </tr> <tr> <td>1</td> <td>/</td> <td>3</td> <td>Dynamic Programming</td> </tr> </tbody> </table>	1	/	3	Functional Programming	1	/	3	Logic Programming	1	/	3	Dynamic Programming								
1	/	3	Functional Programming																					
1	/	3	Logic Programming																					
1	/	3	Dynamic Programming																					
5	/	23	Other	<table border="1"> <tbody> <tr> <td>3</td> <td>/</td> <td>5</td> <td>Topics are irrelevant. The purpose of TCS is to teach problem solving skills.</td> </tr> <tr> <td>1</td> <td>/</td> <td>5</td> <td>The area is not divisible; its breadth is important.</td> </tr> <tr> <td>1</td> <td>/</td> <td>5</td> <td>There is no answer (I am too old).</td> </tr> </tbody> </table>	3	/	5	Topics are irrelevant. The purpose of TCS is to teach problem solving skills.	1	/	5	The area is not divisible; its breadth is important.	1	/	5	There is no answer (I am too old).								
3	/	5	Topics are irrelevant. The purpose of TCS is to teach problem solving skills.																					
1	/	5	The area is not divisible; its breadth is important.																					
1	/	5	There is no answer (I am too old).																					
11.	Which TCS topics do you feel are less important to teach at the undergraduate level? Please explain your answer.																							
6	/	25	Computability / Complexity	<table border="1"> <tbody> <tr> <td>2</td> <td>/</td> <td>6</td> <td>Advanced Computability Issues.</td> </tr> <tr> <td>2</td> <td>/</td> <td>6</td> <td>Advanced Complexity Issues.</td> </tr> <tr> <td>1</td> <td>/</td> <td>6</td> <td>Computability.</td> </tr> <tr> <td>1</td> <td>/</td> <td>6</td> <td>Complexity.</td> </tr> <tr> <td>1</td> <td>/</td> <td>6</td> <td>Decidibility.</td> </tr> </tbody> </table>	2	/	6	Advanced Computability Issues.	2	/	6	Advanced Complexity Issues.	1	/	6	Computability.	1	/	6	Complexity.	1	/	6	Decidibility.
2	/	6	Advanced Computability Issues.																					
2	/	6	Advanced Complexity Issues.																					
1	/	6	Computability.																					
1	/	6	Complexity.																					
1	/	6	Decidibility.																					
3	/	25	Proofs	<table border="1"> <tbody> <tr> <td>2</td> <td>/</td> <td>3</td> <td>Proofs of common topics.</td> </tr> <tr> <td>1</td> <td>/</td> <td>3</td> <td>Correctness Proofs.</td> </tr> <tr> <td>1</td> <td>/</td> <td>3</td> <td>Proof Production, provided there <i>is</i> understanding of proofs.</td> </tr> </tbody> </table>	2	/	3	Proofs of common topics.	1	/	3	Correctness Proofs.	1	/	3	Proof Production, provided there <i>is</i> understanding of proofs.								
2	/	3	Proofs of common topics.																					
1	/	3	Correctness Proofs.																					
1	/	3	Proof Production, provided there <i>is</i> understanding of proofs.																					
8	/	25	Other Abstract Topics	<table border="1"> <tbody> <tr> <td>2</td> <td>/</td> <td>8</td> <td>FSAs</td> </tr> <tr> <td>3</td> <td>/</td> <td>8</td> <td>Formal (Specification) Languages / Compiler Theory / Type Theory</td> </tr> <tr> <td>1</td> <td>/</td> <td>8</td> <td>Category Theory</td> </tr> <tr> <td>1</td> <td>/</td> <td>8</td> <td>Formal Axiomatization of Classical Logic</td> </tr> <tr> <td>1</td> <td>/</td> <td>8</td> <td>Advanced Algorithms</td> </tr> </tbody> </table>	2	/	8	FSAs	3	/	8	Formal (Specification) Languages / Compiler Theory / Type Theory	1	/	8	Category Theory	1	/	8	Formal Axiomatization of Classical Logic	1	/	8	Advanced Algorithms
2	/	8	FSAs																					
3	/	8	Formal (Specification) Languages / Compiler Theory / Type Theory																					
1	/	8	Category Theory																					
1	/	8	Formal Axiomatization of Classical Logic																					
1	/	8	Advanced Algorithms																					
3	/	25	Other Concrete Topics	<table border="1"> <tbody> <tr> <td>1</td> <td>/</td> <td>3</td> <td>PCP</td> </tr> <tr> <td>1</td> <td>/</td> <td>3</td> <td>Oracle Machines</td> </tr> <tr> <td>1</td> <td>/</td> <td>3</td> <td>Genetic Algorithms</td> </tr> <tr> <td>1</td> <td>/</td> <td>3</td> <td>Games</td> </tr> </tbody> </table>	1	/	3	PCP	1	/	3	Oracle Machines	1	/	3	Genetic Algorithms	1	/	3	Games				
1	/	3	PCP																					
1	/	3	Oracle Machines																					
1	/	3	Genetic Algorithms																					
1	/	3	Games																					
7	/	25	Miscellaneous	<table border="1"> <tbody> <tr> <td>4</td> <td>/</td> <td>7</td> <td>I do not know / I politely refuse to answer.</td> </tr> <tr> <td>1</td> <td>/</td> <td>7</td> <td>The area is not divisible; its breadth is important.</td> </tr> <tr> <td>1</td> <td>/</td> <td>7</td> <td>There is no answer (I am too old).</td> </tr> </tbody> </table>	4	/	7	I do not know / I politely refuse to answer.	1	/	7	The area is not divisible; its breadth is important.	1	/	7	There is no answer (I am too old).								
4	/	7	I do not know / I politely refuse to answer.																					
1	/	7	The area is not divisible; its breadth is important.																					
1	/	7	There is no answer (I am too old).																					

12.	<i>From your experience, how would you rate students' enjoyment of TCS topics compared to other computing topics? Why do you think students like/dislike TCS topics?</i>											
14 / 23	They dislike it. Reason:											
	<table border="1"> <tr> <td>6 / 14</td> <td>Lack of background / Too difficult.</td> </tr> <tr> <td>6 / 14</td> <td>Lack of relevance.</td> </tr> <tr> <td>2 / 14</td> <td>Too much like maths and students have learned to dislike maths.</td> </tr> <tr> <td>2 / 14</td> <td>Initially negative, but this changes during course.</td> </tr> <tr> <td>2 / 14</td> <td>Undefined.</td> </tr> </table>		6 / 14	Lack of background / Too difficult.	6 / 14	Lack of relevance.	2 / 14	Too much like maths and students have learned to dislike maths.	2 / 14	Initially negative, but this changes during course.	2 / 14	Undefined.
6 / 14	Lack of background / Too difficult.											
6 / 14	Lack of relevance.											
2 / 14	Too much like maths and students have learned to dislike maths.											
2 / 14	Initially negative, but this changes during course.											
2 / 14	Undefined.											
2 / 23	They like it. Comment:											
	<table border="1"> <tr> <td>1 / 2</td> <td>Those who dislike TCS have no mathematical inclination and prefer hands-on material.</td> </tr> <tr> <td>1 / 2</td> <td>TCS has puzzle appeal.</td> </tr> </table>		1 / 2	Those who dislike TCS have no mathematical inclination and prefer hands-on material.	1 / 2	TCS has puzzle appeal.						
1 / 2	Those who dislike TCS have no mathematical inclination and prefer hands-on material.											
1 / 2	TCS has puzzle appeal.											
6 / 23	Depends ...											
	<table border="1"> <tr> <td>3 / 6</td> <td>... on the topic.</td> </tr> <tr> <td>3 / 6</td> <td>... on the student.</td> </tr> </table>		3 / 6	... on the topic.	3 / 6	... on the student.						
3 / 6	... on the topic.											
3 / 6	... on the student.											
1 / 23	All my theories about students are wrong.											
13.	<i>From your experience, how would you rate students' difficulty with TCS topics compared to other computing topics? Why do you think students find TCS topics easy/difficult?</i>											
14 / 22	Difficult. Reason:											
	<table border="1"> <tr> <td>8 / 14</td> <td>Lack of background.</td> </tr> <tr> <td>4 / 14</td> <td>It is difficult.</td> </tr> <tr> <td>3 / 14</td> <td>Undefined.</td> </tr> </table>		8 / 14	Lack of background.	4 / 14	It is difficult.	3 / 14	Undefined.				
8 / 14	Lack of background.											
4 / 14	It is difficult.											
3 / 14	Undefined.											
2 / 22	Average. Comment:											
	<table border="1"> <tr> <td>1 / 2</td> <td>Students have more trouble with large projects than with proofs.</td> </tr> <tr> <td>1 / 2</td> <td>More difficult than some topics, but easiers than others.</td> </tr> </table>		1 / 2	Students have more trouble with large projects than with proofs.	1 / 2	More difficult than some topics, but easiers than others.						
1 / 2	Students have more trouble with large projects than with proofs.											
1 / 2	More difficult than some topics, but easiers than others.											
0 / 22	Easy.											
6 / 22	Depends ...											
	<table border="1"> <tr> <td>2 / 6</td> <td>... on the topic.</td> </tr> <tr> <td>5 / 6</td> <td>... on the student.</td> </tr> </table>		2 / 6	... on the topic.	5 / 6	... on the student.						
2 / 6	... on the topic.											
5 / 6	... on the student.											
14.	<i>From your experience, how would you rate students' perceived relevance of TCS topics to their studies compared to other computing topics? Why do you think students find TCS topics relevant/irrelevant to their studies?</i>											
11 / 21	Not relevant, because ...											
	<table border="1"> <tr> <td>5 / 11</td> <td>... students do not have enough experience to appreciate relevance.</td> </tr> <tr> <td>4 / 11</td> <td>... the connection with the real world is not made explicit.</td> </tr> <tr> <td>3 / 11</td> <td>... it is in fact <i>not relevant</i> to industry jobs.</td> </tr> <tr> <td>1 / 11</td> <td>... undefined.</td> </tr> </table>		5 / 11	... students do not have enough experience to appreciate relevance.	4 / 11	... the connection with the real world is not made explicit.	3 / 11	... it is in fact <i>not relevant</i> to industry jobs.	1 / 11	... undefined.		
5 / 11	... students do not have enough experience to appreciate relevance.											
4 / 11	... the connection with the real world is not made explicit.											
3 / 11	... it is in fact <i>not relevant</i> to industry jobs.											
1 / 11	... undefined.											

5	/	21	Relevant ...	<table border="1"> <tr> <td>4</td> <td>/</td> <td>5</td> <td>... as part of a well-rounded university education.</td> </tr> <tr> <td>1</td> <td>/</td> <td>5</td> <td>... but the majority of our graduates go on to attend graduate school.</td> </tr> </table>	4	/	5	... as part of a well-rounded university education.	1	/	5	... but the majority of our graduates go on to attend graduate school.																				
4	/	5	... as part of a well-rounded university education.																													
1	/	5	... but the majority of our graduates go on to attend graduate school.																													
4	/	21	Depends ...	<table border="1"> <tr> <td>1</td> <td>/</td> <td>4</td> <td>... on the topic.</td> </tr> <tr> <td>3</td> <td>/</td> <td>4</td> <td>... on the instructor.</td> </tr> </table>	1	/	4	... on the topic.	3	/	4	... on the instructor.																				
1	/	4	... on the topic.																													
3	/	4	... on the instructor.																													
4	/	21	Students' perceived relevance is irrelevant.																													
15.	<p><i>Of the three factors influencing student motivation mentioned in questions 12-14 (i.e. enjoyment, difficulty, and relevance), which do you feel is the largest positive (or least negative) influence? Which do you feel is the largest negative (or least positive) influence?</i></p> <p>This question was interpreted in different ways by different participants and the answers therefore discarded.</p>																															
16.	<p><i>What was your opinion of TCS when you were a student? Did you feel you were an exception or the rule at that time?</i></p>																															
12	/	21	Liked it. They said they were ...	<table border="1"> <tr> <td>8</td> <td>/</td> <td>12</td> <td>... an exception.</td> </tr> <tr> <td>0</td> <td>/</td> <td>12</td> <td>... the rule.</td> </tr> <tr> <td>4</td> <td>/</td> <td>12</td> <td>... undefined.</td> </tr> </table>	8	/	12	... an exception.	0	/	12	... the rule.	4	/	12	... undefined.																
8	/	12	... an exception.																													
0	/	12	... the rule.																													
4	/	12	... undefined.																													
3	/	21	Hated it. They said they were ...	<table border="1"> <tr> <td>0</td> <td>/</td> <td>3</td> <td>... an exception.</td> </tr> <tr> <td>2</td> <td>/</td> <td>3</td> <td>... the rule.</td> </tr> <tr> <td>1</td> <td>/</td> <td>3</td> <td>... undefined.</td> </tr> </table>	0	/	3	... an exception.	2	/	3	... the rule.	1	/	3	... undefined.																
0	/	3	... an exception.																													
2	/	3	... the rule.																													
1	/	3	... undefined.																													
2	/	21	Found it difficult. They said they were ...	<table border="1"> <tr> <td>0</td> <td>/</td> <td>2</td> <td>... an exception.</td> </tr> <tr> <td>2</td> <td>/</td> <td>2</td> <td>... the rule.</td> </tr> <tr> <td>0</td> <td>/</td> <td>2</td> <td>... undefined.</td> </tr> </table>	0	/	2	... an exception.	2	/	2	... the rule.	0	/	2	... undefined.																
0	/	2	... an exception.																													
2	/	2	... the rule.																													
0	/	2	... undefined.																													
4	/	21	Not applicable; have not studied informatics as an undergraduate.																													
17.	<p><i>Do you think there is a difference between student opinion of TCS then and now? If so, can you describe this difference? Why do you think opinion changed?</i></p>																															
8	/	21	No change.																													
7	/	21	Change.	<table border="1"> <tr> <td>5</td> <td>/</td> <td>7</td> <td>Negative change, because ...</td> </tr> <tr> <td>3</td> <td>/</td> <td>5</td> <td>... student have too many things going on in their lives. This makes them less willing and able to devote their resources to learning TCS.</td> </tr> <tr> <td>1</td> <td>/</td> <td>5</td> <td>... it is less appealing; more choices are available.</td> </tr> <tr> <td>1</td> <td>/</td> <td>5</td> <td>... student see themselves of consumers of the product that is their education.</td> </tr> <tr> <td>2</td> <td>/</td> <td>7</td> <td>Positive change, because ...</td> </tr> <tr> <td>1</td> <td>/</td> <td>2</td> <td>... informatics is no longer experiencing a boom.</td> </tr> <tr> <td>1</td> <td>/</td> <td>2</td> <td>... other.</td> </tr> </table>	5	/	7	Negative change, because ...	3	/	5	... student have too many things going on in their lives. This makes them less willing and able to devote their resources to learning TCS.	1	/	5	... it is less appealing; more choices are available.	1	/	5	... student see themselves of consumers of the product that is their education.	2	/	7	Positive change, because ...	1	/	2	... informatics is no longer experiencing a boom.	1	/	2	... other.
5	/	7	Negative change, because ...																													
3	/	5	... student have too many things going on in their lives. This makes them less willing and able to devote their resources to learning TCS.																													
1	/	5	... it is less appealing; more choices are available.																													
1	/	5	... student see themselves of consumers of the product that is their education.																													
2	/	7	Positive change, because ...																													
1	/	2	... informatics is no longer experiencing a boom.																													
1	/	2	... other.																													

	5 / 21	Do not know.
	1 / 21	Other.
18.	<i>Which teaching activities do you think could make TCS more enjoyable for students?</i>	
11 / 21	Student Activities	
	10 / 11	Visualisations / Interactive Tools
	1 / 11	Creative Project
11 / 21	General Teaching Strategies	
	8 / 11	Active learning: peer teaching, solving problem in groups, etc.
	2 / 11	Empower the students / Enthusiatic Instructor
	1 / 11	Base learning on performance, not on deep insight.
10 / 21	Applications / Connections	
	7 / 10	Applications to the real world and the rest of their courses.
	3 / 10	Connection with something fun.
2 / 21	Miscellaneous	
	1 / 2	Classic approach, done well.
	1 / 2	I do not care.
19.	<i>Which teaching activities do you think could make TCS easier to understand for students?</i>	
12 / 21	Examples	
	7 / 12	Visualisations / Interactive Tools.
	4 / 12	Plain Examples.
	2 / 12	Connection with something fun.
	1 / 12	Applications in degree and real world.
7 / 21	General Teaching Strategies	
	6 / 7	Active learning: peer teaching, solving problems in groups, etc.
	1 / 7	Base learning on performance, not on deep insight.
1 / 21	Additional Knowledge	
	1 / 1	Teach students more background mathematics.
4 / 21	Miscellaneous	
	2 / 4	I do not know.
	1 / 4	Forbid Power-Point.
	1 / 4	Students should learn that some things are hard.
20.	<i>Which teaching activities do you think could improve the perceived relevance of TCS for students?</i>	
10 / 21	Academic / Abstract	
	5 / 10	Give examples/reasons of/for relevance.
	2 / 10	Show students real research; give dates, names, etc.
	2 / 10	Create connection with other modules.
	1 / 10	Get students involved in real research.

6 / 21	Industry / Career	<table border="1"> <tr> <td>5 / 6</td> <td>Show students a (timely) real-world application.</td> </tr> <tr> <td>2 / 6</td> <td>Active learning: debates / case studies on real-world topics.</td> </tr> </table>	5 / 6	Show students a (timely) real-world application.	2 / 6	Active learning: debates / case studies on real-world topics.														
5 / 6	Show students a (timely) real-world application.																			
2 / 6	Active learning: debates / case studies on real-world topics.																			
2 / 21	Non-Specific	<table border="1"> <tr> <td>1 / 2</td> <td>Enthusiastic instructors.</td> </tr> <tr> <td>1 / 2</td> <td>Visits of role models.</td> </tr> </table>	1 / 2	Enthusiastic instructors.	1 / 2	Visits of role models.														
1 / 2	Enthusiastic instructors.																			
1 / 2	Visits of role models.																			
4 / 21	Miscellaneous	<table border="1"> <tr> <td>2 / 4</td> <td>Market TCS as 'obtaining problem-solving skills'.</td> </tr> <tr> <td>2 / 4</td> <td>I cannot see or do not care that they would be considered irrelevant.</td> </tr> </table>	2 / 4	Market TCS as 'obtaining problem-solving skills'.	2 / 4	I cannot see or do not care that they would be considered irrelevant.														
2 / 4	Market TCS as 'obtaining problem-solving skills'.																			
2 / 4	I cannot see or do not care that they would be considered irrelevant.																			
21.	<i>Which of the activities you suggested in questions 18-20 have you tried in your teaching and what was your experience with them?</i>																			
16 / 21	Activities with positive experiences:	<table border="1"> <tr> <td>7 / 16</td> <td>Active Learning</td> </tr> <tr> <td>6 / 16</td> <td>Visualisations / Interactive Tools.</td> </tr> <tr> <td>3 / 16</td> <td>Real world example.</td> </tr> <tr> <td>2 / 16</td> <td>Connection with something fun.</td> </tr> <tr> <td>2 / 16</td> <td>Enthusiastic Instructor</td> </tr> <tr> <td>1 / 16</td> <td>Visit from role model.</td> </tr> <tr> <td>1 / 16</td> <td>Real research.</td> </tr> <tr> <td>1 / 16</td> <td>Teach TCS as reasoning skills.</td> </tr> <tr> <td>1 / 16</td> <td>Base learning on performance, not on deep insight.</td> </tr> </table>	7 / 16	Active Learning	6 / 16	Visualisations / Interactive Tools.	3 / 16	Real world example.	2 / 16	Connection with something fun.	2 / 16	Enthusiastic Instructor	1 / 16	Visit from role model.	1 / 16	Real research.	1 / 16	Teach TCS as reasoning skills.	1 / 16	Base learning on performance, not on deep insight.
7 / 16	Active Learning																			
6 / 16	Visualisations / Interactive Tools.																			
3 / 16	Real world example.																			
2 / 16	Connection with something fun.																			
2 / 16	Enthusiastic Instructor																			
1 / 16	Visit from role model.																			
1 / 16	Real research.																			
1 / 16	Teach TCS as reasoning skills.																			
1 / 16	Base learning on performance, not on deep insight.																			
22.	<i>One approach to teaching TCS topics is to assign students a game-building task through which they learn a particular TCS topic (see Learning by Game-Building for more detail on this approach). Have you tried this approach? If so, have you found it successful; what problems does it raise? If not, do you feel it could be successful, or not? What problems do you think it might raise?</i>																			
21 / 21	Not Tried.	<table border="1"> <tr> <td>9 / 21</td> <td>liked the approach.</td> </tr> <tr> <td>11 / 21</td> <td>were neutral or did not specify their opinion.</td> </tr> <tr> <td>1 / 21</td> <td>disliked the approach.</td> </tr> </table>	9 / 21	liked the approach.	11 / 21	were neutral or did not specify their opinion.	1 / 21	disliked the approach.												
9 / 21	liked the approach.																			
11 / 21	were neutral or did not specify their opinion.																			
1 / 21	disliked the approach.																			
0 / 21	Not Tried.																			
5 / 21	Possible problems:	<table border="1"> <tr> <td>4 / 5</td> <td>Approach may not be suitable for all of TCS.</td> </tr> <tr> <td>2 / 5</td> <td>Learning the game-engine takes time away from learning TCS.</td> </tr> <tr> <td>1 / 5</td> <td>Fun for student who like TCS, but does it work for the average student?</td> </tr> </table>	4 / 5	Approach may not be suitable for all of TCS.	2 / 5	Learning the game-engine takes time away from learning TCS.	1 / 5	Fun for student who like TCS, but does it work for the average student?												
4 / 5	Approach may not be suitable for all of TCS.																			
2 / 5	Learning the game-engine takes time away from learning TCS.																			
1 / 5	Fun for student who like TCS, but does it work for the average student?																			

23.	<i>Do you have any additional remarks/comments that you would like to add?</i>
No information relevant to study.	

2.4 Discussion

Because the results were collected using an open-ended questionnaire, they are extremely diverse. However, they do generally seem to conform to the hypotheses that were formulated based on claims made in the literature.

However, the first result is concerned with the design of the study, rather than the verification of the hypotheses. To avoid having to tie the results of the study to a particular institute or country, lecturers were recruited, rather than students. The results of Question 2 showed that we have managed to attract an international audience. Although not as diverse as we might have hoped for, our participants do have a good representation on both sides of the Atlantic.

A small detour was taken when we asked participants about their affinity with TCS teaching. Unsurprisingly, all participants said that they enjoyed teaching TCS. More specifically, the majority said to enjoy it because it is an area close their research interests and some said to enjoy it despite students reactions.

2.4.1 The Importance of TCS

The first major result was that of the importance of TCS in the undergraduate curriculum. The vast majority (80%) of participants considered TCS to be important. This result confirms the hypothesis that was formulated on the issue, although it should be taken with a grain of salt, as they represent the opinion of people who make their living teaching TCS and most of them conduct research in the same area as well. More interesting are the reasons that were provided for the importance of TCS. Three general reasons were provided by participants. First of all, 50% of participants who consider TCS to be important say that it is important because it teaches the students general reasoning skills. It is viewed, so to speak, as the Latin of informatics. Where Latin used to be taught to increase general learning skills, TCS is now being taught to increase general reasoning skills. However, 35% of participants who consider TCS to be important indicate that TCS is important because of its historical significance and its place in a comprehensive informatics curriculum. Finally, another 35% of participants say that TCS education is important because the knowledge is required in other courses and a student's future career.

Within TCS, our participants consider some topics to be more important than others to undergraduates. The Top 3 topics most frequently mentioned as important to undergraduates are:

1. Finite State Automata / Regular Expressions *and*
Turing Machines / Computability / Halting Problem / Rice's Theorem
2. Grammars / Push-Down Automata *and*
Algorithms
3. Logic *and*
Complexity Theory / Cook-Levin Theorem / NP-Completeness

Five out of these six topics are centred around a *modelling framework*, i.e. all topics except for 'Algorithms' aim to teach the students a kind of modelling skill, followed by a set of proofs or reasoning skills about the modelling framework in question. In the remainder of this chapter and indeed the rest of this thesis, topics such as the five mentioned above will be referred to as *modelling skills*. In the hypotheses it was predicted that concurrency and the accompanying (value-passing) CCS would be among the most important topics to teach at the undergraduate level, but the results do not support this claim. The reason probably lies in the fact that it is the most advanced and specialized topic mentioned on the hypothesized list. When it *is* taught at undergraduate level, it is most likely taught in the 3rd or 4th year, to students who are taking it as an elective.

2.4.2 Lecturers' Thoughts on Student Opinion

As far as student enjoyment of TCS is concerned, 61% of participants report that students tend to enjoy TCS less than their other informatics topics and only 9% report that they enjoy it more. The two most common reason participants gave for students not enjoying the TCS material are its lack of relevance and difficulty.

Similarly, 61% of participants report that students find TCS more difficult than their other informatics topics, while none of the participants reported that students find it easier. The main reason provided for the difficulty is lack of background. Interestingly, 18% of participants claim that student find TCS difficult because it *is* difficult.

The number are slightly more positive for the perceived relevance of TCS with 52% of participants reporting that students generally do not find the TCS material

relevant to the rest of their degree or their future career and 24% reporting that they do. However, 19% of participants report that students' perceived relevance is in fact *irrelevant*. Although there is at the very least a partial truth concealed in this statement, it ignores the impact of perceived relevance on motivation and the effect of motivation on learning.

The most common, and very insightful, reason participants provided for the lack of perceived relevance was that undergraduate students lack the background that is needed to appreciate the relevance of TCS. Interestingly, 14% of participants said that TCS was perceived as irrelevant because it *is* actually irrelevant to the informatics undergraduate degree and the students' future career.

When asked for their own experience with TCS as a student, 57% of participants said that they enjoyed it as a student. However, two-thirds of these participants also said they were the exception in their class, while not a single participant claimed to be the rule. Furthermore, 24% of participants actually hated TCS or found it particularly difficult. Of these participants, 80% said they were the rule in their class, but not one claimed to be the exception.

These results are salient for two reasons: (1) it shows once more the unpopular status of TCS and (2) it provides more than the two examples of change that we were hoping to find. There were indeed 3 participants who hated TCS when they were a student, but who ended up teaching it for a living and indeed enjoy doing so. These three participants demonstrate that opinions are not set in stone and, given the right approach, student *can* end up liking a topic they initially disliked.

Similarly, there were 2 other participants who indicated that they really struggled with the TCS material as a student, but who obviously ended up being sufficiently adept at it to teach the material at university level. This result enables us to refute anyone trying to argue a TCS equivalent of Bornat's argument that teaching programming is a futile exercise (Dehnadi and Bornat, 2004).

The participants were divided on the issue of how student opinion has changed over the years. While 24% of participants thought that student opinion of TCS had gone down since they were a student, 10% thought it had gone up, while the majority either had no idea or thought there had been no change.

2.4.3 Suggested Improvements

The improvement suggested by participants were also very diverse. Starting with the suggested improvements for student enjoyment the most common ones were: 48% visualisations and interactive tools, 38% active learning, e.g. peer teaching, group problem solving, and 33% application to the real world and the rest of their degree. A connection with something fun and enthusiastic instructors were also mentioned several times.

For difficulty, the first two of the Top 3 enjoyment suggestions were mentioned again, this time with a frequency of 33% and 29% respectively. Explicit examples and a connection with something fun were also mentioned several times.

Finally, for perceived relevance there were no suggestions which were given by a large number of participants, although several suggestions were mentioned more than once. Starting with the most common one, these include: (a) give explicit examples, (b) show students timely real-world applications, (c) active learning, e.g. debates, case-studies, (d) show real research, and (e) create connections with other modules.

The technique which was successful for the largest number of participants is active learning (33%), followed closely by visualisations and interactive tools (29%). Other methods which our participants found beneficiary include using real-world examples, a connection with something fun, and an enthusiastic instructor.

When asked if the participants had any experience with 'Learning by Game-Building', the approach proposed in this thesis, they all said they were not familiar with the approach. More than 40% of our participants said they liked the approach and less than 5% said they did not like it. The participants identified three possible problems with the approach: (1) it may not suitable for all of TCS, (2) learning the game-engine takes time away from learning the actual TCS material, and (3) while the approach will probably be fun for students who like TCS, it may not be appealing to the average student.

As far as the first potential problem is concerned, it is not claimed that the approach is suitable for all of TCS. In fact, it claims to be suitable for the modelling skills only. The second comment is a valid observation. Learning to work with the game-engine does create some overhead, although every effort has been made to minimize this by using a programming language with which the students are already familiar and a small, but well-documented module in which every game-building operation is directly mapped onto a sensible action in the modelling framework. However, the ap-

proach would not be nearly as viable in a non-informatics course, because the student would lack a basic programming skill and acquiring this would likely create too big an overhead. The third ‘potential problem’ is exactly the question this thesis aims to find an answer to.

2.5 Summary

We found that lecturers do believe students to be less motivated on TCS than they are on other topics in Informatics. Several causes of this were perceived to be: (1) lack of fun, (2) difficulty, and (3) lack of relevance. We also found that five out of the six topics in the top three of most important undergraduate topics can be classified as *modelling skills*. In the next chapter, we will look at solutions to these issues which have previously been suggested in the literature. These suggestions will then form the basis for the proposed approach described in Chapter 4.

Chapter 3

Background

3.1 Introduction

In the previous chapter the current state of theoretical computer science (TCS) education was investigated, with mixed results. Among others, it was observed that students are perceived to enjoy TCS less than their other computing subjects, that they find the subject more difficult, and that they find it less relevant to their degree and future career.

In this chapter we will present an overview of the most influential literature to date in the areas that are relevant to our research. The following topics will be touched upon: (1) learning objectives in theoretical computer science, (2) constructionism, (3) motivation, (4) game-based learning, and (5) a word on transfer.

3.2 Learning Objectives in Theoretical Computer Science

Theoretical Computer Science (TCS) is a very broad area and it would be impossible to capture the entire field using a single approach. However, the focus of this thesis is undergraduate education and the list of TCS topics commonly taught at that level is considerably shorter. In fact, Chapter 2 provided a set of *modelling skills* which were shown to be at the centre of undergraduate TCS education. These modelling skills cover most of the topics in a standard *Theory of Computation* course and a few other ones. Examples include: Finite State Automata, Push-Down Automata, Context-Free Grammars, Context-Sensitive Grammars, Turing Machines, and the value-passing

Calculus of Communicating Systems (CCS).

Besides providing a list of TCS topics commonly taught at undergraduate level, Chapter 2 also showed that students enjoy TCS less than their other informatics modules and that they find the material more difficult and less relevant than that of their other modules.

Stenning (2002) demonstrated an additional problem of teaching TCS. In his study, he found that some students acquire better reasoning skills when they are taught using a visualisation tool, while other acquire better reasoning skills when a traditional syntactic method is used.

Although the study was performed in a TCS context, its consequences are not limited to this area. In general, just because a method works for a particular kind of student, it does not necessarily work for everybody. Similarly, just because a method does not work for everybody, it does not mean it does not work for anybody. The interesting task is to find criteria based on which one can distinguish between those who do well on a certain approach, and those who do not.

3.2.1 Bloom's Taxonomy

There is a question of how well students will be able to learn a topic in TCS through building a game. To answer that question we turn to Bloom's Taxonomy of Learning Objectives (Bloom, 1956). Bloom indicates three domains of learning outcomes: (1) cognitive, i.e. mental skills, (2) affective, i.e. feelings, values, attitudes, for example, listening to others with respect, and (3) psycho-motor, i.e. manual or physical skills. In the area of computer science we are concerned with the cognitive domain only. The cognitive domain is itself split up into six different competence categories:

- (a) *knowledge*: recalling information
- (b) *comprehension*: understanding the meaning of the information
- (c) *application*: applying information to new situations
- (d) *analysis*: breaking down information/structure into its component parts
- (e) *synthesis*: putting parts together to form a new structure
- (f) *evaluation*: judging the value of a structure

Category (a) is the lowest and category (f) the highest. In undergraduate TCS courses students are generally expected to demonstrate competence at the analysis, synthesis, or evaluation level, depending on the topic and the degree for which the student is studying. Game-building is a synthesis activity and therefore has the potential to promote competence at this level. Whether or not it has the potential to promote competence at the evaluation level is more difficult to determine. However Anderson *et al.* (2001) propose a revision of Bloom's taxonomy in which synthesis rather than evaluation is the highest level of competence. In other words, accomplishing competence at the synthesis level implies competence at the evaluation level. Going by the revised taxonomy, a student *can* achieve the highest level of cognitive competence in TCS topics using a game-building approach.

3.2.2 A Younger Audience

Recently, some effort has been made to make theoretical computer science (TCS) accessible to a younger audience. In particular, Bell *et al.* (1998) wrote an anthology of TCS workshops for 5–12 year olds. In order to engage their audience in a topic which, according to Study 1 described in Chapter 2, even computer science students find less enjoyable than their other subjects, the authors use several different techniques. First of all, many of their workshops incorporate a game-element: either the children are asked to participate in some sort of game, or they have to design a game themselves. The second technique the authors make heavy use of is relevance. For every topic, they find an application which is meaningful to the children. Bell *et al.* (1998) is presented as an anthology only. There is no empirical evidence for the fact that the approach supports learning.

Nevertheless, in our study we are hoping do something similar by (1) having the student program a game and (2) letting them choose their own topic for the game. The former attempts to integrate theory and applications, thus addressing the relevance issue, while the latter attempts to make the experience more engaging to the students. Also, perceived relevance could further increase when students choose a topic for their game in which they have a personal interest. However, this line of argument should be considered speculation, because no empirical evidence was provided in support of the approach described in Bell *et al.* (1998).

A second attempt at making TCS accessible for children, or those without a strong mathematical background, is described in Valente (2004). The authors present their

system of *c-cards* as a universal metaphor for a class of TCS concepts including finite state automata (FSAs) and regular expressions (REs). The *c-cards* are square, tile-like cards with two ports on each of their four edges and arrows connecting these ports internally. There are eight different kinds of cards, each with their own unique permutation of arrows representing a particular computational behaviour. The cards can be connected to form more complicated behaviours. The *c-card* system is formalized in Valente (2003).

Although the *c-card* metaphor does seem to stand up, there is no empirical evidence for the fact that *c-cards* support learning. Furthermore, there is a serious question of relevance in this approach. The *c-cards* are a self-contained system, with no obvious connection to any real-world example or application which is meaningful to children. However, the *c-cards* do bear a striking resemblance with popular board-games such as Carcassonne and Game of the Goose. In fact, it seems to be a direct hybrid of the two. Redefining the *c-cards* system in terms of these two board-games could provide motivational and educational benefits.

The study described in this thesis takes a similar approach to that of Valente (2004). However, rather than using a single universal metaphor, a new one is defined for each modelling framework. Although a universal metaphor would have been more desirable from both a practical and an educational viewpoint, it would have limited the domain of our approach. For example, it would have been difficult to capture both turing machines and regular expressions using the same metaphor.

Furthermore, the metaphor used in our study takes the form of a game-building environment, thus avoiding the relevance issue affecting the approach described in Valente (2004).

3.3 Constructionism

The approach to TCS education taken in our research can be classified as *constructionist*. Constructionism is due to Papert (1980) and is based on Piaget's constructivism (Piaget, 1926, 1965, 1969). Harel and Papert (1991) describe the difference between the two:

“Constructionism – the N word as opposed to the V word – shares constructivism’s connotation of learning as ‘building knowledge structures’ irrespective of the circumstances of the learning. It then adds the idea that this happens especially felicitously in a context where the learner is con-

sciously engaged in constructing a public entity, whether it's a sand castle on the beach or a theory of the universe" – from Harel and Papert (1991).

3.3.1 Logo and its Relatives

Papert developed constructionism in the context of his work on Logo (Papert, 1980) which he started in the 1960's at the Massachusetts Institute of Technology (MIT). The best-known application of the Logo language is *'turtle-graphics'*. Originally, the application consisted of an actual mechanical robot, which was connected to the computer with an *'umbilical cord'*. The children could give the robot commands like *forward 100, back 50, left 90, and right 45*.

Logo, and the accompanying robot turtle, was proposed by Papert as a modelling tool which could serve as an example for future modelling tools. He called modelling tools like Logo *objects-to-think-with* and claimed that their ability to represent knowledge and the possibility for personal identification are the key aspects of these objects that facilitate learning. The introduction of Logo also illustrates one of the key differences between Papert's theory of constructionism, and Piaget's constructivism. That is, Piaget explains slower development of a particular concept in terms of its complexity, while Papert blames the same phenomenon on a lack of suitable objects in the environment which promote the development of the concept. In Papert (1980), he claims that Logo is an object which promotes mathematical skills.

Although the ideas presented in Papert (1980) were influential, they were not supported by any empirical evidence. Harel (1991) did provide such evidence for the topic of fractions, but the numbers used in his study were small with only 17 students in the experimental group. Furthermore, the experimental group received the experimental treatment on top of their regular classes, so any increased performance may have been due to the fact that in total, the experimental group spent more time on fraction-related tasks.

Nowadays, the Logo application is still in use, albeit in a different form: instead of controlling a mechanical robot, the children control a turtle on their computer screen. This version of Logo is much faster and more accurate than the original, which means that more complicated computations can be made and that the modelling tool is more widely accessible. This allowed it to be used in studies on subjects other than basic maths in which Logo was expected to facilitate learning, e.g. geometry and problem-solving.

In the area of problem solving, the literature generally failed to find any effect of a

student's exposure to Logo on their problem solving skills (Bruggeman, 1986; LeWinter, 1986; Milojkovic, 1984). However, given specific teacher mediation, students performed better on problem-solving tasks than control groups without programming experience or programming experience in a different language (Clements and Merriman, 1988). These results raise questions about the reason behind Logo's success – in particular over other programming languages – and about the role of the mediation: is Logo the contributor toward better problem-solving skills, or is it the teaching-mediation? Similarly, in the area of geometry it was shown that Logo helps students think about angles, when given the appropriate teacher mediation (Kieran, 1986).

The most recent incarnation of the Logo turtle is StarLogo (Resnick, 1996), a special purpose programming language for modelling decentralised systems, and StarLogo TNG which adds graphical programming and a 3D-world to the original. StarLogo allows children to model phenomena such as bird flocks, traffic jams, ant colonies, and market economies. There is no empirical evidence which shows that StarLogo facilitates learning.

The issue that seems to be underrepresented in the extensive literature on Logo is that of individual differences. Questions like 'which kinds of students benefit from the use of Logo' or 'which aspect of Logo makes it (un)successful' need further investigation.

Besides StarLogo, other modelling tools for children have been developed that take full advantage of recent advances in (3D-)graphics. For example, Ken Kahn's *ToonTalk* (Kahn, 2000, 2004) allows children to create models by interacting with a virtual world in which every abstract programming concept has a concrete representation. For example, methods are depicted as robots, processes are the houses in which the robots work, and communication between processes (houses) is taken care of by homing-pigeons. According to Kahn, there are three major hurdles when learning to program: (1) syntax, (2) abstraction, and (3) conceptual (i.e., breaking up the task in smaller sub-tasks). He claims that *ToonTalk* takes away the first two: the first one because the programming, or modelling, is done by interacting with the virtual world, rather than writing code; the second one because abstractions are derived from concrete objects. The third hurdle remains work in progress. There is no empirical evidence which shows that *ToonTalk* facilitates learning.

Another example of a special purpose programming language for children and first-year undergraduates is *Alice*: a language for programming 3D-animations (Conway, 1997; Conway *et al.*, 2000). Moskal *et al.* (2004) provides evidence for the claim that

the use of *Alice* can improve the performance and retention of those undergraduate students in computer science who are considered to be ‘at risk’. A student is considered to be ‘at risk’ when they have either a weak mathematical background, a weak computer science background, or both. Although this study did demonstrate that the experimental group showed better performance and retention, all three groups – one experimental and two control groups – were small, i.e. between 6 and 18 students. Furthermore, the experimental group is self-selecting, as students chose whether or not to take the *Alice* course on top of their normal course load. Perhaps the students who elected into the course are the most highly motivated students who would have showed better performance and retention regardless of their work with *Alice*.

Another tool developed specifically for undergraduate students is the Robocode project developed by Mathew Nelson (Nelson, www). Robocode is a robotics battle simulator which was originally designed as a tool to teach novice Java programming. It is still being used throughout the curriculum (Hartness, 2004; Bonakdarian and White, 2004), but has also become popular outside school contexts. Large repositories can be found online into which people can upload the robots they write. Other people can then download those robots and simulate a battle between their own robot(s) and the ones they download. In Robocode, Nelson has found an educational tool that people choose to use outside the formal educational context it was created for. There is no empirical evidence which shows that RoboCode facilitates learning, but its popularity outside a formal education context does demonstrate a strong motivational appeal.

Java is not the only language for which approaches to improve novice programming education have been suggested. For example, Merritt (1990) presents the content of an introductory course in the logic programming language PROLOG as the information needed to program your own adventure game in the language. This brings us back to Papert’s Logo, which was designed as a modelling tool for basic mathematics. PROLOG can be seen as a similar modelling tool, but for use on the topic of predicate logic, one of the most commonly taught topics in the theoretical computer science curriculum.

3.3.2 Programming in Other Subject Areas

All of the examples above, with the exception of a few of the Logo ones, are concerned with the teaching of computer related skills. However, programming can also be used to teach other (school) subjects. For example, DiSessa (2000) describes a pro-

programming language called *Boxer*, which is used to teach children in sixth grade (US equivalent of the Scottish P6–P7) and higher, the algebra behind Galileo’s principles of motion. DiSessa explains his idea behind using programming as a means to learn algebra as follows:

“Programming turns analysis into experience and allows a connection between analytic forms and their experimental implications that algebra and even calculus can’t touch” – from DiSessa (2000).

Boxer is essentially a programming language in the Logo-tradition and the approach described in DiSessa (2000) can be classified as constructionist. Both Logo and Boxer promote the ability for every user to be a *creator* as well as a *consumer*. The main difference between the Boxer school-of-thought and the Logo school-of-thought lies in the organisation of the medium. Where more recent incarnations of Logo try to incorporate popular functionalities, Boxer focusses on structural changes to the medium which could allow more kinds of user to become *creators* as well as *consumers*.

Robertson and Good (2005) describe a project in which children develop their narrative skills by programming video games in an informal setting. This project is of particular interest to us because like ours, it is centred around the concept of game-creation. Good and Robertson (2006) present a theory about the learning and motivational affordances of game-building. The authors describe what they call an *organic reflection-revision cycle*, where ‘organic’ refers to the fact that learners naturally engage in this cycle, without being instructed to do so. They contrast this cycle with an example of an English teacher who finds it difficult to motivate her students to revise their essays after they have produced the first draft.

Good and Robertson (2006) claim that there are two attributes of game-building environments which support the reflection-revision cycle: (1) individual action-based feedback and (2) collaborative action-based feedback. The former is acquired when the author of the game plays (part of) the game to test it for possible problems. The latter is acquired when the author of the game observes a member of the target audience play the game. The observation can provide the author with valuable information such as the difficulties that the player encounters while playing the game. In both cases the feedback is presented in terms of the *interaction* with the creative product. This can be contrasted with more traditional static forms of feedback like written comments on an essay, or a word processor underlining misspelled words (examples taken from Good and Robertson (2006)).

The third learning affordance identified by Good and Robertson (2006) is *organic collaboration*. Again, ‘organic’ refers to the fact that learners naturally engage in the collaboration, without being told to do so. The authors describe how a piece of knowledge which a workshop facilitator bestows on one of the pupils, spreads through the classroom without any further intervention from the facilitator.

The theory presented in Good and Robertson (2006) is based on extensive field work and currently accepted theories of learning. The authors hypothesize that game-building environments other than the one used in their fieldwork are likely to provide the same kinds of learning affordances. Because the approach described in this thesis is defined in terms of game-building, we can expect the learning affordances described in Good and Robertson (2006) to apply. However, because our approach focusses on the effect personalisation and choice *within* a game-building environment, this expectation will not be tested. Good and Robertson (2006) also discuss personalisation and choice in their description of the motivational affordances of game-building environments. These are discussed in Section 3.4.3.

In Habgood *et al.* (2005) we find another study on game-programming for children. In this study, the children were asked to write an educational game on a topic of their choice. Previous studies (Kafai, 1995; Reiber *et al.*, 1998) had shown that children rarely attempt to integrate the learning content with the fantasy context when writing educational games. Games like these in which there is no integration of learning content with the fantasy context are referred to as **extrinsic games**. Habgood *et al.* (2005) were hoping to find an increase in the number of **intrinsic games** – games in which the fantasy context *is* integrated with the learning content – compared to previous studies because (a) they had exposed the children to concrete examples of both intrinsic and extrinsic educational games beforehand – previously only extrinsic games had been available – and (b) the learning content was not a set topic. However, results were negative: no increase in intrinsic games was found.

In our approach, the mapping of the model – created by the student – onto the game is defined in the game-engine. This means that the student has no control over extrinsic/intrinsic nature of their game; it is pre-defined by the developer of the game-engine. If an intrinsic mapping is defined, then the games created by the students will all be intrinsic, if an extrinsic mapping is defined, the games will all be extrinsic. Malone and Lepper (1987), as discussed in Section 3.5, suggest that in the context of game-playing, intrinsic games are the more desirable kind, because of their increased motivational appeal. The authors claim that intrinsic games also have the potential to

provide a valuable educational metaphor by linking existing knowledge to the material that is being learned.

There is no immediate reason to believe that game-engines with an intrinsic mapping will exert an increased motivational appeal, like Malone and Lepper (1987) claims that intrinsic games do. However, their potential to provide a valuable educational metaphor is not defined in terms of game-playing. It should therefore apply to game-engines with an intrinsic mapping as well as to intrinsic games. Thus, an intrinsic mapping will be preferable over an extrinsic one in our approach.

3.3.3 The Overhead Problem

Although programming – similar to other modelling tools – can be a valuable teaching aid for various subjects, the approach can also have its drawbacks. In particular, the *overhead problem* has significantly reduced the effectiveness of many attempts to help people learn by programming as described in Littlefield *et al.* (1988). Learning the syntax and semantics of a new programming language can get in the way of learning the actual material that is in the curriculum. The learning-to-program overhead does not only apply to students, but to teachers as well. In addition to learning the new programming language, teachers are also expected to come up with a lesson plan which fits into the current curriculum and monitor the progress of the individual students. According to Littlefield *et al.* (1988) this often turns out to be too heavy a load for many teachers. However, these objections to learning-by-programming do not apply to our current domain because (1) our participants are informatics students who already possess the basic programming skill needed to program a game, and (2) the same argument holds for the teachers/tutors/demonstrators involved in our study.

If the approach was to be extended to cover topics outside informatics though, or if it was to be used for a younger audience, the current set-up would likely cause us to run into the overhead problem and alternative modelling tools might need to be considered.

Several modelling tools which require only minimal programming, or no programming at all have been proposed for game-building tasks in the literature. For example, (Maloney *et al.*, 2004) describes a language called Scratch. Scratch is a tile-based visual programming language which allows the user to create simple computer games. Other examples include, Stagecast Creator (Louca and Zacharia, 2007) and the Neverwinter Nights Aurora toolkit (Good and Robertson, 2006). Although none of these toolkits were designed to teach any particular modelling framework through game-

building, they do show that games can be created with minimal or no programming. Therefore, it opens up a possibility for our approach to move outside the informatics domain in the future.

3.4 Motivation

The central question in our research is: what effect does learning by game-building have on student enjoyment and performance? Chapter 1 mentioned that the approach is not expected to have the same effect on all participants. On the contrary, it is expected that participants will respond differently to the approach, based on their motivational type and gender. Therefore, this section is dedicated to motivation, motivational types, and how to measure these.

3.4.1 Intrinsic vs. Extrinsic Motivation

Skinner (1953) first described the effects of extrinsic rewards on behaviour. The author found that if a reward was given immediately following a specific behaviour, the behaviour was more likely to occur again, compared to a situation in which no reward had been given. The motivation to participate in an activity for the sake of an extrinsic reward is referred to as *extrinsic motivation*. Similarly, the motivation to participate in an activity to avoid punishment is also referred to as *extrinsic motivation*.

White (1959) describes an entirely different kind of motivation. The author observed that some behaviours seem to occur even when no reward is given. He hypothesizes that the reason for participating in such behaviours is the positive experience of developing one's capabilities. This kind of motivation where an individual participates in an activity for its own sake is known as *intrinsic motivation*.

Comparative studies have shown that when a person is extrinsically motivated – as opposed to intrinsically motivated – they tend to show poorer concept attainment (McCullers and Martin as cited in Amabile *et al.*, 1994), impaired complex problem solving (Glucksberg as cited in Amabile *et al.*, 1994), poorer incidental learning (Barrick *et al.* as cited in Amabile *et al.*, 1994), and lower levels of creativity (Amabile as cited in Amabile *et al.*, 1994). Therefore, intrinsic motivation seems more desirable in an educational context. Ironically, the current formal education system focuses mainly on extrinsic motivation, i.e. a good performance is rewarded with a good grade. Furthermore, Deci (1971) showed that tangible rewards can have a detrimental effect on the

intrinsic motivation of college students for an interesting activity. Numerous studies have been published presenting evidence in support of this claim. For example, Lepper (1973) describes a study in which a group of nursery school children who liked to play with marking pens were promised a reward for engaging in this particular activity. A control group was given no such promise. When the promise of the reward was taken away, the children who had been promised a reward played with the marking pens less than the control group. Harlow et. al. (as cited in Pfaffman, 2003) report a similar result in study on a group of monkeys, who liked to take apart puzzles. The group was offered a reward in the form of a raisin for successfully taking apart the puzzles, while a control group received no reward. When the reward was taken away, the monkeys lost interest in the puzzles, while the control group continued to take the puzzles apart as before.

However, the exact conditions under which this undermining effect occurs are subject to debate. For example, Deci *et al.* (1999) found in their meta-analytic review that if a reward was presented in the form of positive feedback, intrinsic motivation increased rather than decreased. The increase was stronger in college students than it was in children. All other forms of rewards were found to decrease intrinsic motivation. Eisenberger *et al.* (1999) found in their meta-analytic review that a reward for exceeding the past performance of others increased rather than decreased intrinsic motivation. Furthermore, the authors found that only a reward for meeting vague performance standards reduces intrinsic motivation. In both studies intrinsic motivation was measured using self-reported interest and free-choice, where free-choice is measured by the task-persistence in the participants' free time.

3.4.2 Scales and Measures

Several proposals have been described for motivation scales that distinguish between more than just intrinsic and extrinsic motivation, frequently in conjunction with measures to validate the scale in question. For example, Amabile *et al.* (1994) divides intrinsic motivation into two different sub-categories: *enjoyment* and *challenge*. Extrinsic motivation is divided into two sub-categories in a similar way: *outward* and *compensation*. The measure that is used to determine the validity of the scale is called the 'Work Preference Inventory' or WPI. The authors present two versions: one for college students and one for working adults.

Ryan and Deci (2000) present a scale in which extrinsic motivation is subdivided into four different categories. These categories differ in the degree to which the external reward has been internalised. For example, a student who studies for an exam because he fears the reaction of his parents should he fail, is extrinsically motivated to do so. A student who studies for an exam because he believes it is valuable for his chosen career, is also extrinsically motivated to do so. The difference between these two students is that while the behaviour of the former is mere compliance with external control, in the behaviour of the latter there is an element of choice: the motivation of the latter is more internalised than the motivation of the former. Ryan and Deci (2000) also present a new category called *amotivation* which represents a state in which a person is lacking any intention to act.

Vallerand *et al.* (1992) describe a measure for a scale which also uses *amotivation* and the degree of internalisation to sub-categorise *extrinsic motivation*. In addition to this, they also sub-categorise *internal motivation* into (1) to know, (2) toward accomplishment, and (3) to experience stimulation. The scale is called the 'Academic Motivation Scale' or AMS.

3.4.2.1 Achievement Goals

Elliot and McGregor (2001) do not use the standard extrinsic/intrinsic division. Instead the authors describe a 2x2 achievement goal framework and a measure, in the form of a questionnaire, which was used to validate the framework. The axes of their framework represent the *valence* and *definition* of motivation and they each contain exactly two values. For valence, these values are (1) positive, i.e. **approaching** success and (2) negative, i.e. **avoiding** failure. For definition, these values are (1) absolute/intrapersonal, i.e. compared to some absolute or personal standard of **mastery** and (2) normative, i.e. compared to the **performance** of others. Combining the two axes results in four different achievement goals: (1) *mastery-avoidance*, (2) *mastery-approach*, (3) *performance-avoidance*, and (4) *performance-approach*. Ignoring the right-most column for the moment, Figure 3.2 shows all four achievement goals in the 2x2 achievement goal framework. Elliot and Thrash (2001) gives a formal description for each of them:

*A **mastery-approach** goal represents striving to attain task mastery of improvement; a **mastery-avoidance** goal represents striving not to fall short of task mastery or striving not to lose one's skills, abilities, or knowledge; a **performance-approach** goal represents striving to do better than oth-*

ers; a *performance-avoidance* goal represents striving not to do worse than others. – from Elliot and Thrash (2001)

Examples of each of the achievement goals follow. An example of *mastery-avoidance* (from Elliot and McGregor, 2001) would be to avoid misunderstanding or failing to learn course material; an example of *mastery-approach* would be to try to learn and fully understand the course material; an example of *performance-avoidance* would be to avoid being in the bottom half of the class; an example of *performance-approach* would be to try being in the top half of the class.

Elliot and McGregor (2001) also describes a questionnaire, aptly named the achievement goal questionnaire (AGQ), which they use to measure a person's achievement goal orientation. The questionnaire contains 12 items, 3 for each of the achievement goals as shown in Figure 3.1. Questions 1–3 are on mastery approach, Questions 4–6 on performance approach, Questions 7–9 on performance avoidance and Questions 10–12 on mastery avoidance. They are presented in a random order to participants. Each of the questions is answered on a 7-point Likert scale, resulting in 4 scores ranging from 3–21, once again one for each of the achievement goals. Finney *et al.* (2004) describe a general academic version of the questionnaire used by Elliot and McGregor (2001). The same set of questions is used, but the wording was adapted to examine general academic motivation, rather than class-specific motivation. As an example, consider the following pair of questions:

- My goal in this class is to get a better grade than most of the other students.
- My goal this semester is to get better grades than most of the other students.

The former is a question from the questionnaire defined in Elliot and McGregor (2001) and the latter is the rewording of this question presented in Finney *et al.* (2004). The questionnaire presented in Finney *et al.* (2004) will be referred to as the general academic achievement goal questionnaire (GAAGQ).

Empirical evidence in support of a fifth achievement goal is presented in Pieper (2003) This 5th achievement goal is referred to as *work-avoidance* and can be characterized by a drive to do as little work as possible. The reliability of the fifth achievement goal is demonstrated using an exploratory factor analysis. Figure 3.2 shows how work-avoidance fits in with the existing framework by creating a third dimension along the mastery/performance axis. The (GA)AGQ is extended with Questions 12–16, the last of which is reverse scored, as can be seen in Figure 3.1.

1. Completely mastering the material in my course is important to me this semester.
 2. I want to learn as much as possible this semester.
 3. The most important thing this semester is to understand the content in my courses as thoroughly as possible.
 4. My goal this semester is to get better grades than most of the other students.
 5. It is important for me to do well compared to other students this semester.
 6. I want to do better than other students this semester.
 7. I just want to avoid doing poorly compared to other students this semester.
 8. The fear of performing poorly is what motivates me.
 9. My goal this semester is to avoid performing poorly compared to other students.
 10. I'm afraid that I may not understand the content of my courses as thoroughly as I'd like.
 11. I worry that I may not learn all that I possibly could this semester.
 12. I am definitely concerned that I may not learn all that I can this semester.
 13. I really don't want to work hard in my classes this semester.
 14. I want to do as little work as possible this semester.
 15. I want to get through my courses by doing the least amount of work possible.
 16. I look forward to working really hard this semester in my coursework.
- (reverse scored)**

Figure 3.1: The General Academic Achievement Goal Questionnaire (incl. Work Avoidance)

	<i>Absolute / Intrapersonal</i> (mastery)	<i>Normative / Interpersonal</i> (performance)	Work Related
<i>Positive (approaching success)</i>	Mastery Approach Goal	Performance Approach Goal	n/a
<i>Negative (avoiding failure)</i>	Mastery Avoidance Goal	Performance Avoidance Goal	Work Avoidance Goal

Figure 3.2: The Five Dimensions of the Achievement Goal Framework

In Study 2 the GAAGQ and the GAAGQ including work avoidance will be used. In Study 3 the AGQ including work avoidance will be used. A choice was made for this particular framework and questionnaire, because the dimensions it identifies can easily be translated into hypotheses about the results of our studies, thanks to an abundance of research linking the different achievement goals with other attributes. For example, Archer (1994) found a link between work-avoidance goals and preference for easy tasks. This allows us hypothesize in Study 2 that participants with a high work-avoidance score will generally choose the task which they perceive to be the easiest. Figure 3.3 shows similar links between the achievement goals and other attributes. The attribute self-determination is adapted from Deci and Ryan (1991). It represents an inherently appetitive desire for autonomy and choice, and can be compared with the attribute *control*, often used in similar studies.

A second, less important reason for choosing the achievement goals framework was the availability of a widely-used questionnaire which has been tested for reliability and validity (Elliot and McGregor, 2001). An additional advantage of the questionnaire is that it can be completed by students in 5 minutes. This was a benefit in our design, because the study was conducted in the context of an assessed assignment.

Besides the (GA)AGQ, other measures have been specified for the achievement goal framework. For example, the patterns of adaptive learning (PALS) questionnaire (Midgley *et al.*, 2000) was designed with middle school children in mind. However, a choice was made for the (GA)AGQ rather than any other measure for achievement goals because it was designed for college students, which make up the groups of participants

Orientation	Associated with ...
<i>Mastery-Approach</i>	<ul style="list-style-type: none"> ● deep processing (Elliot and McGregor, 2001; McGregor and Elliot, 2002) ● long-term retention of exam-material (Elliot and McGregor, 1999) ● class/material engagement (Elliot and McGregor, 2001) ● positive attitude when faced with challenge (McGregor and Elliot, 2002) ● high competence expectancies (Elliot and McGregor, 2001; Pieper, 2003) ● overall need for achievement (Elliot and McGregor, 2001) ● self-determination (Deci and Ryan, 1991)
<i>Performance-Approach</i>	<ul style="list-style-type: none"> ● surface level processing (Elliot and McGregor, 2001) ● competitiveness (Pieper, 2003) ● high competence expectancies (Elliot and McGregor, 2001) ● high grades (Harackiewicz <i>et al.</i>, 1997) ● overall need for achievement (Elliot and McGregor, 2001)
<i>Mastery-Avoidance</i>	<ul style="list-style-type: none"> ● class/material engagement (Elliot and McGregor, 2001) ● fear of failure (Pieper, 2003) ● worry (Elliot and McGregor, 2001) ● disorganisation (Elliot and McGregor, 2001) ● lack of self-determination (Deci and Ryan, 1991)
<i>Performance-Avoidance</i>	<ul style="list-style-type: none"> ● low competence expectancies (Elliot and Church, 1997) ● competitiveness (Harackiewicz <i>et al.</i>, 1997) ● fear of failure (Elliot and McGregor, 2001) ● disorganisation (Elliot and McGregor, 2001) ● lack of self-determination (Deci and Ryan, 1991)
<i>Work-Avoidance</i>	<ul style="list-style-type: none"> ● work minimization strategies (Elliot, 1999) ● competitiveness (Harackiewicz <i>et al.</i>, 1997) ● limited class/material engagement (Dowson and McInerney, 2001) ● fear of failure (Pieper, 2003) ● preference for easy tasks (Archer, 1994)

Figure 3.3: Attributes Associated with the Five Achievement Goals.

used in Studies 2 and 3.

3.4.2.1.1 Derived Measures The (GA)AGQs results in either four or five scores, depending on whether the work-avoidance questions are included or not. However, these scores are commonly transformed into derived measures which either focus on the achievement goals of interest, or redistribute the information contained in these four or five scores. For example, Harris *et al.* (2007) collapses the approach/avoidance axes to obtain two scores:

$$\begin{aligned} \text{mastery} &= \text{mastery avoidance} + \text{mastery approach} \\ \text{performance} &= \text{performance avoidance} + \text{performance approach} \end{aligned}$$

This particular measure is chosen because the study focussed on the difference between mastery-oriented participants and performance-oriented participants. Similar transformations will be applied in Chapter 6 of this thesis.

3.4.3 Choice

As mentioned in Section 3.4.1, we are hoping to find an increase in *intrinsic* motivation, and possibly learning outcome, in a *learning by game-building* scenario where students build their own game, compared to a *learning by game-building* scenario in which students copy a pre-defined game. This expectation was based on (a) the fact that the two conditions differ only in the amount of personalization and choice offered, and (b) the previous literature on the positive effect of personalization and choice on intrinsic motivation. For example, Zuckerman *et al.* (as cited in Pfaffman, 2003) describes a study involving children and puzzles. One group of children was asked to solve 3 different puzzles. Another group was given 6 different puzzles and asked to solve 3 of them. The latter group persisted longer in working with the puzzles and showed more interest in continuing to work with the puzzles.

DeCharms (as cited in Pfaffman, 2003) describes a study in which he asked college students to play with Tinkertoys either in a ‘pawn’ condition, where they had to follow instructions step-by-step, or in an ‘origin’ condition, where they were allowed to create their own model. DeCharms (1971) found increased affect for the students in the ‘origin’ condition. The same group was also more likely to continue working on the model, completed the task more elegantly, and recalled more of the names associated with the model. Our approach is similar to the one described in DeCharms (1971) in the sense that Study 2 and 3 also have a ‘pawn’ condition, and an ‘origin’ condition.

In the ‘pawn’ condition, which is called ‘pre-defined’ in this thesis, participants create a copy of a pre-defined game. In the ‘origin’ condition, which is called ‘own-game’ in this thesis, participants create their own game.

More recently Cordova and Lepper (1996) looked into the effect of personalisation and choice on learning outcome in the context of educational computer games. The choices that were offered covered only instructionally irrelevant aspects of the educational computer game. The children that participated in the study were randomly assigned to one of five groups: (1) generic fantasy, no choice, (2) generic fantasy, choice, (3) personalised fantasy, no choice, (4) personalised fantasy, choice, and (5) no fantasy control group. The authors concluded that both personalisation and choice have a positive effect on enjoyment. They also found that the two factors have a positive effect on learning outcome:

“... they became more deeply involved in the activities, attempting to use more complex operations, and thereby learning more from the activities in a fixed period of time.” – from Cordova and Lepper (1996).

The study presented in Cordova and Lepper (1996) is discussed in more detail in Section 3.5. We believe the results of this study on the effects of personalisation and choice are relevant to our research because writing a game is essentially a set of choices. The choices are present from beginning to end: first the student needs to pick a topic; that is the first choice. Note that this step allows the student to personalise the game as well. Next, a general story line needs to be designed. It is less obvious that this step is a choice, but it can be redefined as a choice with an infinite amount of options. As the students start programming their game, they have the opportunity to add in details, possibly personalising the game as they go. However, game building is more than just a collection of choices, because in the process of making these choices students are producing a personalised sharable object. Good and Robertson (2006), which was discussed in Section 3.3.2, suggest that this aspect of game building takes Papert’s constructionism one step further:

*“While constructionism may suggest that the mere fact of producing **something** is likely to benefit learning, we would suggest that producing **something which is deemed to have value for the producer**, either at a personal or cultural level, is likely to positively affect both learning and motivation even more directly.”* – from Good and Robertson (2006).

They base their claim partially on the work of Pfaffman (2003), which aims to capture the motivational aspects related to hobbies and use them to improve computer-based instruction. Pfaffman found that the primary motivation for a person to engage in

a hobby is the opportunity to produce artifacts, where the definition of artifact includes performances, collections, and objects. Also high on the list were the opportunity to share this artifact, learning the methods necessary to produce the artifact, and being able to personalise the artifact.

This is of particular relevance to the proposed approach, since our participants will be building a game situated in a context *of their choice*. By allowing participants this freedom, they are given an opportunity to draw on their interests and to ground the lesson material in a context which is meaningful to them personally. So besides creating a *public entity* as mentioned by Papert, they are also creating an artifact *which is deemed to have value for the producer*, as mentioned by Good and Robertson (2006). Incidentally, the game-building task by definition contains many opportunities for personalisation and the resulting game is inherently suitable for sharing. These are both motivations for a person to participate in an activity for its own sake, as reported by Pfaffman (2003).

3.4.4 Programming

In the article *How Software Companies Die* (Card, 1995) from Orson Scott Card's series *Windows Made me this Way* we learn Card's view of the inner psyche of the professional computer programmer:

"You're aware that some people think you're a nerd. So what? ... Like soldiers or artists, you don't care about the opinions of civilians. You're building something intricate and fine. They'll never understand".

This quote brings us to the work of Lakhani and Wolf (2003). The authors describe a study into the different reasons people have for contributing to open-source projects. This study is relevant to our research because it sheds some light on the reasons behind voluntarily engagement in the activity, programming, that we will have our participants do in our study.

A central concept in Lakhani and Wolf (2003) is that of *flow*. Introduced in Csikszentmihalyi (1975), *Flow* is the state in which we find ourselves when we are participating in a enjoyable activity and we are 100% engaged in it. This state is frequently accompanied by losing track of time, feeling happy, and not wanting the activity to end. Lakhani and Wolf (2003) found that not external factors, but *enjoyment-based intrinsic motivation* is the strongest drive for contributing to open-source projects and that many programmers experience *flow* while doing so.

In education, *flow* this is the ideal state for a student to be in, most of the time. Ironically, education is one those areas (paid jobs being another) in which a state of flow very seldom occurs. One possible explanation for this could be the detrimental effect of extrinsic motivation (awarding grades) on intrinsic motivation, which was mentioned earlier in the chapter.

However, in the proposed approach, we hope to increase the *enjoyment-based intrinsic motivation* mentioned in Lakhani and Wolf (2003) by increasing the amount of control available to the student.

3.5 Game-Based Learning

Game-based learning generally refers to learning-by game-playing, rather than the *learning by game-building* that is the topic of this thesis. Although the two are different in many respects, they also share several attributes. For example, both have the potential to exert the video-game appeal, particularly to young people, and both generally contain an element of *fantasy*, with the exception of some games, e.g. casual games. This section discusses the literature on game-based learning, focussing on the differences and similarities of game-playing versus game-building.

Advocates of game-based learning claim that (educational) games incorporate pivotal learning principles which can and, they claim, should be applied to non-game-based learning as well. For example, Malone (1981) describes a study comparing several different educational games. *Challenge*, *fantasy*, and *curiosity* are named as the three major factors contributing towards intrinsically motivating instruction. We discuss each one in turn.

Challenge Malone (1981) defines challenge in terms of goal attainment. The author showed that people are more likely to enjoy a game if it has an explicit goal. Based on previous research by others, Malone (1981) claims that in order to invoke intrinsic motivation, the goal attainment should be uncertain and that performance feedback should be provided which engages and enhances the self-esteem of the individual. Furthermore, they claim that previous research suggests that people prefer an optimal level of challenge and similarly, a probability of success near 0.5. Game-building tends to a more open-ended activity than game-playing, which could give the impression that the latter is more focussed on goal attainment than the former. However, this is not necessarily true.

First of all there are open-ended games like *The Sims*, which exceed most goal-oriented games in popularity. Second, in the approach presented in this thesis, the game-building process is guided by a set of requirements (see Section 4.3.2) that a student's game has to satisfy, thus creating a solid framework of subgoals around the game-building task.

Fantasy Malone (1981) distinguishes between two different kinds of fantasy in an educational context, *intrinsic* and *extrinsic*. We came across this distinction earlier in Section 3.3.2. In extrinsic games there is no integration of learning content with the fantasy context, whereas in intrinsic games there is. Malone (1981) hypothesizes that the latter will invoke more intrinsic motivation than the former. Furthermore, they claim that some fantasies of intrinsic games can provide a valuable educational metaphor by linking existing knowledge to the material to be learned. Because our study is concerned with game-building rather than game-playing, there is an immediate opportunity for students to provide their *own* fantasies. In doing so, they may be providing a valuable educational metaphor, as described by Malone (1981), linking their *personal* existing knowledge to the material to be learned.

Curiosity Malone (1981) claims that people have a cognitive drive for their knowledge structures to be well-formed and they will exhibit intrinsic motivation when presented with an opportunity to improve their knowledge structures. This effect is hypothesized to be enhanced if a person is already interested in the topic. Game-building could have an edge over game-playing in this respect as people can design a game on the topic *of their choice*, whereas they are presented with pre-defined content in a game-playing context.

In Malone and Lepper (1987), the original 3-way taxonomy is extended with one more '*individual*' motivation called *control* and three '*interpersonal*' motivations called *cooperation*, *competition*, and *recognition*. We discuss each one in turn.

Control Malone and Lepper (1987) define control in terms of (1) contingency, (2) choice, and (3) power. In terms of instructional environments, this means that to invoke intrinsic motivation, the environment should (1) be responsive to the user's actions, (2) provide a moderate amount of choice, and (3) allow the user to produce powerful effects. The second item is of particular interest to our study,

because four out of the six central question of this thesis are concerned with personalisation and choice. Malone and Lepper (1987) hypothesize that a moderate amount of choice will be most effective in enhancing intrinsic motivation. Too few choices would prevent the user from feeling in control, but too many would overwhelm them and leave them feeling frustrated instead of satisfied. In our studies two experimental conditions were used: (1) no choice and (2) a large but bounded amount of choice. According to Malone and Lepper (1987) the second condition could cause students to feel overwhelmed with the amount of available choice. However, all the available choice in our studies is defined in terms of a single (game-)design task restricted by a set of requirements (see Section 4.3.2). This makes the choice used in our studies different from the more general concept of choice as in: these are the x options, please select y of them, where x and y represent numbers. Therefore, any overwhelming effect invoked by too much choice in terms of the latter, may not necessarily hold for the former. For example, it is possible that presenting a person with a list of one hundred possible names for their character in a video-game can be overwhelming compared to a list containing only five. However, asking a person to provide *any* name for their character as specified by them is not necessarily more overwhelming than a list of five names. Malone and Lepper (1987)'s factor of control bears strong similarities with Deci and Ryan (1991)'s concept of *self-determination*.

Cooperation This is the motivation to cooperate with others. Malone and Lepper (1987) hypothesized that segmenting an activity in inherently interdependent parts will enhance the intrinsic motivation resulting from cooperation.

Competition This is the motivation to compete. Malone and Lepper (1987) mention that this factor is traditionally viewed a 'bad' source of motivation with *cooperation* being the 'good' source of motivation. However, the authors defy the traditional viewpoint and claim that it can be implemented in ways that have beneficial effects. As an example, they mention two or more students with conflicting goals working on interdependent tasks.

In this thesis, a competition option is available to the participants in Study 2. By comparing other data, e.g. a participant's motivation type, against their choice for or against competition, we hope to provide some insight into the kind of student who chooses to add a competition element to their task, and those who do not.

Recognition Malone and Lepper (1987) hypothesize that intrinsic motivation may be enhanced if the results of a student's activities are visible to other people. The authors identify three different ways of doing this: (1) the process may be visible, e.g. artists painting in public, (2) the product may be visible, e.g. paintings in an exhibition, or (3) some other result may be visible, e.g. published names of prize winners in a contest. These examples were taken directly from Malone and Lepper (1987). In this thesis, a publication-only option is available to the participants of Study 2. That is, the participants choosing this option share their game with their peers, but they do not enter the competition. By comparing other data, e.g. a participant's motivation type, against their choice for or against publication-only, we hope to provide some insight into the kind of student who chooses to add a recognition element to their task, and those who do not.

Malone and Lepper (1987) mention that the factors in their taxonomy play a role not only in the context of educational games, but in all instruction which aspires to be intrinsically motivating. Lepper and Malone (1987) presents some empirical evidence in support of the taxonomy. However, not all learning principles had the same motivational effect on all participants. In fact, some had opposite motivational effects on different participants. For example, introducing competition may enhance motivation for participants with high self-esteem, but decrease that of participants with low self-esteem. Furthermore, the authors found that an enhanced motivational appeal of games used in the study did not produce enhanced learning.

Gee (2003) took a similar approach to that of Malone and Lepper (1987). In his book "What Video Games have to Teach us about Learning and Literacy" he describes 36 learning principles which he believes to be present in good commercial games. He also claims that most of these learning principle are currently not present in most forms of formal education and that the situation needs to change. However, this does not mean that all subjects should be taught using educational games. For example, Learning Principle 16 (Multiple Routes Principle) says:

***“Learning Principle 16:** There are multiple ways to make progress or move ahead. This allows learners to make choices, rely on their own strengths and styles of learning and problem solving, while also exploring alternative styles”.*

There are several ways to incorporate this principle outside the context of educational games. One of them is offering choice when setting exams or assignments. We believe that our research incorporates at least 24 out of the 36 learning principles described in

Gee (2003) including the Multiple Routes Principle. Another example is the Insiders Principle:

“Learning Principle 36: The learner is an ‘insider’, ‘teacher’, and ‘producer’ (not just a ‘consumer’) able to customise the learning experience and domain/game from the beginning and throughout the experience”.

This principle holds for our game-building approach, because the students are *producing* a game of their choice, and more importantly they are *producing* an object of significant size and complexity in the appropriate modelling framework along with it.

However, Gee (2003) does not present any empirical evidence for the fact that his learning principles support learning. Therefore, the fact that our approach satisfies a large number of them is not necessarily meaningful.

Approaches that are more oriented towards the use of actual games, educational or off-the-shelf, in an educational context can be found in Prensky (2001); Squire (2004); Garris *et al.* (2002); Aldrich (1995). Prensky’s and Aldrich’s books are among the few publications on game-based learning which covers the topic of adult learning. In fact, game-based learning in a corporate environment is the focus of both these books. Toward the end of his book, Prensky mentions that regardless of the approach, there will always be people who dislike it. He suggests providing alternative approaches which accomplish the same learning objectives. Unfortunately, these books are quite squarely aimed at industry and provided little or no justification for the ideas presented in them.

Squire (2004) experimented with the off-the-shelf simulation game Civilization 3 as part of the history curriculum. Similar to Prensky (2001), Stenning (2002) (as described in Section 3.2), and Lepper and Malone (1987), one of Squire’s conclusions was that no one way of teaching suits everybody and different people thrive in different environments. In some cases, game-based learning allows children to live up to their full potential, which they had not managed to do in a traditional teaching framework for various reasons, e.g. lack of motivation or lack of identification with the course material.

However, there are also children that thrive in that same traditional teaching framework and would perhaps not be able to reach their full potential without it. We expect to find a similar result in our research, that is, we expect our constructionist approach to have a positive effect on some students, but not on others.

Finally, Garris *et al.* (2002) describe a new research model for game-based learning, in terms of game-playing, based on an amalgamation of previous literature. The

authors list the key features of games which they believe are of interest from an educational perspective: (1) fantasy, (2) rules/goals, (3) sensory stimuli, (4) challenge, (5) mystery, (6) control. Based on this taxonomy, Garris *et al.* (2002) describe ‘the game cycle of user judgements, behaviour, and feedback that is a hallmark of engagement in game play’. Besides previous research no evidence is provided in support of this game cycle though. Finally, the authors describe the types of learning outcomes that can be achieved through game-based learning, again based on previous research. They conclude that in Bloom’s taxonomy (Bloom, 1956, as described in Section 3.2.1), game-playing can promote competence all three domains of learning outcomes: (1) cognitive, i.e. mental skills, (2) affective, i.e. feelings, values, attitudes, and (3) psycho-motor, i.e. manual or physical skills. Within the cognitive domain, competence can be achieved at various levels. Unfortunately, Garris *et al.* (2002) give few details and fewer examples of competence in the cognitive domain. Furthermore, it is very much tied to a game-playing context, rather than a more general instructional environment context which is generally adapted in the literature. This makes it very difficult to infer information about anything other than game-playing environments from the paper. As such, the main contribution of Garris *et al.* (2002) is not easily adapted for a game-building environment.

All of the publications mentioned in this section provide, in one way or another, support for the claim that game-based learning enhances student motivation. However, in this study we are hoping to find an additional result. We are hoping to find that game-based learning, in a game-building context where students can build a game of their choice, enhances student learning outcome as well as motivation. There is some support for this claim in Cordova and Lepper (1996), as discussed earlier in Section 3.4.3, which describes a study into the effect of personalisation and choice on affect and learning outcome. The authors used customized versions of the existing program ‘How the West was One + Three \times Four’. In this game, players are asked each turn to combine three randomly generated numbers in an arithmetic expression. The outcome of the expression determines how many steps the player is allowed to advance on the game board. The goal of the game is to reach the target square, fifty.

Cordova and Lepper (1996) used five different versions of the game: (1) the original game, (2) the original game in a space fantasy context, (3) the original game in a space fantasy context with personalisation, (4) the original game in a space fantasy context choice, and (5) the original game in a space fantasy context with personalization and choice. The personalization was offered by asking participants to provide

personal details, e.g. the name of their best friend or their favourite food, which was used in the introduction space story to the game. The choice was offered by letting participants make choices on game details like the picture that was to represent their space ship. All choices, personalizations and fantasies that were offered covered only instructionally irrelevant aspects of the educational computer game.

The authors found that all three of the experimental conditions – contextualisation, personalisation, and choice – increased both motivation and learning outcome. These results are encouraging for our study, because like the educational games that Cordova and Lepper (1996) experimented with, game-building tasks also afford personalisation, choice, and contextualisation. However, the results of Cordova and Lepper (1996) should be contrasted with studies like Parker and Lepper (1992) in which the available choice was found to have no effect on learning outcome. However, the addition of a fantasy context was found to have beneficial effects on learning outcome by Parker and Lepper (1992).

Besides advocates, there are those that oppose game-based learning (and video games in general). As a spokesperson for its adversaries, (Provenzo, 1991) discusses four major objections against game-based learning, and computer-games in general. That is, video-games (1) can lead to aggressive behaviour, (2) encourage gender stereotyping, (3) promote ‘rugged individual’ attitudes, and (4) inhibit creative play. Provenzo bases his claims on a review of the 10 most popular video games at the time, but he does not present any strong evidence in support of these claims. He does point out the need for further research. This discussion re-iterates an important difference between game-building and game-playing; the act of game-building is of an entirely different nature than that of game-playing. That is, when a child *plays* a video game, (s)he is exposed to the content of the game, which could potentially influence the attitude of the child. However, when a child *builds* a video game, there is no such content to bring about a change in attitude. Furthermore, we believe that game-building encourages, rather than inhibits creative play. Therefore, the arguments presented by (Provenzo, 1991) against game-playing can not easily be extended to included game-building.

Adversaries of game-based *learning* in particular have raised different issues. For example, Stoll (1999) argues that computer games do not belong in the classroom because, so he claims, they cannot replace teachers or even improve learning. Furthermore, he claims that a lot of learning is not fun and neither should it be, because a lot of jobs are not fun either. Stoll (1999) presents a very traditional view of education and evidence against most of his claims can be found in the literature, e.g. Cordova

and Lepper (1996); Lepper and Malone (1987).

Another line of argument against game-based learning is expressed in Healy (1998). The author argues that games should not be used in the classroom, because there is not enough research into the long-term effect of video-game exposure. She claims, that it is unclear if video-game exposure increases a child's risk of depression and whether or not it affects the immune system. While this a valid criticism of any new approach and can be investigated if this is thought to be appropriate, it should not stand in the way of progress.

Finally, Merrill (1999) claims that there is too little learning and too much enjoyment taking place in game-based learning. This is in effect another form of the overhead problem mentioned in Section 3.3.3. While this may be true for some educational games, it depends heavily on the game in question. Furthermore, how do we determine what 'too much enjoyment' is? Consider a game which is 10% learning and 90% enjoyment. This game does not seem to have a very high educational value. However, if a child chooses to play the game after school, just because it is fun, does this really matter? Clearly, this issue is a complicated one.

For game-building, a similar issue exists. There could be some overhead caused by the game-building framework, but if the overhead is relatively small and it causes a student to want to work on the game-building task in their spare time, does the overhead really matter?

To summarize this section, the relationship between game-building and game-playing can be characterized as being concerned with the same type of entity, i.e. a video-game or interactive story. The difference between the two lies in their interaction with this entity: in the game-playing scenario, students are '*consuming*' the interactive story, whereas in the game-building scenario, they are creating it. As a result, learning by game-playing exposes students to pre-defined content, while learning by game-building does not. Instead, it allows students a significant amount of control and opportunities for personalisation that is simply not present in the game-playing context.

On the other hand, the game-building and game-playing scenarios share the video-game appeal (and similarly, the video-game aversion), they generally share an element of fantasy, and they are both concerned with a shareable entity, i.e. students can lend the games to their friends.

3.6 A Word on Transfer

Latin used to be taught in schools because it was believed to discipline the mind, preparing people for better performance in other subjects. Thorndike and Woodworth (as cited in Perkins and Salomon, 1992) showed that this belief is ill founded. They compared the performance of students who had taken Latin to the performance of those who had not, in other subjects. Thorndike found no advantage of taking Latin; i.e. no *transfer* was found. Earlier, Thorndike (as cited in Perkins and Salomon, 1992) described one of the first studies into the existence of transfer. He generally failed to find any.

However, not all research into the subject of *transfer* is of a negative nature. For example, Bransford and Schwarz (1999) paint a more positive picture by challenging the way we test for transfer: they claim that transfer *does* take place, it just fails to be captured by our current tests. The authors claim that transfer should instead be evaluated in terms of the questions people are able to ask themselves when confronted with a new problem. The example mentioned by Bransford and Schwarz (1999) concerns a recently graduated primary school teacher on the first day of their first job. A traditional measure of transfer would be to measure the performance of this teacher on their first day at work. An example of the sort of measure proposed by Bransford and Schwarz (1999) would be the extent to which the teacher manages to improve their performance over a certain period of time.

Furthermore, Bransford et al., as reported by Bransford and Schwarz (1999), sheds some light on the benefits of using concrete examples to teach abstract concepts. The authors found that concrete example have the ability to facilitate initial learning, because the students can appreciate the relevance of the material. The authors also found that presenting a concept in several different contexts has the ability to increase subsequent transfer.

Presenting material in a problem-solving setting can also lead to increased transfer as reported by Adams *et al.* (1988). However, overly contextualized information has the opposite effect, reportedly because the information is too tied to its original context, as described in Gick and Holyoak (1980).

Although we do not use an explicit measure of transfer in our study, it is – to some degree – present in the measure of performance used in Study 3. Section 6.2.2 provides a detailed description of the performance measure used in Study 3.

It could be argued that game-building activities have the potential to facilitate initial

learning, because learners are able to appreciate the relevance of the material, through the concrete example that is their game. On the other hand, if the game-building environment is overly contextualized, game-building activities could have the opposite effect. Finally, if the game-building task is specified as a problem-solving task, game-building activities could lead to increased transfer.

However, the current study does *not* investigate the difference between a game-building task and other tasks; it investigates the difference between a pre-defined game-building task and an own game-building task. These two scenarios differ only in the amount of choice that is offered to the participant. The literature does not provide much insight into the expected difference in transfer, if any, between these two scenarios. Cordova and Lepper (1996), as described in Section 3.4.3, come closest, but the authors do not explicitly mention transfer. Therefore, no formal hypotheses with regard to transfer are specified in this thesis. Still, this is an interesting direction for future research on learning by game-building.

3.6.1 Is Programming the New Latin?

After the introduction of the computer into schools a new transfer question has arisen: does programming discipline the mind, preparing people for better performance in other subjects? In Soloway (1993) a number of researchers including Andrea diSessa and Seymour Papert give their opinion on the question: *Is programming the new Latin?* We believe that this discussion is not relevant to our study. First of all, Soloway himself says “*Ignore for a moment those students who wish to major in computer science; yes, they should learn to program in school . . .*”. Second, we are not teaching the students to program, we are simply taking advantage of the programming skill they already have on entering the course. In fact the approach should not be classified as learning-through-programming, as this is too specific a term. Learning-through-construction or learning-through-modelling would be a more appropriate description of our learning by game-building approach.

3.7 Summary

In this chapter, an overview of the literature relevant to our research was presented. As part of this overview, the contrasts between the popular *game-based learning*, which is *learning by game-playing*, and the proposed approach of *learning by game-building*

were identified.

The overview also showed that increasing control has in the past often led to increased motivation, enjoyment, and even performance. For this reason, we will hypothesize in Chapter 4. that the success of the proposed approach will depend on the amount of available control.

Study 3, described in Chapter 6, tests this hypothesis. In particular, it looks for differences in performance and enjoyment in two conditions which differ only in the amount of control available.

Although a difference in enjoyment is expected, based on the amount of available control, the current chapter did show that in general, no single approach works for all students. As a result, one of our aims will be to find a criterion based on which we can distinguish between those students who do well in a setting with increased control, and those who do not.

A student's motivational type is thought to be one of those criteria, and therefore, several motivational scales were discussed in the current chapter. Based on this discussion, the achievement goal framework was deemed to be the most suitable. Several very similar questionnaires were identified which determine a person's achievement goals. These will be used in Chapters 5 and 6.

Finally, constructionism was encountered in the overview presented in the current chapter and it was shown that our research can be described in terms of this paradigm. In particular, Chapter 4 describes a constructionist approach to teaching TCS, Study 2 investigates the demand for control within a constructionist setting, and Study 3 evaluates the effect of the amount of control available within a constructionist setting. In other words: how much does the fact that students get to build their own structures rather than pre-defined ones contribute to their enjoyment and performance?

The next chapter presents the proposed approach of *Learning by Game-Building* in detail. It shows how the *modelling skills* in the TCS curriculum can be mapped onto a game-building framework and how (a) game-building task(s) might be integrated in an undergraduate course on TCS.

Chapter 4

Learning by Game-Building in Theoretical Computer Science

In the previous chapter, an overview of the literature relevant to our study was presented. Furthermore, Chapter 2 provided evidence in support of the claim that students enjoy TCS less than their other computing subjects, that they find the subject more difficult, and that they find it less relevant to their degree and future career.

In the current chapter a novel approach to teaching TCS, *learning by game-building*, will be presented which claims to address the issues raised in Chapter 2. The approach was influenced by results from previous research presented in the previous chapter, especially the research in the area of constructionism, and the effect of personalization and choice. Various examples covering a range of topics in the TCS curriculum are included. Following the current chapter are two chapters on the empirical evaluation of *learning by game-building*.

4.1 Introduction

As mentioned in Section 3.3.2, learning by game-building has previously been applied in several different areas. However, theoretical computer science (TCS) does not appear to be one of those areas. In this chapter and the next, it is argued that TCS is a particularly suitable domain for learning by game-building, and that valuable educational opportunities are being missed out on by ignoring the potential of learning by game-building in the area of TCS.

The claim that TCS is a particularly suitable domain for learning by game-building is based on the fact that most TCS topics commonly taught at undergraduate level

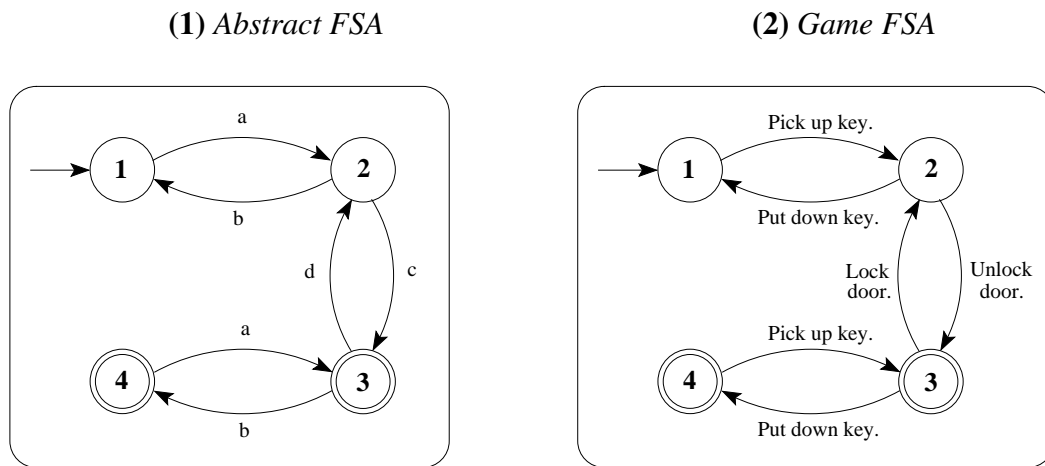


Figure 4.1: Two Different Versions of the Same FSA.

are concerned with some kind of model-building, as was shown in Chapter 3.2. The same holds for game-building: you are building a model of the game-world. The current chapter will elaborate on this close relation between TCS and game-building by presenting several examples of TCS topics and explaining for each of them how they could be taught using a learning by game-building context.

4.2 Examples

To show how straightforward the mapping between theoretical computer science and game-building really is, four example topics within TCS will be presented and it will be demonstrated how these could be taught using learning by game-building. The first example is on finite state automata (FSAs), the other examples are: context-sensitive grammars (CSGs), Turing machines (TMs), and concurrency.

4.2.1 Finite State Automata (FSAs) and Regular Expressions (REs)

Finite state automata (FSAs) model behaviour. In their simplest form, they are composed of states and transitions between these states. Consider the first FSA in Figure 4.1. In its *initial state*, the state with the incoming horizontal arrow, there is only one transition possible: *a*. If this transition is executed, one would end up in State 2. From this state, there are two options: (i) a *b*-transition back to State 1, or (ii) a *c*-transition to State 3. If a choice for the latter is made, you end up in a state with an extra circle inside it. States with extra circles are the *final states* or *accept states* of

an FSA. If you end up in one of them, you can either choose to execute one of the transitions available in that state, or you can choose to terminate the behaviour. A path from the initial state to a final state (possibly with many detours) represents a correct behaviour for the FSA in question.

Interestingly, FSAs can also be used to describe languages. Instead of transitions for actions, there can be transitions for parts of expressions/words/sentences. A path from the initial state to a final state (possibly with many detours) represents a correct expression/word/sentence. An FSA M represents a language L if M accepts all words in L and none that are not in L .

The FSA framework is of interest to computer scientists for two reasons: (1) because they can be used to specify program behaviour and (2) because the class of languages that can be described by an FSA is the class of regular languages. The class of regular languages is the smallest class in Chomsky's hierarchy of languages (see Figure 4.2). Chomsky's hierarchy describes the set of all computable languages: every language is a Type-0 language; a subset of the Type-0 languages is also a Type-1, or context-sensitive, language; a subset of the Type-1 languages is also a Type-2, or context-free, language; and a subset of the Type-2 languages is also a Type-3, or regular, language. Language classes lower in the hierarchy have less expressive power, but are more attractive from a computational point of view. For example, parsing and testing can be done in linear time for Type-3 languages, but is expensive or even intractable in the other sets. It is important for computer scientists to know the boundaries of the different classes in Chomsky's hierarchy of languages so that they can always define the problem they are working on in the lowest class possible, because it will bring significant computational benefit.

Regular expressions (REs) are closely related to finite state automata, because they can both be used to express any language in the set of all Type-3 languages, but no language outside this set. Any FSA can be converted into a RE and vice versa.

Looking back at Figure 4.1, notice that instead of using the first FSA as the example at the beginning of this section, the second FSA could have been used as well. In fact, many people would have probably preferred the second FSA. As we discussed Chapter 3.6, concrete examples can facilitate initial learning, because the students can appreciate the relevance of the material more.

The second FSA illustrates this concept. It is much less abstract and because of its labels, it represents a very simple adventure game. It is indeed exactly the same as the first FSA except for the labels on the transitions: label a has been substituted by *Pick*

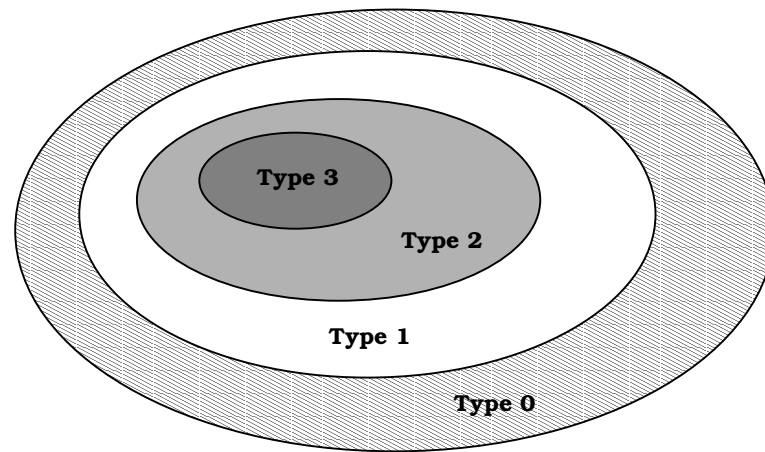


Figure 4.2: The Chomsky Hierarchy.

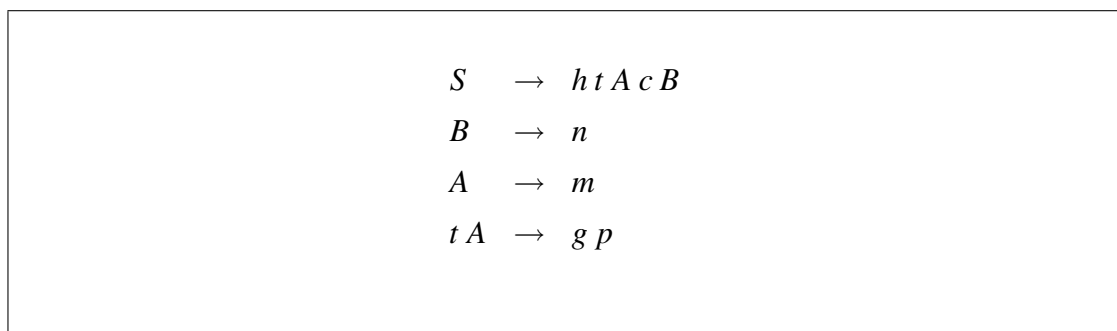


Figure 4.3: A Simple CSG.

up key, *b* by *Put down key*, *c* by *Unlock the door*, and *d* by *Lock the door*. From the actions the following information can be inferred: State 1 represents the state in which the door is closed and you do not have a key, in State 2 you have a key, but the door is still closed, in State 3 the door is finally open and you have the key, in State 4 the door is also open, but you no longer have the key. Furthermore, it can be inferred that the goal is to open the door, since States 3 and 4 are both final states.

In order to construct larger games, students can combine several parts of their game using FSA-combinators such as union, intersection, interleave, and closure. During this process they will have an opportunity to learn about the relationship between combinators, the FSAs they produce, and the REs they represent.

4.2.2 Context-Sensitive Grammars (CSGs)

Looking back at the Chomsky hierarchy in Figure 4.2, we could ask the following question: where do programming languages and natural languages fit? Neither can be

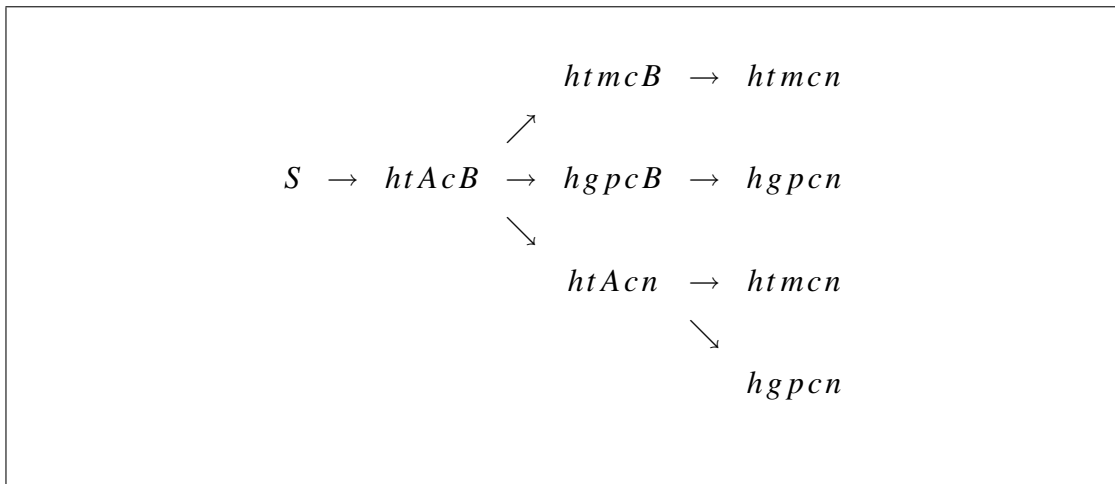


Figure 4.4: Possible Derivations.

parsed by FSAs or REs, therefore they are not Type-3 languages. Type-2 languages are more expressive than Type-3 languages and can be parsed using Context-Free Grammars (CFGs). However, natural languages and even most programming languages are more expressive still. Nearly all natural languages are Type-1 languages. This means they can be parsed using Context-Sensitive Grammar (CSGs). The majority of programming languages are actually *mostly* context-free, but do contain some context-sensitive elements.

Even natural languages use only a fraction of the expressiveness of CSGs though. As mentioned before, with increased expressiveness comes increased complexity. In fact, parsing a Type-1 language can take exponential time. Therefore most practical parsers for both natural and programming languages use a restricted form of context-sensitivity. Students of theoretical computer science need to be aware of the complexity issues involved in writing parsers.

The CSG framework is defined by a rewriting system. For example, Figure 4.3 contains a simple CSG with which only two different words can be formed. The forming of these words happens like this: an occurrence of an expression on the left-hand side of an arrow can be substituted by its counterpart on the right-hand side. Furthermore, S is always the start symbol. Figure 4.4 shows how this process results in the two possible words: *htpcn* and *htpcn*.

Instead of Figure 4.4, Figure 4.5 could have been used as an example. Figure 4.5 represents a line up for the guillotine during the French revolution, based on the existing card-game Guillotine (Peterson, 1998). In this two-player game the players play two rival executioners. They are placed on either end of the line and will start chop-

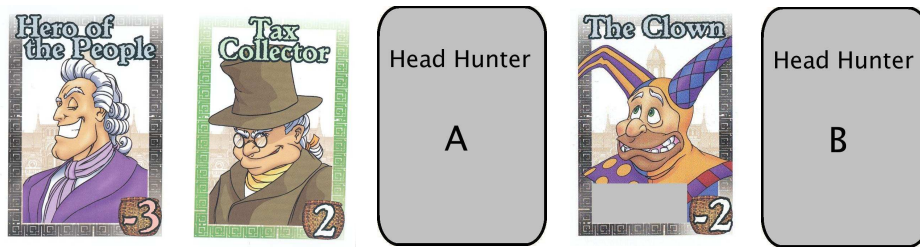


Figure 4.5: CSG Game: the Line-Up.



Figure 4.6: CSG Game: The Full Cast.

ping off heads as soon as all the head hunters have returned with people who have been sentenced to death. Each head is worth a certain amount of points as indicated on the cards. The goal is to get as many points as possible. Figure 4.6 show the full cast of the game.

The rules of Figure 4.3 will be used to play the game. The variables, which are called *non-terminals*, represent head-hunters and the constants, which are called *terminals*, follow the obvious mapping: p is a piss boy, t is a tax collector, c is a clown, n is a bad nun, m is a martyr, h is a hero of the people, and g is a general. For example, in the context of our game, the last rule in Figure 4.3 means that a tax collector may be sent out with head-hunter A (in that order) to retrieve the general and a piss boy (once, and in that order). Each player gets to send out one head hunter (team) each turn. Figures 4.7, 4.8, and 4.9 present all three possible scenarios for the game. Scenario 1 is clearly most advantageous one for the player on the right hand side of the line. This player will have the heads of the nun (3), the clown (-2), and half of the martyrs (-0.5), which is a total of 0.5. The left player would have -1.5 points. Scenario 2 is not much better from the perspective of the player on the left, because in this scenario he will always get -1.5 points whereas the player on the right will get 0.5 points. Scenario 3 is not a good one from either player's perspective, because the result of the game is left up to the other player. In a nutshell, the result of this very simple game depends entirely on who gets to start. However, one can imagine a similar game with a more uncertain outcome by adding only a few more rules to the CSG in Figure 4.3. You could have multiple jobs – which would translate to multiple rules – for the same headhunter or headhunter-team, more than one bad nun, headhunters retrieving more than one person, headhunters retrieving other headhunters, the possibilities are endless and the game can be made as complicated as desired.

Of course the example of the guillotine game is just one of many games that can be implemented using CSGs. Another possible example is based on the old missionaries and cannibals puzzle. In the original puzzle, three missionaries and three cannibals come to a river and find a boat that holds two. If the cannibals ever outnumber the missionaries on either bank, the missionaries will be eaten. How shall they cross? In our game one player will play the missionaries and the other will play the cannibals. There are three different kinds of terminals: cannibals, missionaries, and rivers. For every group in which the missionaries outnumber the cannibals and there is at least one cannibal, the missionaries get one point i.e. they convert the cannibal. On the other hand, for every group in which the cannibals outnumber the missionaries and there is

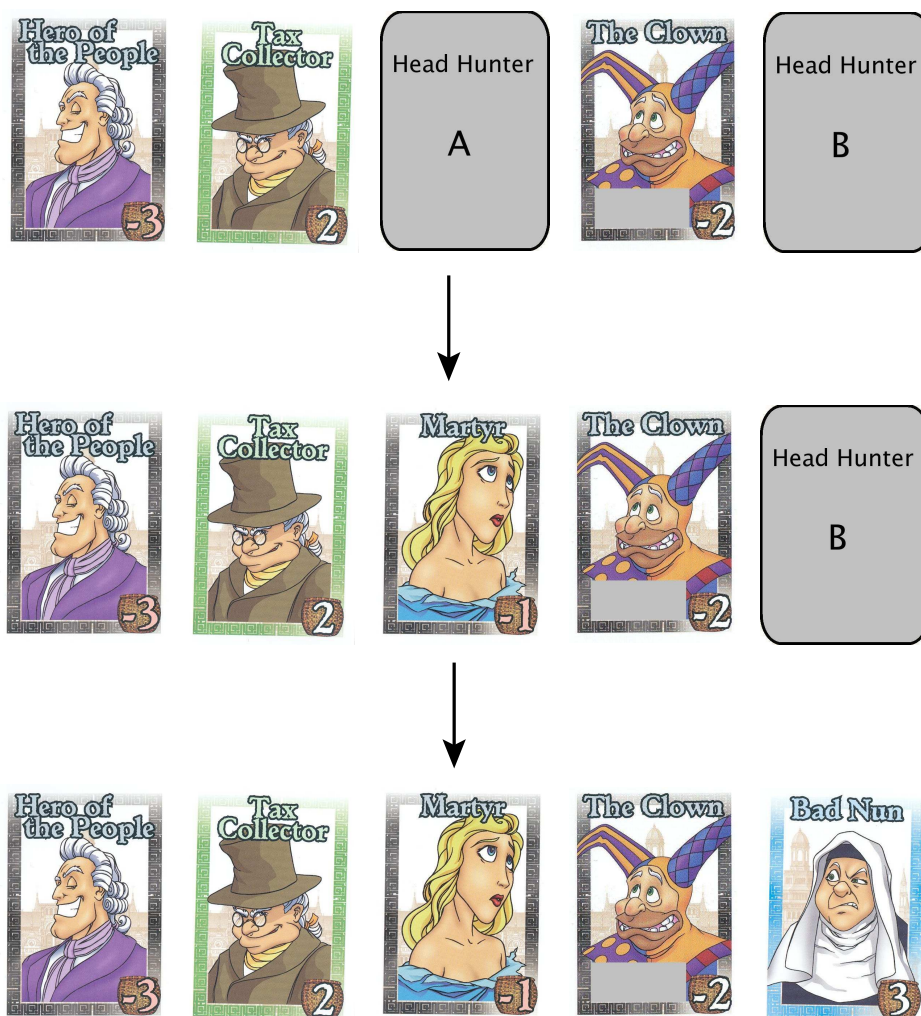


Figure 4.7: CSG Game: Scenario 1.

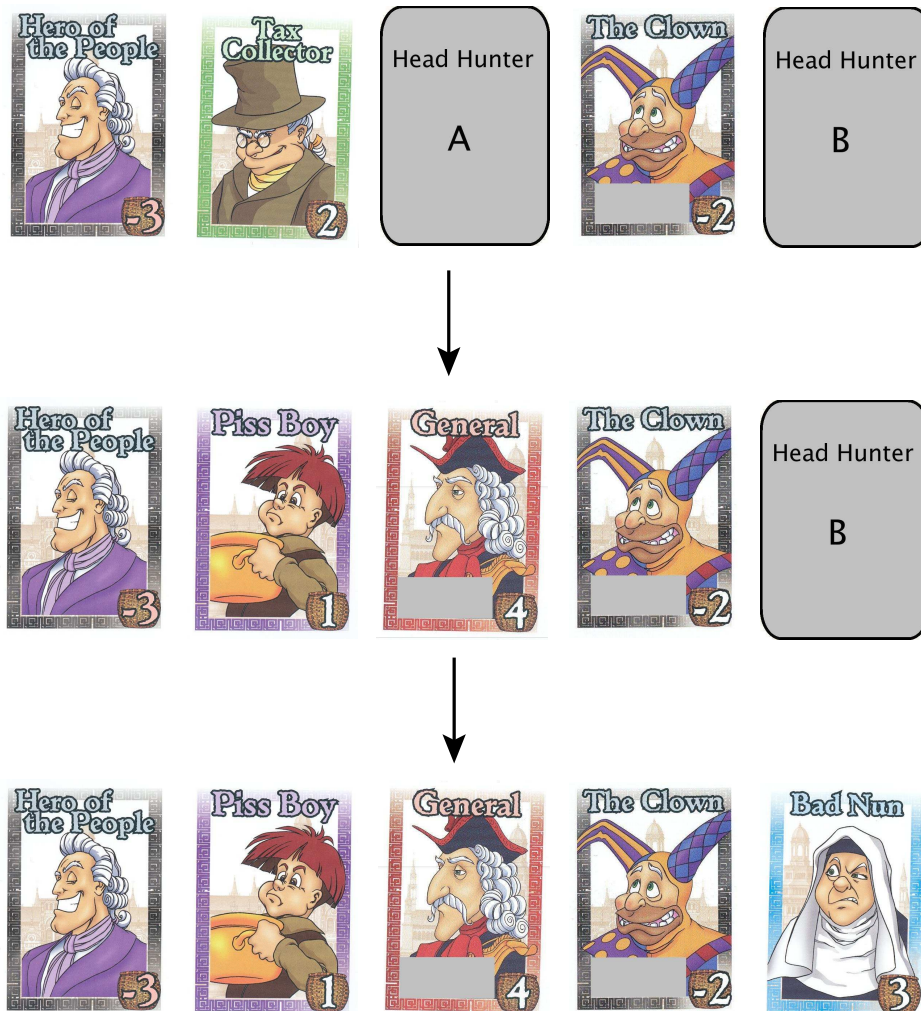


Figure 4.8: CSG Game: Scenario 2.

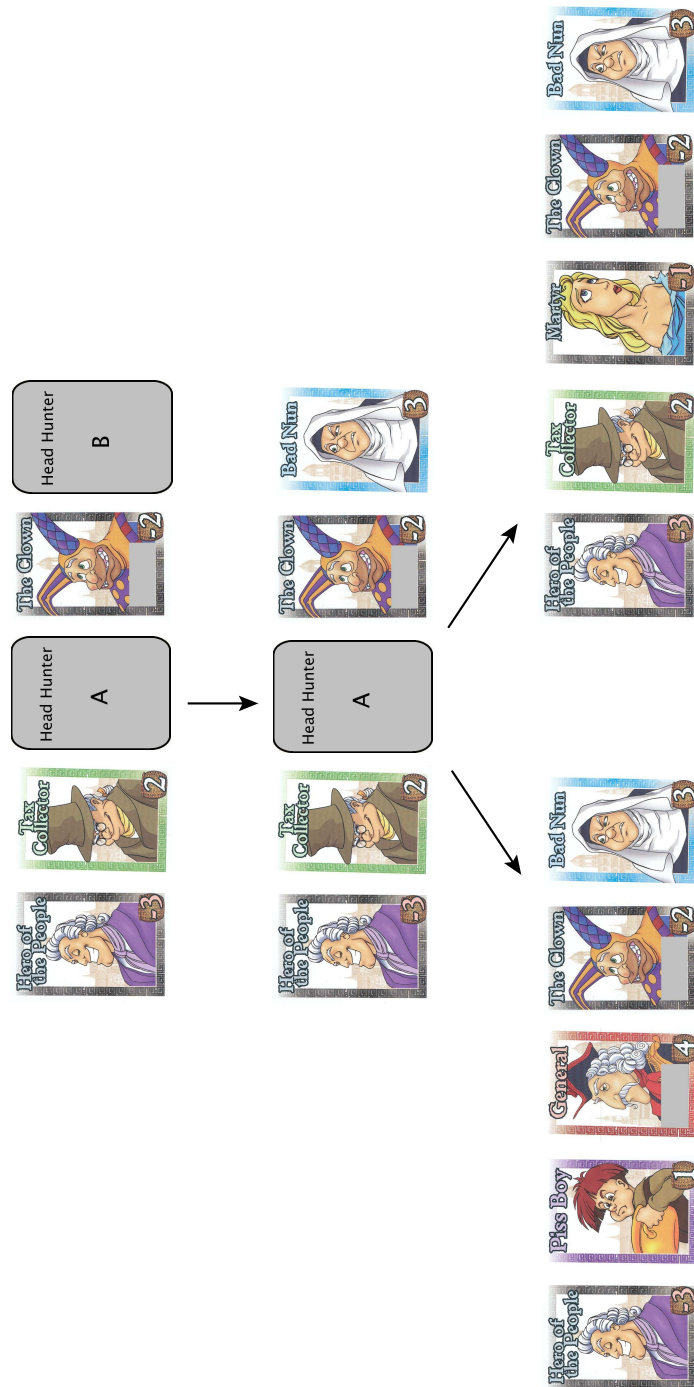


Figure 4.9: CSG Game: Scenario 3.

at least one missionary, the cannibals get one point i.e. they eat the missionary. A group is defined as a maximum row of connecting cannibal and missionaries cards. So in the row of cards mm r mmmccc r ccm, there are three groups and the cannibals win with 1–0.

Many more similar games can be created in this manner. Anything which involves a row of cards that can be manipulated through a set of rules, and where each player has a different configuration of the final row as their goal is a candidate.

4.2.3 Turing Machines (TMs)

Like FSAs, TMs are commonly found in a TCS curriculum and are therefore the topic of the second part of our second study. Turing machines are similar to FSAs in that they have states and transitions between these states. However, unlike FSAs which do not have any memory, TMs have unlimited memory in the form of an infinite tape. Figure 4.10 shows a TM which accepts an input – read from the tape – if and only if it is a member of the language described by the regular expression ba^* . The current state is 0 and the input $baab$. The program of a TM looks and behaves very much like an FSA, but instead of simple labels, a TM has rules on its transitions. For example, in Figure 4.10 on the transition from state 0 to state 1, we see the rule $b \rightarrow \epsilon, R$. This means that if the TM is in state 0 and it reads a ‘ b ’ on the tape, it will replace that ‘ b ’ with an ‘ ϵ ’ (shown on the tape as ‘.’) and then move its head to the **(R)**ight. The head with which the TM reads symbols from the tape is shown in Figure 4.10 as an upside-down triangle just above the tape. It so happens that our TM is indeed in state 0 and its head is currently reading a ‘ b ’, so this ‘ b ’ will indeed be replaced by a ‘.’ and the TM’s head will move one to the right. The TM is now in state 1. The program will continue until it either reaches an accept state like state 2 in our example, or until it is stuck. The latter happens when the TM lands itself in a situation for which no appropriate rule is specified. An example would be if the TM is in state 1 and its head reads a ‘ b ’¹. No rule specifies what to do in such a situation, so the TM is stuck and fails. In our example this would mean that the input string was rejected.

Amazingly, Turing machines can do everything a desktop computer can do. In fact, it technically outperforms any desktop, because it has unlimited memory and desktop computers do not. The Church-Turing thesis states that anything that can be computed, can be computed using a TM which is the reason they are commonly found on TCS

¹This actually happens when you continue with the example in Figure 4.10.

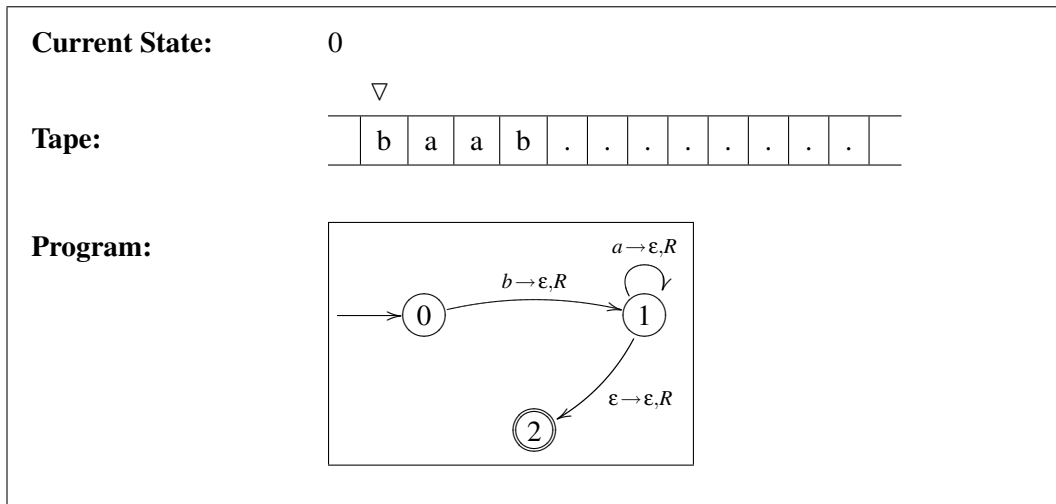
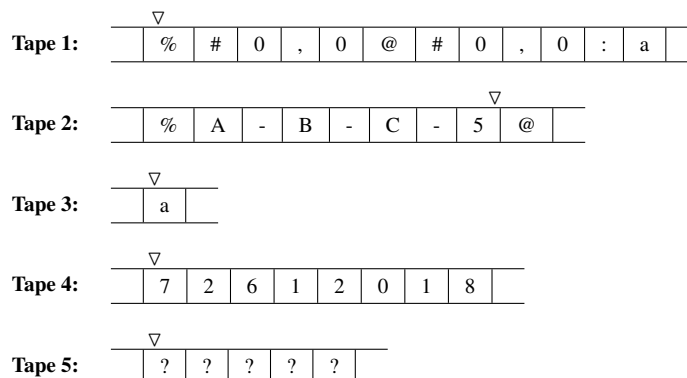


Figure 4.10: Example Turing Machine.

curricula. For practical purposes, TMs with more than one tape are often used. These are known as *multi-tape* Turing machines. In our game-building example we will make use of a 5-tape Turing machine. It is important to note that any multi-tape TM can be simulated by a single tape TM; they are equally expressive. The only reason for using more than one tape is to make programming the TM a little easier. The game-building example which follows, should illustrate this point.

Unlike FSAs, TMs do not lend themselves quite as naturally to a game-building approach. They are more expressive than FSAs though, so at the very least we could apply the same mapping we used for FSAs. However, in order to use the full expressiveness of TMs, a slightly different approach is taken. Using a 5-tape TM, the student is provided with a basic game framework within which they can design their own game. Each of the five tapes has a specific function, which are best explained using their initial values as shown below.



Tape 1 is a representation of the game-world. The game world consists of a Cartesian grid with (0,0) as its origin. The '#'-character indicates the beginning of a single

pair of location coordinates. To the left of the '@'-character, the coordinates of the current location are stored on the tape. To the right of the '@'-character, the coordinates of all other known locations are stored, together with any information that is known about the location. As you can see on Tape 1, initially (0,0) is the current location and in fact the only known location, as there are no other coordinates to the right of the '@'-character.

The information about each location is stored inside a single character, the meaning of which can be looked-up in a dictionary defined by the student/game-builder. For example, we have defined the 'a' following the coordinates (0,0) to mean the 'home village'.

The game-world gets populated with new locations as the player plays the game. Each time he moves to a non-existing square, a new location is created with the specified coordinates and assigned a location-type (e.g. 'home village', 'woodland', 'mountains', 'lake', etc.), possibly non-deterministically. The new location together with its type – e.g. #1,0 : w – gets stored on Tape 1, immediately behind the last entry currently on it. This particular feature demonstrates the unbounded nature of TM computation, since new locations can be added any number of times and the game-board can in theory be extended indefinitely.

Tape 2 has two functions: it (a) stores the values of game variables (e.g. possession of objects A, B, and C or health) to the left of the '@'-character and (b) the part of the tape to the right of the '@'-character serves as the working tape on which the TM can do its more complicated computations. The '-'-characters behind each of the characters A, B, and C indicate that the player currently does not own any of these three objects. The '5' immediately preceding the '@'-character shows that the player's health is currently at its maximum level. When the player gets hurt, this counter will go down until it hits '0' at which point the player dies.

Tape 3 is the output tape. Every time a *new* symbol is printed on this tape, the game-engine looks up its meaning in the dictionary defined by the student and writes the result to the console. For example, when the game-engine looks up the 'a' currently on the tape, it will print the following to the console: "You are currently in your home-village. You can eat some (f)ood, or go (n)orth, (s)outh, (e)ast, or (w)est". This tells the player that he can type one of 'f', 'n', 'e', 's', or 'w' as an instruction.

Tape 4 is the decision tape. Before starting the TM program a pseudo-random number generator in the game-engine produces a series of pseudo-random numbers between 0 and 9 inclusive. This series of numbers is then inserted as Tape 4 in the

TM and one character is read from it, every time a non-deterministic state is reached in the game. The decision of which transition to take from the non-deterministic state is made based on this character.

Tape 5 is the input tape which initially consists of only question marks. Every time the TM encounters of these question marks on this tape, it will prompt the player for a character. After receiving a character – like ‘f’, ‘n’, ‘e’, ‘s’, or ‘w’ – the TM will put this character on *Tape 5* where the question mark used to be and proceed as normal.

In order to write a game using the framework described above, students need to (a) specify the desired behaviour of the TM and (b) define the dictionaries used by the game-engine. The latter of these students can do in a way which is relevant to them personally.

4.2.4 Concurrency: Value-Passing CCS

Although concurrency was not in the Top 3 of most important topics at undergraduate level, this chapter shows that the approach is also viable for modelling skills which are perhaps a little less basic than those previously mentioned, thus making the approach more versatile.

Concurrency is a property of systems which have more than one process running simultaneously. It is often taught using a process calculus. In these calculi, processes can communicate with each other by sending and receiving messages over designated channels. Figure 4.11 shows the simplified syntax for one these calculi called the value-passing calculus of concurrent systems (CCS). The two most interesting operators of the calculus are the first and last one and in particular, their interaction. Consider the following two processes:

$$x.A|\bar{x}.B$$

$$(x.A|\bar{x}.Bi)\setminus\{x\}$$

The first process can choose from three different actions: x , \bar{x} , or a synchronization of the two which is denoted as τ . This last one would leave us with $(A|B)$, because both the x and \bar{x} are executed simultaneously.

The second process has only one option available, i.e. τ , because the process is restricted by the channel x . This means that only synchronized x -actions are allowed to take place within the process.

Figure 4.12 shows an example of a relatively simple process defined in the value-passing CCS. The process consists of a number of processes running in parallel: $P_1 \dots P_n$

$E ::= E_1 \mid E_2$	<i>The process described by E_1 in parallel with the process described by E_2</i>
$E_1 + E_2$	<i>The process described by E_1 or the process described by E_2</i>
$\bar{a}(\vec{v}).E$	<i>Wait to receive a set of values over channel a and, once received, store them in the vector \vec{v}, followed by the process described by E</i>
$a(\vec{v}).E$	<i>Send the vector \vec{v} over channel a, followed by the process described by E</i>
$E \setminus L$	<i>The process E restricted by the set of channels L</i>

Figure 4.11: Simplified Syntax of the value-passing CCS

$$(P_1 | \dots | P_n | J) \setminus L$$

where

$$L = \{s_1, p_1\} \cup \dots \cup \{s_n, p_n\} \cup \{j, c\}$$

$$P_x = \bar{s}_x \langle i_x \rangle . (j \langle i_x, p_y, s_x \rangle . P_x + \bar{p}_x \langle r \rangle . r \langle i_x, s_x \rangle . P_x)$$

$$J = \bar{j} \langle i_x, p_y, s_x \rangle . p_y \langle c \rangle . \bar{c} \langle i_y, s_y \rangle . (s_x \langle i_x + 1 \rangle . s_y \langle i_y \rangle . J + s_x \langle i_x \rangle . s_y \langle i_y + 1 \rangle . J)$$

Figure 4.12: Specification of a Process in the value-passing CCS

and J . Each P_x -process first waits to receive a value over channel s_x , stores it in the variable i_x once received, and then does one of two things: it either (1) sends the three values i_x , p_y , and s_x over channel j after which it goes back to the beginning, or (2) it waits to receive a value over channel p_x , stores it in the variable r once received, then sends two values i_x and s_x over channel r before going back to the beginning. The J -process starts by waiting to receive three values over channel j and, once received, stores them in the variables i_x , p_y , and s_x . It continues by sending a value c over channel p_y and waits to receive two values over channel c which, once received, it will store in the variables i_y and s_y . Afterwards, it will do one of two things: (1) it sends out the value of i_x increased by one over channel s_x and then the value of i_y over channel s_y , or (2) it sends out the value of i_x over channel s_x and then the value of i_y increased by one over channel s_y .

Notice that the P_x -processes and the J -processes can communicate because they share certain channels. For example, P_x is sending over channel j , while J is receiving over that same channel.

Process descriptions like this illustrate very clearly a large part of the problem this thesis aims to address: abstract descriptions are often challenging to comprehend. Using a concrete example could greatly enhance a person's ability to understand processed like the one above. Fortunately, abstractness is not at all required to deliver the example. Consider the multi-player game definition in Figure 4.13 as a replacement of the elaborate variable web presented in Figure 4.12.

$$\begin{aligned}
& (\text{Player}_1 \mid \dots \mid \text{Player}_n \mid \text{Joust}) \setminus \text{Channels} \\
& \text{where} \\
\text{Channels} &= \{\overline{\text{receiveScore}}_1, \text{player}_1\} \cup \dots \\
& \quad \cup \{\overline{\text{receiveScore}}_n, \text{player}_n\} \cup \{\text{joust}, \text{receiveScore}\} \\
\text{Player}_x &= \overline{\text{receiveScore}}_x \langle \text{score}_x \rangle. \\
& \quad (\text{joust}(\text{score}_x, \text{player}_y, \overline{\text{receiveScore}}_x). \text{Player}_x + \\
& \quad \overline{\text{player}_x} \langle \text{sendScore} \rangle. \text{sendScore}(\text{score}_x, \overline{\text{receiveScore}}_x). \text{Player}_x) \\
\text{Joust} &= \overline{\text{joust}} \langle \text{score}_x, \text{player}_y, \text{sendScore}_x \rangle. \\
& \quad \text{player}_y(\overline{\text{receiveScore}}). \\
& \quad \overline{\text{receiveScore}} \langle \text{score}_y, \text{sendScore}_y \rangle. \\
& \quad (\text{sendScore}_x(\text{score}_x + 1). \text{sendScore}_y(\text{score}_y). \text{Joust} + \\
& \quad \text{sendScore}_x(\text{score}_x). \text{sendScore}_y(\text{score}_y + 1). \text{Joust})
\end{aligned}$$

Figure 4.13: Specification of a Game in the value-passing CCS

Note that the process in Figure 4.13 is exactly the same as that in Figure 4.12, only the names of the channels and variables have been changed. This process describes a simple online multi-player role-playing game. Each player has a score and can try to increase his score by initiating a joust against another player. This player will then be required to provide his current score and the players will joust. As a result of this joust, one player's score will be increased by one, while the other player's score remains the same.

This process description is not only a lot shorter, it is also a lot more transparent for two reasons: (1) the example is now concrete rather than abstract and (2) for at least some people, it presents the new knowledge through links with existing knowledge.

4.3 Practicalities

In this section, the practicalities that arise when using these mappings in the classroom are described and some solutions are proposed. The focus will be on practicalities that arise in every learning by game-building scenario, rather than specific instances.

4.3.1 Playing the Games

One important issue that has failed to come up so far is how the game-specification written by the student translates into a playable game. Section 4.2.3 does mention the 'game-engine' several times and herein lies the answer. When the students receive the game-building task, they receive two other things along with it: (1) a simple library for specifying their model in the appropriate framework and (2) a purpose-built game-engine. This game-engine simply takes the student's model – and possibly some additional information – and returns a playable game. An example of the extra information that may be required by the game-engine is the dictionary described in Section 4.2.3. The choice to keep the student's model and the game-engine strictly separate was a deliberate one. The game-building experience has to be as similar as possible to any other modelling task the student might be confronted with in the future – be it on an exam or in a prospective job – to prevent possibly interfering with transfer. No grand claims about transfer are made in this thesis, but an effort is made avoid impeding the process based on what little is known about it.

Although ideally, the model-building process should be as similar as possible to what it would be like in a non-game-building assignment, some adjustments are usually

desirable from the game-engine's point-of-view. For example, in an FSA-game, brief description for every state are required for the player of the game to know where they are in the game-world. In the second FSA in Figure 4.1 this would mean providing information like the following:

(State 1) The door is closed. The key is on the floor.

(State 2) The door is closed. You are holding the key.

(State 3) The door is open. You are holding the key.

(State 4) The door is open. The key is on the floor.

However, this information does *not* need to be provided for each individual state. Students only specify fragments of information for the most basic FSAs. This information then gets propagated to larger FSA which contain the basic FSAs. For example, the state information of each of the 4 states described above actually contains 2 fragments each: one describing the state of the door and one describing the state of the key. This is because the FSA is composed of two smaller FSAs: one modelling the behaviour of the door and one modelling the behaviour of the key. Chapter 6.3 describes this interaction between state information and constructing large FSAs in more detail.

In some ways requiring the students to provide this additional state-information forces them to think about the FSAs in a way which focusses on the states rather than the transitions, which could be beneficial for some students.

Similar augmentations are required in the other game-engines. Because these augmentations would not be present in a non-game-building modelling task, it is important that the student is made aware of the distinction between them and the basic modelling framework in order to prevent interfering with transfer.

4.3.2 Guiding the Building Process

A problem that arises from any educational setting in which the student is given a lot of freedom – and possibly the reason there is so little of it in assessed education – is the lack of control the educator can exercise over the student's learning. In fact, it can even be difficult to monitor a student's progress in such a context.

In the game-building tasks described in the current chapter, the student is given a lot of freedom, yet it also has to be ensured that the learning objectives are met, because in our study the approach is used in a real educational setting. Similarly, it

has to be possible to monitor a student's progress. So, to ensure that students explore the entire subject area while they are building their games, a list of requirements is provided which the game has to satisfy. For example in the game-building task on FSAs, we used a list which included the following requirements:

(Requirement 1) It is always possible to reach the goal.

(Requirement 2) Your game should contain at least one 'long' cycle (i.e. at least 5 states before a repeat).

(Requirement 3) Within at least one game-level it is always possible to reach the initial state.

(Requirement 4) Your game has at least one interleave.

As shown above, these requirements can be behavioural, structural, or otherwise. Together they specify all the skills on the curriculum that need to be covered in the task.

4.3.3 The Overhead Problem

There is also a question of overhead. Do the potential benefits of more control justify the amount of time students have to spend familiarizing themselves with the game-engine? Similarly, it is important to know if students will spend the majority of their time on the modelling task, and not trying to deal with the game-engine.

To investigate the nature of this balance, a pilot study was conducted in which students were given an FSA game-building task. There were a number of optional contact hours in which they were able to ask questions about the task. All these questions were recorded and revealed that 64% of questions during the contact hours were on the modelling task, 25% on computer-related issues which would occur in any computer-based task, and only 9% on game-engine related problems. These numbers are most likely due to the simplicity of the game-engine and the direct mapping of its function onto FSA combinators.

Although we are unable to give a conclusive answer to the question posed at the beginning of this section, the simplicity and direct mapping will at least minimize the amount of time that needs to be spent familiarizing oneself with the game-engine. Furthermore, the 9% ratio indicates that if there is an overhead, it is likely to be a small one. Also, a transparent API and a small example game could further reduce the overhead.

It should be noted that the overhead problem would be of an entirely different order if students without a basic programming skill were to attempt the game-building task. However, the focus of our research is on students who do possess such a skill.

4.3.4 Assessing the Resulting Games

Depending on the number of students attending a module, it may or may not be possible to assess each and every one of the students' games individually. When dealing with courses of 50+ students marking in this way would certainly put a large strain on teaching resources. It would also be difficult to mark the games fairly. However, the requirements used to ensure that students explore the entire subject area conveniently present us with a easy way to assess the individual student games. That is, a number of marks can be awarded for every requirement that is satisfied. In some cases, like the behavioural requirements of the FSA assignment, these can even be marked automatically by running simple checks on the game-FSAs. In other cases, it usually suffices to have a quick search through the student's code to check if the requirement has been fulfilled.

4.4 Summary

In this chapter we have seen four examples of how a game-building approach can be used to teach modelling skills in TCS. We also saw that it is the intuitive mapping between the modelling skill and the game-building skill that potentially might facilitate learning by game-building in the area of theoretical computer science. A deliberate choice was made to keep the game-engine separate from the model-building, to prevent interfering with transfer. For the same reason, any augmentations to the original modelling-framework were made explicit to the student. In fact, the sort of modelling task the student is asked to do when writing a game is very similar to any other modelling task they might be asked to do in another course on the subject. The difference lies in the fact that game-building provides the student with the option of personalizing their work.

In the next chapter the question whether or not there is a demand for own-game building will be put to the test. Additionally, an interest is taken in the motivational differences between participants who choose to write their own game, and those who choose to copy a pre-defined game.

In Chapter 6, the effect of learning by game-building on performance and enjoyment will be investigated. In particular, the differences in performance and enjoyment between a context in which participants are asked to write their own game and a context in which their are asked to copy a pre-defined game will be looked at.

Chapter 5

Study II: Motivation and Preference

5.1 Introduction

Previous chapters presented *learning-by-game-building* as a novel approach to theoretical computer science education. In particular, the opportunity to design an own game was identified as the feature most likely to have a positive influence on affect and performance. However, do students *want* to design an own game? If students lack any desire to participate in such a task and would rather copy a pre-defined game, all attempts to design an own-game task with the purpose of improving affect and performance could prove to be futile. So as an initial test of the viability of the proposed approach, this question needs to be addressed.

Furthermore, Lepper and Malone's research on motivation (Malone and Lepper, 1987; Lepper and Malone, 1987) described in Section 3.5 also raised some interesting questions the relationship between motivation and the presence of opportunities for publication and competition. A game-building approach lends itself to both, but is there any demand for it and how is this demand related to a student's motivational type?

Thus, the current chapter presents an investigation into the motivational type of our participants, their preferred degree of personalisation, and their preference for publication or competition in a game-building task. Using this data, the following three questions will be addressed: (1) is there a demand for the availability of personalisation in assessed assignments, (2) given the availability of personalisation, is there a demand for publication or competition, and (3) is there a correlation between a participant's motivational type and their preference for personalisation and/or publication or competition? By slight abuse of terminology, we define a 'demand' for own-game

building to be the following:

There is a demand for personalizable tasks among informatics undergraduates, if at least some of them choose these tasks over non-personalizable tasks when given a choice between the two. The size of the demand depends on the number of students preferring personalizable tasks over non-personalizable tasks.

Because our interest lies with game-building, the concept of personalisation used to address Questions 1, 2, and 3 is defined in terms of the amount of freedom students are allowed on a game-building task. That is, if personalisation is available, they would be asked to design their own game; in a scenario in which personalisation is not available, they would be asked to copy a pre-defined game. Publication and competition are also defined within a game-building context, the former referring to students sharing their game with their peers and the latter to students entering their game into a peer-reviewed competition. Finally, the General Academic Achievement Goal Questionnaire (GAAGQ) as proposed by Finney *et al.* (2004) was used as a measure of motivational types for the reasons described in Chapter 3.

5.1.1 Design

5.1.1.1 Empirical Design

To address the first two questions, a study was designed in which participants were given a description of two game-building tasks. In the first task, they were asked to design their own game; in the second, they were asked to copy a pre-defined game. Participants could choose which of the two tasks they preferred to fulfil. Those participants who chose to design their own game had two more options at their availability: (a) share their game with their peers or (b) enter their game into the peer-reviewed competition. The latter entails the former, since peers need to be able to access the game in order to review it. Of course participants could also choose neither publication or competition. These two choices give rise to four different experimental conditions: (1) building a pre-defined game, (2) building an own game only, (3) building an own game and publishing it, or (4) building an own game, publishing it, and entering it into a competition.

To address the third question, participants were given a GAAGQ to fill out in order to determine their motivational type.



Figure 5.1: Example Java Game.

5.1.1.2 The Game-Building Task

For both game-building tasks, the own-game and pre-defined game, participants were asked to write a simple computer game similar to the one shown in Figure 5.1. The games are written using a Java game engine which was designed specifically for this study by DeLesley Hutchins and the author. The game engine was implemented by DeLesley Hutchins, who was the teaching assistant for the course in which the study was conducted.

The student games written using the game engine were to satisfy a short list of requirements like (1) your game should include an object which modifies the state of the player, (2) your game should contain an object which creates a new entity, or (3) your game should contain an object which depends on the location of the player. The full sets of requirements used in the assignments can be found in Appendices C and D. These requirements ensure that participants explore the full subject domain. Additionally, to determine how well a participant has done on the game-building task, their game can be assessed according to the number of requirements it satisfies. See Section 4.3.4 for further explanation of assessing games.

Because it is non-trivial to detect satisfaction of requirements like these using an automated marking script, participants were asked to record a demo of their game in which they show how their game satisfies each item on the list of requirements. The demos are recorded using a special-purpose recorder which is part of the game engine used to implement the games. Figures 5.2 and 5.3 show two screen-shots of such a demo being recorded and an explanation follows.

In Figure 5.2 there are three open windows visible. From back to front they are: (1) the game, (2) the demo-recorder, and (3) a window containing a request from the demo-recorder to specify how the participant plans to satisfy requirement 6. This last window would have appeared after pressing the button *'Begin R6'* in the demo-recorder. When the participant has finished describing how he intends to satisfy requirement 6, the *'OK'*-button is pressed and the participant can continue to show the satisfaction of requirement 6 in the game-window. Once he has accomplished this, all that remains is for him to press the button *'End R6'*. This button is not visible in Figure 5.2, because its space is currently being occupied by the button *'Begin R6'*. The change of button labels happens immediately after the *'OK'*-button is pressed in the front window in Figure 5.2. When the button *'End R6'* is pressed, the label changes again, this time to *'Done R6'* which is shown in Figure 5.3. This second screen shot also shows that the demo-recorder window now contains the full description of the requirement satisfaction as entered by the participant and as promised a 'red' pill, a capital 'P', lying right next to Morpheus who is portrayed in the game as capital 'M'.

The Two Tasks The fact that there are two game-building tasks rather than a single one introduces an issue with both an ethical and a methodological side: how do the two tasks compare? From a methodological point of view, having the two tasks be identical except for the amount of personalisation available is of vital importance. Participants should make their choice for one or the other condition based only on the difference in the amount of personalisation available, without any other factors being of influence. From an ethical point of view, it is imperative that both game-building tasks incorporate the same learning objectives, since we do not want one group of participants to be (dis)advantaged compared to the other.

The set of requirements, which was introduced to ensure participants explore the full subject domain and to assess the games they produce, offers a solution to both the methodological and the ethical side of this issue. By using identical lists of requirements for the own-game and pre-defined game, and by having the pre-defined game

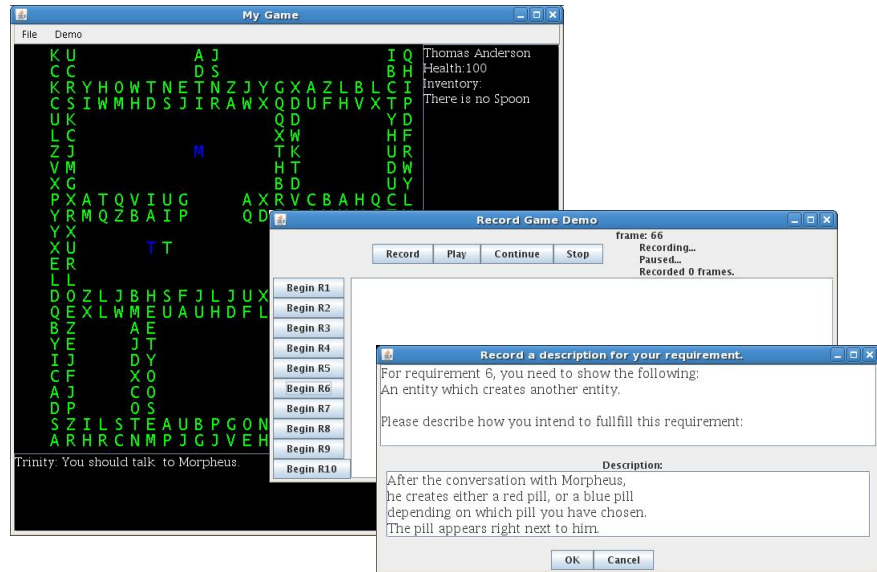


Figure 5.2: Recording a demo (screen shot 1 of 2).

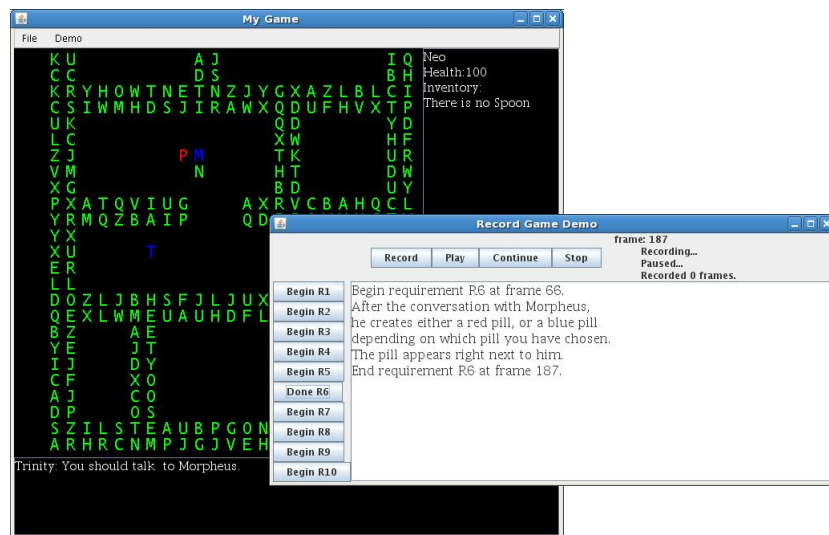


Figure 5.3: Recording a demo (screen shot 2 of 2).

satisfy those requirements exactly, the two game-building tasks incorporate the same learning objectives and differ only in the amount of personalisation available.

Of course the pre-defined game may still be conceived by some participants as the 'easy option', because it does not involve any design work in addition to the programming work. However, other students may find that they like the design aspect of the own game task and because of this, they may even find that they can write an own game with less effort than the pre-defined game. Therefore, the design aspect of an own-game task has the potential to influence the participants' decisions in either direction.

Finally, some participants may even consider the own-game task to be the 'easy option' from the start, because the pre-defined game is in fact not an absolute *minimal* implementation of the set of requirements and it is possible to write a smaller game which still satisfies all of the requirements.

Furthermore, both tasks required the student to record a demo as described in the previous section. Recording a demo could help some students in their understanding of the material, as it requires reflection on a higher level.

5.1.1.3 Measuring Motivation

Because our study was conducted in the context of an assessed assignment for an introductory programming course, the available options for measuring motivation of all participants was limited. In particular, the study required a measure in the form of a short, self-explanatory questionnaire which could be completed in a matter of minutes. Because several well-established questionnaires exist for measuring motivation, it was considered more appropriate to use one of these, rather than devise a new measure. The final requirement for the motivation measure was that it had to be discriminatory in the areas where a difference in choice for and/or response to the own-game game-building task was expected. The General Academic Achievement Goal Questionnaire (GAAGQ as described in Finney *et al.* (2004)) was chosen because it satisfied all criteria. First of all, it comprised twelve short questions answered on a 7-point Likert scale. Second, the literature on the kinds of motivation it identifies is extensive, as is the literature on the questionnaire itself. Third, the questionnaire has been tested for reliability and validity. Finally, the original GAAGQ identifies four different kinds of motivation or *achievement goals*, which appear to suggest there are students that will do well in an own-game scenario, and other that will not. The four kinds of achievement goals measured by the GAAGQ are: (a) *mastery-approach*, (b) *mastery-avoidance*, (c)

	<i>Absolute / Intrapersonal</i> (mastery)	<i>Normative / Interpersonal</i> (performance)	Work Related
<i>Positive (approaching success)</i>	Mastery Approach Goal	Performance Approach Goal	n/a
<i>Negative (avoiding failure)</i>	Mastery Avoidance Goal	Performance Avoidance Goal	Work Avoidance Goal

Figure 5.4: Dimensions in the Original and Extended Achievement Goal Framework.

performance-approach, and (d) *performance-avoidance*. Figure 5.4 shows a graphical representation of these four kinds of achievement goals in a 2-dimensional array.

A person scoring high on mastery approach is characterised by trying to increase their competence in their own eyes, whereas a high score on performance approach indicates a drive to increase their competence in the eyes of others. Similarly, a person scoring high on mastery avoidance is characterised by trying to avoid feeling incompetent, in their own eyes, whereas a high score on performance avoidance indicates a drive to avoid looking incompetent in the eyes of others. Chapter 3 contains further discussion of motivation measures including the GAAGQ.

As described in the previous paragraph, the original GAAGQ comprises twelve questions. Within these twelve, there are three questions on each of the four different achievement goals. All questions are answered on a 7-point Likert scale. The GAAGQ yields four scores, one for each of the four different achievement goals, which indicate how strong a participant's affinity is with that particular kind of achievement goal. In the first of the two data collection opportunities of the study described in this chapter, the 4-way GAAGQ, as described above, was used.

An extension to the original GAAGQ was proposed in Pieper (2003) in which four questions were added on the achievement goal of *work-avoidance*. Figure 5.4 shows the position of this new achievement goal in the 2-dimensional array. The extended GAAGQ thus yields five scores: the original four, plus (e) work-avoidance. A person scoring high on work-avoidance can be described as generally trying to do the minimum required amount of work to achieve their goals. Because of its potential to affect

a participant's choice of task, it was decided to use the extended GAAGQ during the second data collection opportunity of the study described in this chapter. Figure 5.5 shows both the original and extended GAAGQ: Questions 1–12 and 1–16 respectively. Questions 1–3 measure mastery approach; Questions 4–6 measure performance approach; Questions 7–9 measure performance avoidance; Questions 10–12 measure mastery avoidance; and Questions 13–16 measure work avoidance. The questions were presented in the order suggested by Pieper (2003): 4, 7, 1, 13, 10, 5, 2, 8, 14, 3, 11, 6, 15, 12, 9, 16.

5.1.2 Hypotheses

In the next section a study is described which was designed to test the following three hypotheses:

1. When presented with two game-building tasks, an own-game and a pre-defined-game one, some participants will choose to write their own game and some the pre-defined game.

As discussed in Chapter 3, different kinds of people thrive in different environments (e.g. Squire, 2004; Stenning, 2002; Lepper and Malone, 1987). Therefore, we expect some people to prefer an environment with a lot of choice, and others to prefer a more traditional environment, with very little choice.

2. When presented with the options of entering an own-game into a competition and making it available to their peers, some participants will select these, while others will not.

Similarly, we expect some people to enjoy a competitive environment, while others are less keen on competition. Malone and Lepper (1987) discuss the issue of competition in the context of game-based learning.

3. There is a relationship between a participant's GAAGQ-results and the options they choose.

Based on attributes associated with each of the five different achievement goals, as described in Figure 3.3, we expect there to be a relationship between a participant's GAAGQ-results and the task and competition/publication options they choose. In particular, we expect the following correlations to hold:

1. Completely mastering the material in my course is important to me this semester.
2. I want to learn as much as possible this semester
3. The most important thing this semester is to understand the content in my courses as thoroughly as possible.
4. My goal this semester is to get better grades than most of the other students.
5. It is important for me to do well compared to other students this semester.
6. I want to do better than other students this semester.
7. I just want to avoid doing poorly compared to other students this semester.
8. The fear of performing poorly is what motivates me.
9. My goal this semester is to avoid performing poorly compared to other students.
10. I'm afraid that I may not understand the content of my courses as thoroughly as I'd like.
11. I worry that I may not learn all that I possibly could this semester.
12. I am definitely concerned that I may not learn all that I can this semester.
13. I really don't want to work hard in my classes this semester.
14. I want to do as little work as possible this semester.
15. I want to get through my courses by doing the least amount of work possible.
16. I look forward to working really hard this semester in my coursework. **(Reverse scored)**

Figure 5.5: Original and Extended GAAGQ.

3a. Participants who choose to write their own game score

- *higher* on mastery approach and performance approach,
- but *lower* on mastery avoidance, performance avoidance, and work avoidance

compared to those who choose to write a pre-defined game.

These expectations are based on the fact that both **mastery approach** and **performance approach** are associated with an *overall need for achievement* (see Figure 3.3). Because an own game task offers more opportunities for achievement than the pre-defined task, we expect own-game choosers to score highly on these *approach* goals.

Furthermore, **mastery avoidance**, **performance avoidance**, and **work avoidance** are all associated with *fear of failure* (see Figure 3.3). Because the own-game task does not only offer more opportunities for achievement, but also more opportunities for failure, we expect own-game choosers to score lower on the *avoidance* goals than their pre-defined game choosing peers.

3b. Participants who choose to enter the competition score

- *higher* on performance approach
- but *lower* on mastery avoidance, performance avoidance, and work avoidance

compared to those who write their own game but do not enter the competition.

These expectation were slightly more difficult to formulate, as the attributes associated with the five achievement goals give contradictory signals. First of all, according to Figure 3.3, there are no attributes related to competition associated with **mastery approach**; as a result, no hypothesis is specified with regards to this achievement goal. **Performance approach** on the other hand is associated with *competitiveness*. Therefore, we expect participants who enter the competition to score higher on this achievement goal than those who did not enter the competition.

Performance avoidance and *work avoidance* are also associated with *competitiveness*. However, because they are also associated with *fear of failure* (both), *lack of determination* (performance avoidance), and *preference for easy tasks* (work avoidance), we do not expect participants who enter the competition to score higher on these

two avoidance goals. In fact, we expect the effect of these associated attributes to outweigh the effect of *competitiveness*. Therefore, participants who choose to enter the competition should score lower on **performance avoidance** and *work avoidance* than those who did not enter the competition.

Finally, Figure 3.3 shows that **mastery avoidance** is associated with *fear of failure* and *lack of self-determination*. This causes us to hypothesize that participants who enter the competition score lower on this achievement goal than those who did not enter the competition.

5.2 Method

5.2.1 Design

The study was conducted as a between-group, or independent, design. The groups were self-selecting, i.e. each participant chose the task they wished to perform, which put them into one of the four experimental conditions. Participants could choose from the following four tasks: (1) build a pre-defined game, (2) build an own game, (3) build an own game and publish it, or (4) build an own game, publish it, and enter it into the competition.

5.2.2 Participants

Two-hundred-and-twelve undergraduates participated in this study. All participants were taking a first-year level Informatics course in *Object-Oriented Programming & Data and Analysis* at the University of Edinburgh during the academic years 2005–2006 and 2006–2007. They ranged in age from 17 to 40 with a mean age of 20. Of the 212 participants, 88% were male and 12% female. Furthermore, 82% of participants were required to take the course as part of their degree and 18% were not. See Figure 5.1 for a detailed breakdown.

5.2.3 Materials

The materials used in this study consisted of four parts:

Game Engine A purpose-built game engine¹ for the programming language Java allowed for games like the one in Figure 5.1 to be built.

¹Designed and written by DeLesley Hutchins.

Table 5.1: Breakdown of Gender by Degree.

	Required for Degree		Total
	<i>Yes</i>	<i>No</i>	
Gender <i>Female</i>	21	5	26
<i>Male</i>	153	33	186
Total	174	38	212

API of Game Engine The API of the game engine provided detailed documentation of all the features available in the game-engine.

Hand-out A 10-page hand-out described the game-building task participants were asked to perform. It contained instructions for all four experimental conditions. Besides task instructions, the hand-out also included the following:

- the list of *requirements* the games had to satisfy
- a description of the most important features of the game engine, including a manual for the demo-recorder
- a General Academic version of the Achievement Goals Questionnaire (GAAGQ)

In 2005–2006 the GAAGQ comprised twelve questions, three on each of the following achievement goals: (a) mastery-approach, (b) mastery-avoidance, (c) performance-approach, and (d) performance-avoidance, as proposed by Finney *et al.* (2004). In 2006–2007 four questions were added to the original twelve to collect data on a fifth achievement goal: (e) work-avoidance. The resulting variation of the GAAGQ is described in Pieper (2003). See Appendices C and D for the hand-outs of the academic years 2005–2006 and 2006–2007 respectively.

Playable Demo A playable demo of the pre-defined game was also available.

5.2.4 Procedure

The hand-out, together with the rest of the materials, was presented in the form of an assessed assignment. Throughout the rest of this thesis, this assignment will be referred to as Assignment 1. Participants were given two weeks to complete the assignment. During those two weeks, several resources were available to participants who needed help with the game-building task: (a) a newsgroup answered by course staff and (b)

Table 5.2: Comparison of Choice of Game by Choice of Competition/Publication.

		Public Choice			Total
		<i>Publ.</i>	<i>Comp.</i>	<i>Neither</i>	
Game Choice	<i>Pre-Defined Game</i>	n/a	n/a	92	92
	<i>Own Game</i>	3	50	67	120
Total		3	50	159	212

five 2-hour drop-in lab-sessions per week where questions are answered by the course staff in person. At the end of the two weeks participants handed in two files: their game and a plain-text file containing their answers to the GAAGQ and their choice of experimental condition. The submitted GAAGQ counted as 5 marks out of the 100 total for the assignment.

5.3 Results

5.3.1 Experimental Condition Choice

Of the 212 participants, 43% chose to write the pre-defined game and 57% chose to write their own game. Of the 57% who chose to write their own game, 42% chose to enter the peer-reviewed competition, which includes publication, 2% chose to only publish their game, and 56% chose not to make their game available to their peers at all. Figures 5.2 and 5.6 illustrate these findings.

Figures 5.7 and 5.8 show the same data, but split into the academic years in which the data was collected. In 2005–2006, 44% of the participants chose to write the pre-defined game and 56% chose to write their own game. Of the 56% who chose to write their own game, 50% chose to enter the peer-reviewed competition, which includes publication, 3% chose to only publish their game, and 47% chose not to make their game available to their peers at all. In 2006–2007, 43% of the participants chose to write the pre-defined game and 57% chose to write their own game. Of the 57% who chose to write their own game, 31% chose to enter the peer-reviewed competition, which includes publication, 2% chose to only publish their game, and 67% chose not to make their game available to their peers at all.

Pearson's Chi-Square test shows that there is no significant difference between the two years in terms of choice, $\chi^2(2) = 4.25, p > 0.10$. These results were obtained after

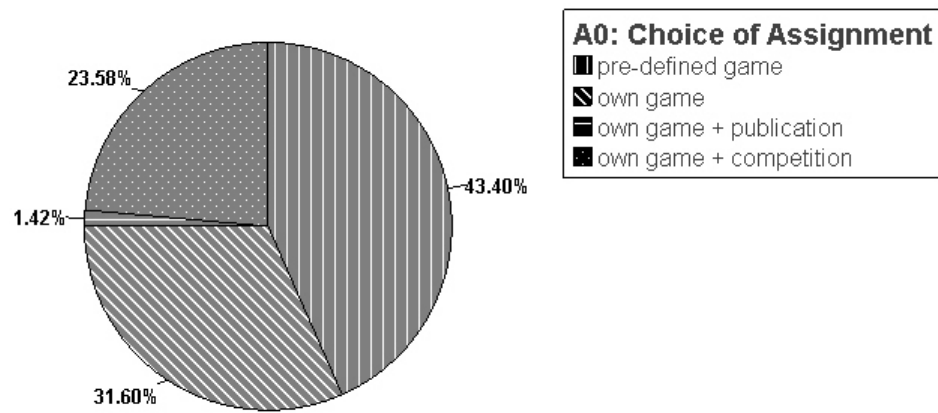


Figure 5.6: Choice distribution (based on 212 data-points).

Year	Public Choice			Total		
	<i>Publ.</i>	<i>Comp.</i>	<i>Neither</i>			
'05-'06	Game Choice	<i>Pre-Defined Game</i>	n/a	n/a	52	52
		<i>Own Game</i>	2	33	31	66
	Total		2	33	83	118
'06-'07	Game Choice	<i>Pre-Defined Game</i>	n/a	n/a	40	40
		<i>Own Game</i>	1	17	36	54
	Total		1	17	76	94

Figure 5.7: Comparison of Choice of Game by Choice of Competition/Publication, split by the Year the Data was Collected.

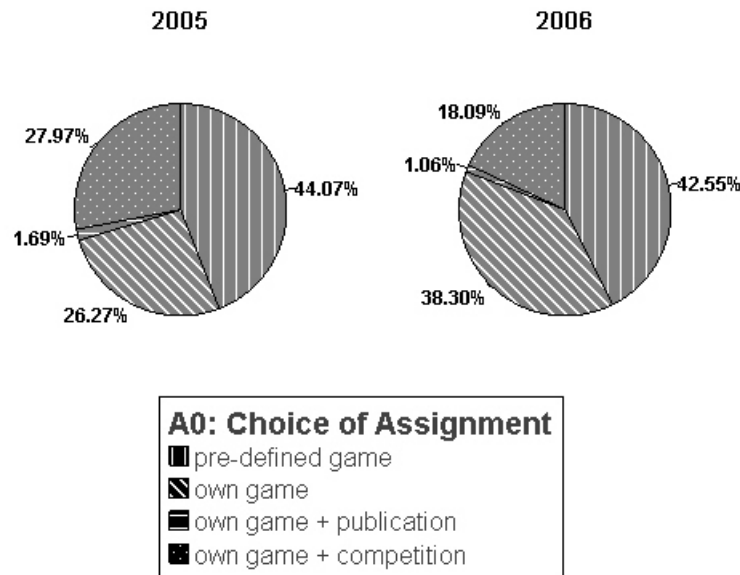


Figure 5.8: Choice distribution divided by year (based on 118 and 94 data-points respectively).

collapsing the own-game and own-game-plus-publication categories, because the latter would yield an estimated frequency of less than five, thus violating one of the assumptions of Pearson's Chi-Square test. A choice was made to collapse the category with the own-game category, rather than own-game-plus-competition category, because we are interested in the motivational effect of competition entry.

5.3.2 Motivation (Achievement Goals)

Figures 5.9 and 5.10 show the raw results from the GAAGQ. Scores on mastery-approach, performance-approach, mastery-avoidance, and performance-avoidance ranged from 3 to 21, while scores work-avoidance ranged from 4 to 28. Work-avoidance data is not available for the year 2005–2006.

Both figures show small but significant positive correlations between mastery approach and performance approach, and between mastery avoidance and performance avoidance. In Figure 5.9 additional small but significant correlation are found between performance approach and performance avoidance, and between performance approach and mastery avoidance, although the latter is of a lower significance. Furthermore, the 2006-2007 data in Figure 5.10 displays a small but significant positive cor-

Descriptive Statistics				
Abbr.		Median	Mean	Std. Dev.
MAp	<i>Mastery-Approach</i>	16.00	15.92	3.596
PAP	<i>Performance-Approach</i>	13.00	12.42	4.632
PAv	<i>Performance Avoidance</i>	12.00	11.61	4.319
MAv	<i>Mastery Avoidance</i>	12.00	12.29	4.558

Correlations: Kendall's τ				
	MAp	PAP	PAv	MAv
MAp	1	0.210**	-0.053	0.019
PAP	-	1	0.127**	-0.120*
PAv	-	-	1	0.262**
MAv	-	-	-	1

** Correlations significant at the 0.01-level.

* Correlations significant at the 0.05-level.

Figure 5.9: GAAGQ Descriptives and Correlations (based on 212 data-points).

relation between performance approach and performance avoidance, a medium-sized significant negative correlation between mastery approach and work avoidance, and a very small and barely significant positive relationship between performance avoidance and work avoidance.

None of the achievement goal data were found to be parametric, so non-parametric tests were carried out instead. The Mann-Whitney test shows that there is no significant difference between the two years in mastery avoidance, $U = 5439.50, p > 0.80$, or performance approach, $U = 5389.50, p > 0.70$. However, the 2006 participants did score significantly higher on mastery approach ($Mdn = 17.00$) than their 2005 counterparts ($Mdn = 16.00$), $U = 4645.50, p < 0.05, r = -0.14$. In turn, the 2005 participants scored significantly higher on performance avoidance ($Mdn = 13.00$) than their 2006 counterparts ($Mdn = 11.00$), $U = 4614.00, p < 0.05, r = -0.14$.

5.3.3 Achievement Goals and Choice

Because none of the achievement goal data are parametric, the nature of the relationship between achievement goals, as identified by the GAAGQ, and the participants'

Descriptive Statistics

Year	Abbr.		Median	Mean	Std. Dev.
'05-'06	MAv	<i>Mastery-Approach</i>	16.00	15.56	3.606
	PAp	<i>Performance-Approach</i>	13.00	12.55	4.324
	PAv	<i>Performance Avoidance</i>	13.00	12.20	4.025
	MAv	<i>Mastery Avoidance</i>	12.00	12.19	4.239
	WAv	<i>Work Avoidance</i>	n/a	n/a	n/a
'06-'07	MAv	<i>Mastery-Approach</i>	17.00	16.38	3.550
	PAp	<i>Performance-Approach</i>	13.00	12.26	5.012
	PAv	<i>Performance Avoidance</i>	11.00	10.87	4.577
	MAv	<i>Mastery Avoidance</i>	12.00	12.40	4.950
	WAv	<i>Work Avoidance</i>	10.00	11.02	4.727

Correlations: Kendall's τ

Year		MAp	PAP	PAv	MAv	WAv
'05-'06	MAp	1	0.220**	-0.062	0.005	n/a
	PAP	-	1	0.040	-0.117	n/a
	PAv	-	-	1	0.263**	n/a
	MAv	-	-	-	1	n/a
	WAv	n/a	n/a	n/a	n/a	n/a
'06-'07	MAp	1	0.192*	-0.035	0.023	-0.366**
	PAP	-	1	0.214**	-0.118	0.023
	PAv	-	-	1	0.255**	0.177*
	MAv	-	-	-	1	0.000
	WAv	-	-	-	-	1

** Correlations significant at the 0.01-level.

* Correlations significant at the 0.05-level.

Figure 5.10: GAAGQ Descriptives and Correlations by Year (based on 118 and 94 data-points respectively).

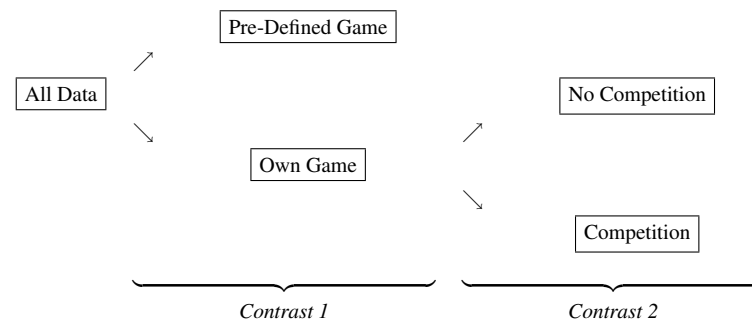


Figure 5.11: Planned Contrasts.

choice of experimental condition was determined using a *Kruskal-Wallis test* followed by planned comparisons using one-tailed Mann-Whitney tests. The results of the Mann-Whitney tests were corrected using a Bonferroni correction.

The participants' choice of experimental condition was used as the independent variable. As in Section 5.3.1, the own-game-plus-publication category was collapsed with the own-game category because the former does not comprise sufficient cases. After the collapse, three categories or experimental conditions remain: (1) pre-defined game, (2) own-game-no-competition, and (3) own-game-plus-competition.

For the dependent variables, the five dimensions of the GAAGQ – mastery-approach, performance-approach, mastery-avoidance, performance-avoidance, and work-avoidance – were used. However, because the work-avoidance data is missing for the year 2005–2006, two separate analyses were conducted: one on the complete set of data, which comprises 212 data-points, and one divided by year to create an opportunity to discover any potential relationships involving work-avoidance, as well those involving the other four dimensions.

Following the main Kruskal-Wallis analysis, two planned contrasts were carried out. The first contrast compared the pre-defined game condition with both own-game conditions (i.e. competition and no-competition); the second contrast compared the own-game-no-competition with the own-game-plus-competition condition. Throughout the rest of this section these contrast will be referred to as Contrast 1 and Contrast 2. Figure 5.11 shows a graphical representation of the planned contrasts. A Bonferroni correction was applied, so all contrast results reported are at the 2.5% significance level.

Ranks			
	Condition	N	Mean Rank
<i>Mastery Approach</i>	pre	92	107.50
	own-no-comp	70	97.26
	own-plus-comp	50	117.60
<i>Performance Approach</i>	pre	92	106.51
	own-no-comp	70	96.73
	own-plus-comp	50	120.16
<i>Performance Avoidance</i>	pre	92	114.19
	own-no-comp	70	107.97
	own-plus-comp	50	90.29
<i>Mastery Avoidance</i>	pre	92	115.66
	own-no-comp	70	109.73
	own-plus-comp	50	85.12

Figure 5.12: Achievement Goal Ranks by Condition (based on 212 data-points).

5.3.3.1 All Data (excluding Work Avoidance)

Using the statistical analysis described above on the complete set of data, the only significant overall effect that was found, was that of *mastery avoidance* on choice, $\chi^2(2) = 8.361, p < 0.05$. See Figure 5.12 for a table of the ranks used in the analysis.

Contrast 1 revealed a near-significant tendency for pre-defined game choosers to score higher on *mastery avoidance* ($Mdn = 14.00$) than their own-game-choosing peers, ($Mdn = 12.00$), $U = 4677.00, p = 0.028, r = -0.13$.

Contrast 2 revealed that own-game choosers who did not enter the competition scored lower on *performance approach* ($Mdn = 11.50$) than their competition-entering peers, ($Mdn = 14.00$), $U = 1366.00, p < 0.025, r = -0.14$. Contrast 2 also revealed that own-game choosers who did not enter the competition scored higher on *mastery avoidance* ($Mdn = 13.00$) than their competition-entering peers, ($Mdn = 10.50$), $U = 1347.00, p < 0.025, r = -0.15$. See Figures 5.13 and 5.14 for the ranks used in both contrasts.

Contrast 1: Ranks

	Condition	N	Mean Rank
<i>Mastery Approach</i>	pre	92	107.50
	own	120	105.73
<i>Performance Approach</i>	pre	92	106.51
	own	120	106.49
<i>Performance Avoidance</i>	pre	92	114.19
	own	120	100.60
<i>Mastery Avoidance</i>	pre	92	115.66
	own	120	99.48

Figure 5.13: Contrast 1: Achievement Goal Ranks by Condition (based on 212 data-points).

Contrast 2: Ranks

	Condition	N	Mean Rank
<i>Mastery Approach</i>	own-no-comp	70	55.68
	own-plus-comp	50	67.25
<i>Performance Approach</i>	own-no-comp	70	55.01
	own-plus-comp	50	68.18
<i>Performance Avoidance</i>	own-no-comp	70	64.89
	own-plus-comp	50	54.35
<i>Mastery Avoidance</i>	own-no-comp	70	66.26
	own-plus-comp	50	52.44

Figure 5.14: Contrast 2: Achievement Goal Ranks by Condition (based on 120 data-points).

		Ranks		
Year	Condition	N	Mean Rank	
'05—'06	<i>Mastery Approach</i>	pre	52	54.57
		own-no-comp	33	62.23
		own-plus-comp	33	64.55
	<i>Performance Approach</i>	pre	52	58.40
		own-no-comp	33	58.23
		own-plus-comp	33	62.50
	<i>Performance Avoidance</i>	pre	52	66.08
		own-no-comp	33	57.88
		own-plus-comp	33	50.76
<i>Mastery Avoidance</i>	pre	52	57.59	
	own-no-comp	33	69.23	
	own-plus-comp	33	52.79	
'06—'07	<i>Mastery Approach</i>	pre	40	53.41
		own-no-comp	37	37.51
		own-plus-comp	17	55.32
	<i>Performance Approach</i>	pre	40	48.30
		own-no-comp	37	40.58
		own-plus-comp	17	60.68
	<i>Performance Avoidance</i>	pre	40	48.64
		own-no-comp	37	50.74
		own-plus-comp	17	37.76
	<i>Mastery Avoidance</i>	pre	40	57.96
		own-no-comp	37	44.08
		own-plus-comp	17	30.97
	<i>Work Avoidance</i>	pre	40	43.44
		own-no-comp	37	57.31
		own-plus-comp	17	35.71

Figure 5.15: Achievement Goal Ranks by Condition by Year (based on 118 and 94 data-points respectively).

Contrast 1: Ranks				
Year		Condition	N	Mean Rank
'05—'06	<i>Mastery Approach</i>	pre	52	54.57
		own	66	63.39
	<i>Performance Approach</i>	pre	52	58.40
		own	66	60.36
	<i>Performance Avoidance</i>	pre	52	66.08
		own	66	54.32
	<i>Mastery Avoidance</i>	pre	52	57.59
		own	66	61.01
'06—'07	<i>Mastery Approach</i>	pre	40	53.41
		own	54	43.12
	<i>Performance Approach</i>	pre	40	48.30
		own	54	46.91
	<i>Performance Avoidance</i>	pre	40	48.64
		own	54	46.66
	<i>Mastery Avoidance</i>	pre	40	57.96
		own	54	39.95
	<i>Work Avoidance</i>	pre	40	43.44
		own	54	50.51

Figure 5.16: Contrast 1: Achievement Goal Ranks by Condition by Year (based on 118 and 94 data-points respectively).

5.3.3.2 By Year (including Work Avoidance)

Using the same analysis, but with the data divided by the year in which it was collected, no significant overall effect was found for 2005–2006. The data collected in 2006–2007 on the other hand showed significant effects for *mastery approach* ($\chi^2(2) = 8.328, p < 0.05$), *performance approach* ($\chi^2(2) = 6.412, p < 0.05$), *mastery avoidance* ($\chi^2(2) = 12.453, p < 0.01$), and *work avoidance* ($\chi^2(2) = 8.907, p < 0.05$). See Figure 5.15 for a table of the ranks used in the analysis.

Contrast 1 revealed that the 2006–2007 pre-defined-game-choosers score higher on *mastery avoidance* ($Mdn = 15.00$) than their own-game-choosing peers ($Mdn = 10.00$), $U = 672.50, p < 0.01, r = -0.22$.

Contrast 2: Ranks				
Year		Condition	N	Mean Rank
'05—'06	<i>Mastery Approach</i>	own-no-comp	33	32.68
		own-plus-comp	33	34.32
	<i>Performance Approach</i>	own-no-comp	33	32.27
		own-plus-comp	33	34.73
	<i>Performance Avoidance</i>	own-no-comp	33	35.91
		own-plus-comp	33	31.09
	<i>Mastery Avoidance</i>	own-no-comp	33	38.17
		own-plus-comp	33	28.83
'06—'07	<i>Mastery Approach</i>	own-no-comp	37	24.07
		own-plus-comp	17	34.97
	<i>Performance Approach</i>	own-no-comp	37	23.74
		own-plus-comp	17	35.68
	<i>Performance Avoidance</i>	own-no-comp	37	29.93
		own-plus-comp	17	22.21
	<i>Mastery Avoidance</i>	own-no-comp	37	29.96
		own-plus-comp	17	22.15
	<i>Work Avoidance</i>	own-no-comp	37	31.82
		own-plus-comp	17	18.09

Figure 5.17: Contrast 2: Achievement Goal Ranks by Condition by Year (based on 66 and 54 data-points respectively).

Contrast 2 revealed that the 2005-2006 own-game choosers who did not enter the competition scored higher on *mastery avoidance* ($Mdn = 14.00$) than their competition-entering peers ($Mdn = 12.00$), $U = 390.50$, $p < 0.025$, $r = -0.24$.

For the 2006–2007 data, Contrast 2 revealed significant differences in *mastery approach* (MAp), *performance approach* (PAp), and *work avoidance* (WAv) between own-game choosers who did not enter the competition and those who did. The own-game-choosers who did not enter the competition scored higher on *work avoidance* ($Mdn_{WAv} = 12.00$) than their competition choosing peers ($Mdn_{WAv} = 9.00$), $U_{WAv} = 154.50$, $p < 0.01$, $r = -0.41$. However, they scored lower on both *mastery approach* and *performance approach* ($Mdn_{MAp} = 16.00$, $Mdn_{PAp} = 11.00$), than the competition choosers ($Mdn_{MAp} = 18.00$, $Mdn_{PAp} = 15.00$), $U_{MAp} = 187.50$, $p < 0.01$, $r = -0.32$; $U_{PAp} = 175.50$, $p < 0.01$, $r = -0.35$. See Figures 5.16 and 5.17 for the ranks used in both contrasts.

5.4 Discussion

While some of the results of the study validate the hypotheses presented in Section 5.1.2, others raise more questions.

5.4.1 Choice

As expected, all four experimental conditions were populated, although not all of them in equal numbers and not all of them consistently over the two years in which we collected our data. The two groups *were* approximately evenly split (43% vs. 57%) and remained consistent across years (+/- 1%) are the own-game and pre-defined-game groups.

The interest in competition was less consistent. It varied from 31% entering the competition to 50%. Although this difference was not significant and therefore likely due to chance, two alternative explanations for this finding can be speculated upon. First of all, this chapter tried to establish a connection between a participant's motivational type and their choice of experimental condition. If the motivational make-up of the two years differ, it could explain the discrepancy between year in terms of competition choice. We will revisit this hypothesis when we discuss the results of the GAAGQ. Secondly, competition is a social phenomenon; if all your friends enter, it is likely you will too. On the other hand, if none of your friends enter, you may decide not to bother

either. This could be particularly true of people scoring high on performance goals, since they care most about the opinion of their peers. So, the fact that competition is a social phenomenon could explain why some years there will be more demand for competition than others.

Very few people chose the publication-only option and this was consistent across years. The reason for this is not entirely obvious, but we expect it might be linked to the make-up of the group of participants we chose, e.g. informatics students. However, one other possible explanation could be that no additional effort was required to enter the competition compared to publication-only. If a participant has decided to publish their game they could think that they might as well enter the competition because they would gain the possibility of winning the competition, and lose nothing compared to the publication-only option. Alternatively, some participants may not have even considered the publication-only option, because it was included in the competition option. They may have considered only the competition versus no competition option.

5.4.2 Achievement Goals

The raw achievement goal results do not throw up many surprises. As expected the two approach goals are consistently positively correlated and so are the two original avoidance goals. Perhaps surprisingly, the two performance goals do not seem to correlate as strongly and the two mastery goals do not seem to correlate at all. However, our main interest does not lie with the internal properties of the achievement goals, but with their ability to predict a participants' choice of experimental condition.

In the discussion of the choice results, we predicted that the discrepancy in competition choice between the two years could possibly be explained in terms of the motivational make-up of the two years. However, our data do not seem to support this suggested explanation. In fact, they seem to suggest the opposite. While the 2006–2007 group had a higher mastery approach score on average, which should predict an inclination to enter the competition, the 2005–2006 actually contained a higher percentage of people entering the competition. Similarly, the 2006–2007 group had a lower performance avoidance score on average, which again should predict an inclination to enter the competition, while as we now know the 2005–2006 actually contained a higher percentage of people entering the competition. Our hypothesis is hereby refuted and the difference in competition entry between the two years is more likely to have been caused by social factors, as discussed in the previous section.

5.4.3 Achievement Goals and Choice

Recall the hypotheses we formulated on the relationship between achievement goals and choice of experimental condition in Section 5.1.2:

1. When presented with two game-building tasks, an own-game and a pre-defined-game one, some participants will choose to write their own game and some the pre-defined game.
2. When presented with the options of entering an own-game into a competition and making it available to their peers, some participants will select these, while others will not.
3. There is a relationship between a participant's GAAGQ-results and the options they choose.
 - (a) Participants who choose the write their own game score
 - *higher* on mastery approach and performance approach,
 - but *lower* on mastery avoidance, performance avoidance, and work avoidance

compared to those who choose to write a pre-defined game.
 - (b) Participants who choose to enter the competition score
 - *higher* on performance approach
 - but *lower* on mastery avoidance, performance avoidance, and work avoidance

compared to those who write their own game but do not enter the competition.

Although Hypothesis 1 and Hypothesis 2 were found to be true, Hypotheses 3a and 3b were only partly supported by our data. In the case of Hypothesis 3a, the data provided support for decreased **mastery avoidance**, but no evidence was found for any of the other correlations. In the case of Hypothesis 3a, the data once again provided support for decreased **mastery avoidance**, and *some* support was also found for increased **performance approach** and decreased **work avoidance**. However, further research is needed to ensure the reliability of these last results on **performance approach** and **work avoidance**.

So although we can confirm our general hypothesis (3), it does come with caveats. First of all, only mastery avoidance influences a participant's choice between pre-defined and own-game. Second, mastery avoidance is also the only confirmed influence on competition choice, but there is some indication that mastery approach, performance approach, and work avoidance could be of influence as well.

5.4.4 Design

The study was conducted in an assessed assignment setting. In fact, participants received 5% of their marks for completing the questionnaire itself. Previous research has shown that assessment affects motivation for learning (see Section 3.4.1), so at a first glance, this could be considered to be of influence on our results.

However, the GAAGQ does *not* measure motivation on the game-building task; it measures a participant's general academic motivation, which should not be influenced by the fact that the game-building task is being assessed. The argument for the influence of assessment on our results cannot be entirely refuted though, because the whole undergraduate curriculum our participants were registered for is still very much based around assessment. So the influence of this much wider assessment can possibly have been of influence on the results of the GAAGQ. In general we would expect the general motivation to be lower than in a context without assessment. As for our particular study, it is conceivable that the percentage of own-game choosers would increase, since there would be less worry about not being able to come up with objects which satisfy the requirements, thus limiting the impact of a high score on avoidance goals. This would be an interesting direction for further research.

The fact that participants received 5% of their marks for completing the questionnaire could have caused participants to choose the 'correct' answer, rather than the answer that is true of them. However, an brief statement was included at the top of the questionnaire to minimize this effect:

Before you submit your game, you must answer some questions as a reflection on your learning. There are no right or wrong answers. There are only answers that are true of you, and answers that are not true of you.

Although the results may still to some extent have suffered from a *social desirability effect* (Ganster *et al.*, 1983), this statement should have taken away any doubts participants may have had regarding the 'academic correctness' of their answers.

A second factor which has the potential to influence our results is the choice of motivation framework. A choice for the achievement goal framework was made because it identified those aspects of motivation which were considered to be the most likely candidates for influencing choice of experimental condition. However, this hypothesis was not tested and it is possible there are other aspects of motivation influencing the participants' choice. This too would be an interesting direction for further research.

Similarly, our choice of motivation measure can potentially have had an effect as well. Although the (GA)AGQ was chosen in part because it has been tested for correctness and reliability, it is not the only measure for achievement goals. Other measures like PALS, which is also widely used, could have given us a different set of results.

5.4.4.1 Discrepancy between Data Sets

Although consistent support was found regarding the effect of mastery avoidance on choice of experimental condition, a number of the other hypotheses were supported by the 2006–2007 data, but not the 2005–2006 data, or even the overall data. To confirm the results of the 2006–2007 data, at least one more data-set is needed. Furthermore, because work avoidance was only measured the second year in which data was collected, results involving this achievement goal cannot be verified against a second data set. One more data-set would be desirably from this point of view as well.

5.4.5 Validity

In both the design and the execution of this study, every reasonable effort was made to ensure the validity of our results. However, the measure of *motivation* does raise some questions with respect to construct validity.

The **motivational framework** of achievement goals and the accompanying questionnaire (AGQ) do not give a complete motivational profile. As such, the framework reveals only a fraction of the full complexity of a person's motivation and focusses mainly on a person's fears and ambitions using a fairly crude set of distinctive classes of achievement goals. Different motivational frameworks focus on different aspects of motivation and would have possibly shone an entirely different light on our performance and enjoyment data.

Furthermore, there is also a validity issue with regard to **task equivalence** between the own-game and pre-defined scenario: the pre-defined task is not a minimal implementation of the list of requirements provided in the assignment. However, every

effort was made to make the pre-defined game a minimal implementation of the requirements, *while still being an actual game rather than an incoherent collection of interactive objects*. Furthermore, for all of the ‘additional’ programming involved in the pre-defined game it was possible to recycle code from essential parts of the game. In other words, no new concepts were used on top of the minimal implementation of the set of requirements.

5.4.5.1 Ethical Considerations

As mentioned in the Section 5.1.1.2, there are certain ethical considerations to be made when a study, such as this one, is conducted in a real-world setting. First of all, should research be conducted in a real-world setting at all? After all, the educational environment is being manipulated solely for the purpose of our study, yet these manipulations can potentially have real consequences for the participants of the study; i.e. it may have an impact on their marks.

Clearly, it is important to tread carefully in a situations like this. On the other hand, results from research that was conducted in a real-world environments can be much more valuable than results obtained from controlled ‘sterile’ environments, especially in education. That is, an approach which is shown to be effective in a controlled environment does not necessarily scale to a classroom environment since many other variables will have the potential to influence the effectiveness there. An approach which was shown to be effective *in a classroom environment* in the first place does not suffer from the problem.

Still, if we are to conduct research in a real-world environment, we will need to carefully consider the ethical questions involved. First of all, participants need to be given a choice of whether or not to participate. Because our study involved a assessed assignment, all participants had to do the assignment in order to get a mark, but they were given a choice of whether or not they wanted their data to be used in our study. If requested, they could have their data retracted from the study at any point. Participants were also given a short description of the study, a rough idea of what their data would be used for, and a contact email address for further questions.

Unfortunately, the wording of the information that participants were given did not allow us to use the participants’ overall academic performance. If this had been pick up on earlier – before the information was communicated to the participants – it would have been possible to run an interesting 3-way analysis between preference, motivation, and academic performance. However, as this would have been unethical, no such

analysis was carried out in the end.

Finally, there is the issue of task-equivalence which was mentioned earlier in Section 5.1.1.2 and Section 5.4.5. The pre-defined task is in fact *not* a minimal implementation of the list of requirements provided in the assignment. From an ethical point view though, the issue is at least partially addressed by the fact that for all of the ‘additional’ programming involved in the pre-defined game it was possible to recycle code from essential parts of the game. In other words, no new concepts were used on top of the minimal implementation of the set of requirements. Furthermore, participants were not forced into one or the other condition, but were asked to complete the task of their choice.

Finally, the own-game task required a certain amount of (game) design work compared to the pre-defined task, in which only programming work was required. This could lead the participants who choose to write their own game to become more proficient in their design skill. However, since design-skill was not on the curriculum for the course in question, this discrepancy should not directly disadvantage those who chose to write the pre-defined game.

5.5 Summary

This chapter has shown that there is indeed a majority demand for the availability of personalisation in assessed assignments, but the demand for publication-only was near non-existent and the demand for competition demonstrated a tendency to vary by year. Furthermore a relationship between achievement goals and choice of experimental condition was found, although not every achievement goal contributed to this relationship. In particular, only *mastery avoidance* was found to have a significant relationship with the pre-defined versus own game choice in that those who chose to write a pre-defined game generally showed higher mastery avoidance.

In the next chapter, we will investigate the enjoyment and performance of each of the participants on two different assignments: one *pre-defined-game* and one *own-game*. The study described in this next chapter was designed to find out if there is a certain type of student who does well on and/or likes own-game game-building tasks better than pre-defined-game game-building tasks. The ‘certain kind’ is expected to be definable in the motivational domain, since the two tasks differ only in the amount of personalisation offered. Just as some people’s affect and performance improve in the presence of competition while they decrease in others depending on a person’s

motivational type (Malone and Lepper, 1987), it is expected that some people's affect and performance will improve in an own-game situation while they will decrease in others for the same reason: differences in motivational types. As a preliminary test of this hypothesis, the current chapter showed that there is a certain kind of student, where 'certain kind' is defined in terms of motivation, who chooses to build their own game rather than copying a pre-defined one when given a choice.

Chapter 6

Study III: Affect and Performance

6.1 Introduction

In the previous chapter we saw that when given a choice between an own-game task and a pre-defined task, a majority of students chose the own-game task. By slight abuse of terminology, we defined a ‘demand’ for own-game building to be the following:

There is a demand for personalizable tasks among informatics undergraduates, if at least some of them choose these tasks over non-personalizable tasks when given a choice between the two. The size of the demand depends on the number of students preferring personalizable tasks over non-personalizable tasks.

As such, a majority demand for own-game building was observed in the previous chapter. The chapter also demonstrated that there is a relationship between a person’s motivation in terms of achievement goals and their choice for an own-game or pre-defined game task. However, the game-building task described in the previous chapter was designed with novice programming learning objectives in mind. Our learning-by-game-building research will now venture into the theoretical computer science (TCS) curriculum.

Chapter 4 already demonstrated how a game-building task can be specified to accomplish learning objectives within TCS, and Chapter 2 illustrated the need for improvement in enjoyment and performance in the area of TCS. The current chapter will test the effectiveness of the techniques presented in the former, in addressing the issues raised by the latter.

We do not expect learning-by-game-building to have the same influence on all participants’ performance and enjoyment, because the literature strongly suggests that

different approaches are successful for different kinds of people. For example, Stening (2002) provides evidence for the fact that some students acquire better reasoning skills when they are taught the TCS topic logic using a visualisation tool, while other acquire better reasoning skills when a traditional syntactic method is used. The same is true in the area of motivation. For example, Malone and Lepper (1987) propose a set of guidelines for the design of intrinsically motivating instructional environments. Lepper and Malone (1987) discovered that some of these guidelines, when implemented, may actually have opposite effects on different students.

The results of the study described in the previous chapter suggested that some aspects of a participant's motivational type, as measured by the GAAGQ, are correlated with their choice of game-building task. For example, there was a tendency for participants with a high mastery avoidance score to choose a pre-defined task over an own-game task. It might be inferred from this that mastery avoidance may be an indicator of choice.

The current chapter presents Study 3: an investigation into the effects of learning by game-building on enjoyment and performance in the area of TCS. In particular, we will investigate whether the level of personalisation offered in the task has a significant effect on these two factors. Therefore, the question at the centre of this study is whether or not a 'build-your-own' ('own-game') scenario differs from a 'build-this-particular-game' ('pre-defined') scenario in its effect on performance, enjoyment or both.

6.2 Design

6.2.1 Empirical Design

Both the own-game and pre-defined condition were encountered in Chapter 5. The main difference between the conditions used there and the conditions used in the current chapter is their function in the experimental design. In Chapter 5 participants were given a choice of experimental condition; in the current chapter, participants will be assigned to one and then the other. To avoid individual difference influencing the results of the study, a repeated measures design was used, such that two data-points were collected for each participant: one in the pre-defined condition and one in the own-game condition. In addition to the elimination of individual differences, a repeated measures design also ensures that no one group is advantaged over the other. This is of importance ethically, because the experimental condition has the potential to influence

course credit. In fact, one of our hypotheses is that it will.

The decision to use a repeated measures design meant that each participant completed two game-building tasks: a pre-defined one, and an own-game one. If the same game-building task or even the same TCS-topic were to be used for both game-building tasks, the study would be likely to have suffered from practice effects and possibly from boredom effects. To reduce the influence of these effects, a design involving two separate game-building tasks on two different TCS-topic was used for the two data-collection opportunities: Finite State Automata (FSAs) and Turing Machines (TMs). An additional advantage of this design was that it could be designed to fit into the curriculum of our participants without any overlap of learning objectives.

Unfortunately, there is also a disadvantage: differences in task and topic can themselves result in differences in enjoyment and performance, thus polluting our data. Although the difference between the two topics is out of our hands, the difference in game-building tasks can be minimized. The first measure to be taken in an effort to achieve this, was the use of identical task layout. Section 6.2.2 presents a detailed explanation of the tasks' 3-part layout. The second measure to be taken was the use of *normalized* marks rather than raw marks in the analyses. A *normalized* mark indicates how well a participant performed relative to their peers, rather than on an absolute scale. This measure does not only address the issue of difference in game-building task, but in a way, it also addresses the issue of different topics. For example if one topic were to be more difficult than the other, resulting in a lower average performance, the likelihood of finding a within-subjects effect with regard to performance would be decreased. Using the relative rather than the absolute performance allows us to avoid this problem.

Initially, participants were divided into two groups of equal size semi-randomly. In practice this meant that while each student did have a 50–50 chance of ending up in Group A or B, it was also ensured that the two groups were balanced with respect to the choice made in Study 2 (for those students who took part in Study 2). As such, Group A contained the same number of own-game choosers as Group B, the same number of pre-defined game choosers as Group B, the same number of competition choosers as Group B, and the same number of participants for whom no choice was recorded during Study 2.

In Study 3a, the first of the two data-collection opportunities, Group A was assigned to the pre-defined scenario, while Group B carried out the own-game task. In Study 3b, the second of the two data-collection opportunities, these roles were reversed

such that Group B was assigned to the pre-defined scenario, while Group A carried out the own-game task.

6.2.2 Measuring Enjoyment and Performance

Several measurements were collected during Study 3a and 3b. First of all, the dependent variables of the design, enjoyment and performance, were measured. For enjoyment the participants were asked to indicate how strongly they identified with the statement '*I liked this assignment*' on a 7-point Likert-scale ranging from *not at all true of me* (1) to *very true of me* (7). Because the game-building tasks were assessed for course credit, the marks received for the assignments were used as a measure of performance. The advantage of using this measure is that it is objective, because the marking was done by PhD students who were employed by the course and who were not involved with the study. To prevent inconsistency between markers, a combination of semi-automated marking scripts and detailed marking guides was used. Appendices G and J contain the marking guides used in Studies 3a and 3b respectively.

The game-building tasks, and the associated marks, were split into three parts: (A) introduction, (B) main game-building task, and (C) theory questions. In Part A, the participant was introduced to the game-engine step-by-step in a task similar to an interactive tutorial. On completion of Part A participants would have built a very small game. This task was designed to enable all participants to complete it without difficulty. Furthermore, it was the same for both experimental conditions. There are 40 marks to be earned on Part A.

In Part B the participants were asked to write a full game either of their own making, or a copy of a pre-defined game. Both kinds of games were assessed according to the number of requirements they satisfied. The list of requirements was provided to the participants as part of the task description and it was identical for both the own-game and the pre-defined version.

Besides this list of *positive requirements* on which participants could earn marks, a smaller set of *negative requirements* were also provided. These, as their name suggests, would *cost* rather than *earn* a participant marks if they failed to satisfy them. Two kinds of negative requirements were used: (1) requirements which enforce good programming practice, and (2) requirements which would allow for the games to be marked easily. Some examples of the former are:

1. Your code should be well documented.

2. Your variables should have informative names.
3. You should provide at least one label with each call to the `singleton`-function.

The latter included:

1. Your code should compile in less than 1 minute on a standard lab machine.
2. Your levels are composed using the function `composeLevels`.
3. Your game should include a dictionary called `gameDicti` which makes for a playable game.

The complete lists of requirements can be found in Appendices E and H. There were 30 marks in total to be earned on Part B.

In Part C, participants were asked to answer a number of theory questions related to the TCS-topic around which the game-building task was based. For the first task, this meant questions about FSAs and for the second task this meant questions about TMs. There were 30 marks to be earned on Part C and the questions were the same regardless of the experimental condition to which a participant was assigned to. The marks distribution was included in the task description and therefore known to the participants at the time they were working on the task.

6.2.3 Measuring Motivation and Time

In addition to enjoyment and performance, two more types of data were collected. First of all, participants are asked to indicate how much time they spent on each assignment on a 5-point scale: (a) less than 5 hours, (b) between 5 and 20 hours, (c) between 20 and 30 hours, (d) between 30 and 45 hours, or (e) more than 45 hours. This information was needed to control for the effect that time spent on a task may have an effect on performance on that task.

The participants' motivational type was also measured, using an Achievement Goal Questionnaire (AGQ) which measures five different achievement goals: mastery-approach, performance-approach, performance-avoidance, mastery-avoidance, and work-avoidance. Section 3.4.2.1 describes the AGQ, due to Elliot and McGregor (2001); Pieper (2003), in more detail.

The AGQ is a task-specific version of the General Academic Achievement Goal Questionnaire (GAAGQ) used in Chapter 5. There is much research such as (Julkunen,

1989; Boekaerts, 1986) to suggest that general motivation, also called *trait motivation*, is the relatively stable general motivation of an individual. Task-specific, also called state-motivation, is then the actual manifestation of this motivation in the learning context. It is suggested that the latter can be influenced by motivational embellishments, like personalisation and choice, while the former cannot.

For this reason, and because our interest lies mainly with the motivation a participant's experiences on game-building tasks, a choice was made to use the task-specific AGQ.

As an example of the shift in focus from the GAAGQ to the AGQ, consider the following question from the GAAGQ:

(2) I want to learn as much as possible this semester.

This question appears in the AGQ described by Elliot and McGregor (2001) in the following form:

(2) I want to learn as much as possible from this class

In our questionnaire, we are interested in motivation on a specific task, so the wording was changed to the following:

(2) I want to learn as much as possible from this assignment

Similar adaptations were made to the wording of the other fifteen questions to reflect a change from module-specific to task-specific. Figure 6.1 shows the AGQ as it was used in Studies 3a and 3b. Questions 1–3 measure mastery approach; Questions 4–6 measure performance approach; Questions 7–9 measure performance avoidance; Questions 10–12 measure mastery avoidance; and Questions 13–16 measure work avoidance. The questions were presented in the order suggested by Pieper (2003): 4, 7, 1, 13, 10, 5, 2, 8, 14, 3, 11, 6, 15, 12, 9, 16. All questions are answered on a 7-point Likert-scale, ranging from *not at all true of me* (1) to *very true of me* (7).

Furthermore, the AGQ was presented to participants only once, before they started the first task and crucially, before they knew which of the two groups they had been assigned to. At this point, they did not know that the task would be a game-building one. As far they were concerned, they were rating their motivation on an average course task.

As mentioned in the introduction, a participant's motivational type is measured in the hope that it will allow us to discriminate between the kind of participants on whom

1. Completely mastering the material in this assignment is important to me.
2. I want to learn as much as possible from this assignment.
3. The most important thing for me while working on the assignment is to understand the contents as thoroughly as possible.
4. My goal for this assignment is to get better grades than most of the other students.
5. It is important for me to do well compared to other students on this assignment.
6. I want to do better than other students on this assignment.
7. I just want to avoid doing poorly compared to other students on this assignment.
8. The fear of performing poorly on this assignment is what motivates me.
9. My goal for this assignment is to avoid performing poorly compared to other students.
10. I am afraid that I may not understand the content of the assignment as thoroughly as I'd like.
11. I worry that I may not learn all that I possibly could from this assignment.
12. I am definitely concerned that I may not learn all that I can from this assignment.
13. I really don't want to work hard for this assignment.
14. I want to do as little work as possible for this assignment.
15. I want to get a pass mark for this assignment by doing the least amount of work possible.
16. I look forward to working really hard for this assignment. **(Reverse scored)**

Figure 6.1: AGQ as used in Studies 3a and 3b.

learning-by-game-building has a positive effect and those on whom it has no effect or even a negative effect.

6.2.4 Derived Motivation Measures

There are several different ways to calculate a participant's motivational type given their AGQ-results. Three of these were used in the analyses presented in the current chapter. The alternative measures will be referred to as motivation(x), where x represents the number of variables that make up the measure. A description of each of the three derived measures and their calculations follows. The first two derived measures were taken from the literature, the third was defined for the current study in analogy with a derived measure defined by Kolb (1984).

Motivation(5) The first and most straightforward way to obtain a participant's motivational type from their AGQ-results is to use 5 dimensions, one for each of the motivational goals measured by the AGQ: (1) mastery-approach, (2) mastery-avoidance, (3) performance-approach, (4) performance-avoidance, and (5) work-avoidance. This measure is used in most studies involving an AGQ, e.g. Elliot and McGregor (2001); Pieper (2003).

Motivation(2) The major disadvantage of motivation(5) is revealed by its use in the analysis. As mentioned in the introduction, motivational type will be used as a predictor in the analysis. In other words, we would like to be able to distinguish between participants who benefit from an own-game approach and those who do not based on their motivational type. As such, the characteristic traits of own-game-benefitters should be concentrated in as few variables as possible to increase the power of the model. In motivation(5), information, characteristic or not, is dispersed rather than concentrated. For example, a participants' affinity with mastery goals is dispersed over two separate variables, mastery avoidance and mastery approach, while their affinity with avoidance goals is dispersed over three separate variables.

To reduce some of this spread of information, a radical reduction from five to only two dimensions was made. The result is motivation(2).

In motivation(2) mastery and performance goals are collapsed while work avoidance is discarded to form a motivational type with two dimensions: (1) avoidance and (2) approach. To obtain a participant's approach score, the mastery approach score and

performance approach score are added up. Similarly, a participant's mastery avoidance score and performance avoidance score are added up to obtain the new avoidance score.

While in theory collapsing the AGQ framework along the approach-avoidance axis is an option as well, see Harris *et al.* (2007), for an example of this, a collapse along the mastery-performance axis was deemed more suitable for our purposes because of the characteristics that are associated with approach-avoidance goals. The characteristics associated with avoidance goals include *disorganisation, low self-determination, and fear of failure*, while the characteristics *persistence while studying, higher grades, and overall need for achievement* are generally associated with approach goals (Elliot and McGregor, 2001). These descriptions appear to be viable factors based on which we can predict a participant's performance in the own-game condition compared to the pre-defined condition. This argument will be elaborated on in Section 6.4.

Although the information in motivation(2) is much more 'concentrated' compared to motivation(5), there is less of it. For example, if the mastery score were a good discriminator of participants who benefit from an own-game approach and those who do not, this would never be discovered using motivation(2).

Therefore, our third and last measure of motivational type tries to capture both: preservation of information without dispersion.

Motivation(3) In motivation(3), which was designed specifically for the study described in the current chapter, there are three dimensions: (1) *mastery-performance score*, (2) *approach-avoidance score*, and (3) *total motivation score* or *motivation size*. These three scores are extracted from a participant's answers to the first twelve questions of the AGQ to obtain a 3D-framework similar to the 4D-framework described by Myers and Myers (1995) or the 2D-framework described by Kolb (1984). Kolb presented a framework of learning-styles consisting of two axes: (x) a doing/planning – reviewing/watching axis and (y) a thinking/concluding – sensing/feeling axis. In his system a participant's learning style could be described by two coordinates, one for each of the axes. Motivation(3) follows the same principle for mapping the *mastery-performance score* onto x-axis and the *approach-avoidance score* onto y-axis, but then adds a z-axis. This third axis will be justified after the calculations of the first two axis.

To determine a participant's *mastery-performance score*, the following calculation was performed:

$$(\text{sum of performance questions}) - (\text{sum of mastery questions})$$

As a result, our mastery-performance score ranges from -36 to 36, with -36 indicating an extreme orientation toward mastery goals, 36 indicating an extreme orientation toward performance goals, and 0 indicating equally strong orientations toward mastery and performance goals.

A similar calculation is performed to obtain a participant's *approach-avoidance score*:

$$(\text{sum of avoidance questions}) - (\text{sum of approach questions})$$

Once again, the new score ranges from -36 to 36, with a score of -36 indicating an extreme orientation toward approach goals, a score of 36 indicating an extreme orientation toward avoidance goals, and a score of 0 indicating equally strong orientations toward approach and avoidance goals. Work-avoidance was omitted from the calculations, because it was unreconcilable with the system described in Kolb (1984). That is, in order to obtain a representative score on the avoidance-approach axis, the approach score should have the same range and precision as the avoidance score. This is true when only mastery avoidance/approach and performance avoidance/approach are used, but false when work avoidance is entered into the equation.

Although this new representation of a participant's motivational type is more compact, while still preserving a lot of information, one important aspect of the original 5-dimensions motivational type, besides work-avoidance, has been lost. For example, we can no longer distinguish between Participant A who scored a total of 42 on the performance questions and a total of 42 on the mastery questions, and Participant B who scored just 6 on each: they both score 0 on the mastery-performance scale. To regain this potentially vital information, we introduce the third and last component of the motivational type: the *total motivation score*. The total motivation score can be calculated by taking the sum of a participant's answers to all twelve questions. As a result of this definition, it ranges from 12 to 84. On this scale, a score of 12 indicates a general lack of identification with any of the achievement goals, a score of 84 indicates an extreme identification with all of the achievement goals, and a score of for example 48 indicates a moderate identification with the achievement goals.

Throughout the rest of this chapter, we will refer to the motivational type described above as *motivation(3)*. The '3' refers to the fact that *motivation(3)* is not a single variable, but a set of three which together capture the motivational type of a participant.

6.3 The Game-Building Tasks

Python was the designated language for the Inf2A module, which is why both game-building tasks are programmed in this language using two game-engines specifically designed and implemented for this study by the author. Both game-engines were built on top of existing open-source libraries covering the modelling skills in question: FSAs and TMs.

There is no reason why the study described in the current chapter could not be done in for example Haskell, SML, Java, Perl, etc. In fact, it would probably be even easier, because a *working* FSA library already exist for all of these languages.

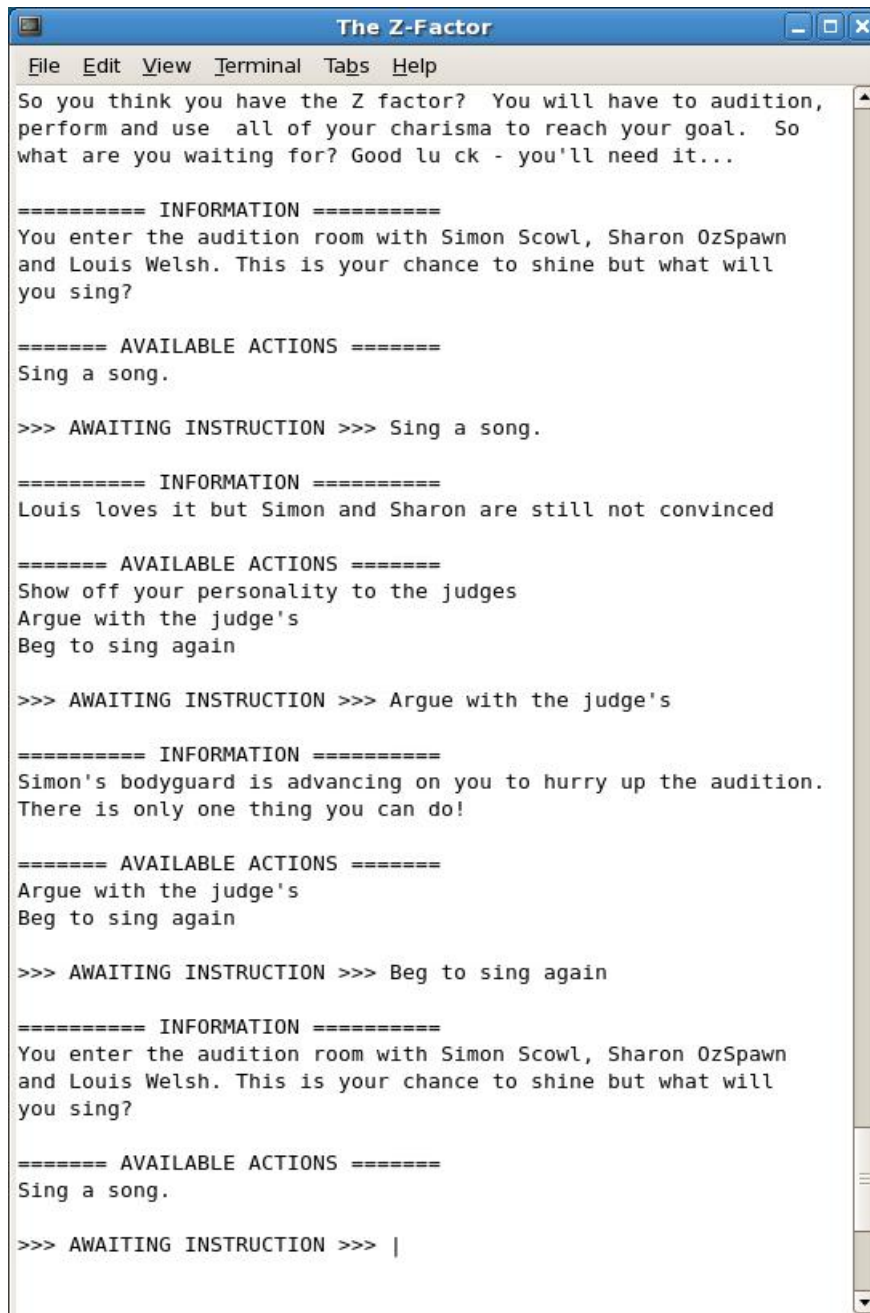
6.3.1 Study 3a: Finite State Automata

In Section 4.2.1 the basic concept of writing a game using the FSA framework and a game-engine was described. To provide a better understanding of the task participants were asked to complete, we will discuss (part of) the game shown in Figure 6.2 which was written by one of the participants. The code for the first level of this game can be found in Figure 6.3.

The first thing to notice is how simple the program is and how little the participant needed to implement in order create a single level. The second thing to notice is that two of the FSA-combinators mentioned in Section 4.2.1 appear in this piece of code: concatenation and closure. The former is self-explanatory whereas the second represents the Kleene-star operation used in regular expressions. Most importantly though, there is an instance of the function `singleton`¹. This function creates the basic FSAs which are the building blocks of the game-FSA. The function takes three arguments: (1) the action or label for the FSA-transition, (2) a description of state 0, and (3) a description of state 1. Figure 6.4 illustrates how the function `singleton` uses these three arguments. The left window is a Python interactive shell and shows the `show`-function² being called on the `song`-object. This is the `song`-object defined in Figure 6.3. The `show`-function does two things: (1) it displays the FSA belonging to `show` in a new window, and (2) it displays information for each of the states of the FSA in the Python interactive shell window. As we mentioned in Section 4.3.1, the state information is not a part of the standard FSA framework, but an addition required

¹There were in fact three more instances of `singleton` – defining the variables `beg,talk`, and `argue` – but these have been taken out for brevity

²Available from the FSA-library.



```
The Z-Factor
File Edit View Terminal Tabs Help
So you think you have the Z factor? You will have to audition,
perform and use all of your charisma to reach your goal. So
what are you waiting for? Good luck - you'll need it...

===== INFORMATION =====
You enter the audition room with Simon Scowl, Sharon OzSpawn
and Louis Welsh. This is your chance to shine but what will
you sing?

===== AVAILABLE ACTIONS =====
Sing a song.

>>> AWAITING INSTRUCTION >>> Sing a song.

===== INFORMATION =====
Louis loves it but Simon and Sharon are still not convinced

===== AVAILABLE ACTIONS =====
Show off your personality to the judges
Argue with the judge's
Beg to sing again

>>> AWAITING INSTRUCTION >>> Argue with the judge's

===== INFORMATION =====
Simon's bodyguard is advancing on you to hurry up the audition.
There is only one thing you can do!

===== AVAILABLE ACTIONS =====
Argue with the judge's
Beg to sing again

>>> AWAITING INSTRUCTION >>> Beg to sing again

===== INFORMATION =====
You enter the audition room with Simon Scowl, Sharon OzSpawn
and Louis Welsh. This is your chance to shine but what will
you sing?

===== AVAILABLE ACTIONS =====
Sing a song.

>>> AWAITING INSTRUCTION >>> |
```

Figure 6.2: Example of an FSA Game.

```

# Level 1 - Audition

song = FSA.singleton( "Sing a song."
                    , "You enter the audition room with Simon \
                      Scowl, Sharon OzSpawn and Louis Welsh. \
                      This is your chance to shine but what \
                      will you sing?"
                    , "Louis loves it but Simon and Sharon are \
                      still not convinced" )

beg = ...
talk = ...
argue = ...

audition1 = FSA.concatenation(song, argue)
audition2 = FSA.closure(FSA.concatenation(audition1, beg))
audition3 = FSA.concatenation(audition2, song)
audition4 = FSA.concatenation(audition3, talk)

levell = FSA.minimize(audition4)

```

Figure 6.3: Partial Code of an FSA Game (First Level Only).

by the game-engine.

6.3.2 Study 3b: Turing Machines

In Section 4.2.3 we described the basic concept of writing a game using the TM framework and a game-engine. This process of programming the TM is fairly straightforward but, just like the FSA framework, it needed a few additions before it could be turned into a playable game. We will discuss each of these additions in turn, using the game shown in Figure 6.5 – written by one of the participants – as an example.

As described in Section 4.2.3, the TM used for writing games has five tapes. The first one represents the game world and because none of the information on this tape is ever being fed directly to the player, no augmentations are needed for the game-engine to interpret this tape.

The second tape stores the current values of all the game-variables. As can be seen in Figure 6.5, these values can be requested by the player using the ‘I’-command. In order for the game-engine to offer this functionality, it needed to be provided with a function which turns a tape into the information provided after each ‘I’-request.

The third tape is the output-tape. Everything on this tape is directly presented to the player of the game. Unfortunately, the TM only reads one symbol at a time, so the

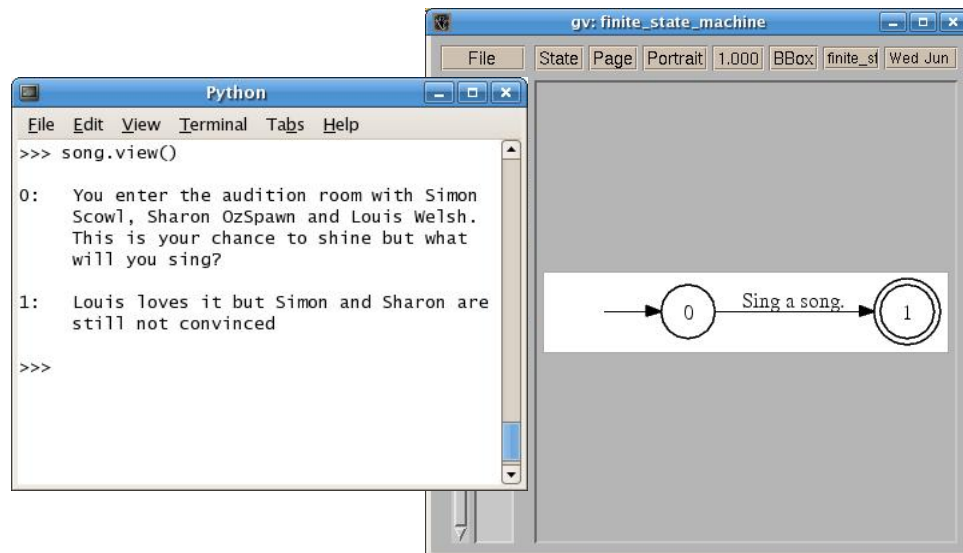


Figure 6.4: Screenshot demonstrating the Behaviour of singleton.

participant needed to provide the game-engine with a dictionary mapping each symbol which can occur on the output-tape onto a more intelligible message. For example, the game in Figure 6.5 uses a dictionary containing the following entries:

- (a) You are in your home village. Commands: display (I)nformation, move (n)orth, (e)ast, (s)outh, or (w)est.
- (w) Wasteland: there is nothing to gain or lose here.
- (s) Spring: fill your w(a)ter flask, or move on (n,e,s, or w).
- (W) You've gained a new resource: water!
- (c) Cave: there is a bear on the path. If you run (n,e,s, or w) you will not gain anything, but you won't die either. If you fight the (b)ear, you may gain a valuable resource, but you can also get hurt.
- (L) You lost the fight, but are still alive. You have lost 1 health though.
- (m) Mountains: pick up some (r)ocks, or move on (n,e,s, or w).
- (R) You've gained a new resource: rock!

The fourth and fifth tape are implemented entirely by the game-engine and no input from the participant is required. The latter should still be considered an augmentation

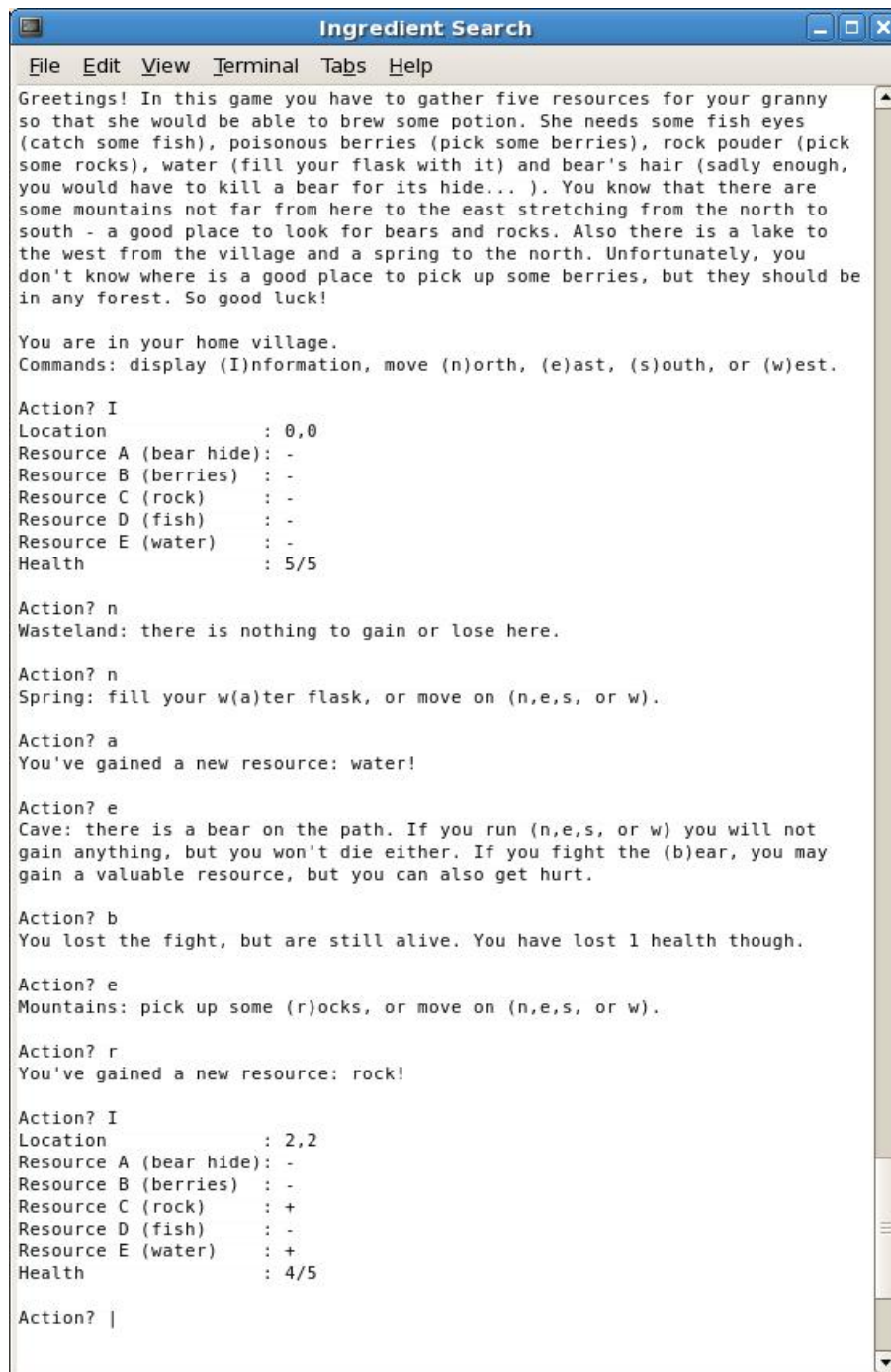


Figure 6.5: Example of a TM Game.

though, because input is not part of the standard TM framework. One last augmentation that was made concerns the TM program the students write. So-called *wild-cards* were introduced in order to reduce the number of TM rules participants had to write. Wild-cards were used instead of tape-symbols in TM-rules. These TM-rules can be found on the transitions of a TM-program. When a wild-card is used in a TM-rule, this rule can be applied to any tape-symbol for which no other rule is specified. In effect, one TM-rule using a wild-card is the equivalent of one or more rules without wild-cards. Figure 6.6 shows a TM program with wild-cards – denoted by \$ – on the left and that same program without wild-cards on the right. Both programs succeed if the input on their first tape is the same length as the input on their second tape and if the inputs are not shorter than one. This is an example of a very small program with only two tapes and an alphabet containing only two characters and already there is a dramatic increase in transitions when the wild-cards are taken out. In a 5-tape TM with a much larger alphabet, this exponential increase would make it impossible for participants to write a game without wild-cards.

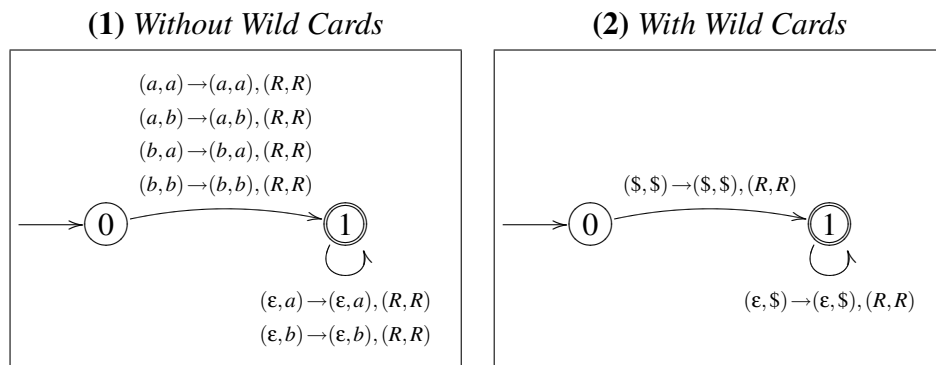
6.4 Hypotheses

The study described in this chapter was designed to test the hypothesis that there is a difference between an own-game scenario and a pre-defined scenario in terms of a participant's performance and enjoyment on the game-building task. An overview of the hypotheses for this study follows.

6.4.1 Performance

1. When participating in the own-game condition, participants will achieve a performance (measured using marks out of 100) equal to their performance in the pre-defined condition.

In terms of performance no significant overall difference between a pre-defined scenario and an own-game scenario is expected. In other words, the null hypothesis is expected to hold. It is possible that participants in the own-game condition do slightly better than those in the pre-defined condition because there is more choice and personalization available (Cordova and Lepper, 1996), but there is also evidence against a claim like this, such as that in (Parker and Lepper, 1992).



N.B. These two TM-programs are written for a 2-tape TM. Because 2-tape TMs have two heads, the rules in their programs looks slightly different from those in the program of a single-tape TM. In particular, wherever a single-tape program rule would use a single tape-symbol, the 2-tape program rule uses a pair of tape-symbols and where a single-tape program rule would use a single head-movement instruction, the 2-tape program rule uses a pair of head-movement instructions. The first item of the pair refers to the tape-symbol or head-movement of the first tape and the second item refers to the tape-symbol or head-movement of the second tape.

Figure 6.6: The Effect of removing Wild-Cards.

2. When participating in the own-game condition, participants will achieve a higher performance (measured using marks out of 100) compared to the pre-defined condition, after controlling for *the amount of time spent on a game-building task* (measured on a 5-interval scale) and a participant's *motivational type* (as measured by the AGQ).

However, if such an effect does exist, controlling for the amount of time spent on the tasks and for motivational type will give a more accurate picture of this effect for the following reasons. First, the time a participant spends on a task is an obvious external influence on performance: the more time (s)he spends on the task, the higher their performance will generally be. Second, there is strong support in the literature that different students thrive in different educational settings as described in Section 3.5. The current study is not expected to be any different in this respect, the only question that remains is how to model 'different students'. Motivation was deemed a suitable candidate for this control factor, because the previous chapter provided initial evidence for its use as a discriminator between an own-game condition and a pre-defined condition.

3. Participants who score highly on avoidance will achieve a higher performance (measured using marks out of 100) in the pre-defined condition than in the own-game condition.

Besides the main effect of condition described above, some interaction effects are also expected to show up, in particular between aspects of the control factor motivation and the experimental condition. First of all, we expect an interaction effect between avoidance goals and condition, where participants who score highly on avoidance, score higher on a pre-defined task than on an own-game task. This expectation is based on the fact that people scoring highly on avoidance goals tend to *be disorganized, lack self-determination, and have a fear of failure* (see Chapter 3 for additional details). We expect that these people may feel overwhelmed by the freedom and design-aspect of an own-game task, compared to the safety of a pre-defined task. This may lead to decreased motivation, which in turn can lead to impaired performance.

4. Participants who score highly on approach will achieve a higher performance (measured using marks out of 100) in the own-game condition than in the pre-defined condition.

Similarly, we expect an interaction effect between approach goals and condition, where participants who score highly on approach, score higher on an own-game task

than on a pre-defined task. This expectation is based on the fact that people scoring highly on approach goals tend to *have high competency expectancies, achieve higher grades, and have an overall need for achievement* (see Chapter 3 for details). The own-game task gives these people additional opportunities for achievement, both personal and social, which can lead to increased motivation, which can in turn lead to improved performance.

5. Participants with a low motivation score will achieve a higher performance (measured using marks out of 100) in the own-game condition than in the pre-defined condition.

Finally, we expect an interaction effect between motivation size and condition where poorly motivated participants, i.e. participants with a low motivation score, score higher on an own-game task than they do on a pre-defined task; but where the opposite is not necessarily true of highly motivated participants, due to a ceiling-effect. That is, if a student is already highly motivated, one cannot expect to improve their performance by improving their motivation, because it may already be at a maximum.

6. A participant's gender is not related to their performance (measured using marks out of 100) by condition.

The analysis used is incremental, which means the factors get introduced in order, starting with no factors and followed immediately by the one most likely to be of influence. As explained above, time spent on the tasks is expected to fit this profile. Motivation was also mentioned as a possible predictor, so it is introduced in the third analysis. The fourth and final analysis introduces gender. Gender is not expected to have any influence on performance and Analysis 4 was designed to test this hypothesis.

6.4.2 Enjoyment

1. When participating in the own-game condition, participants will experience an increased sense of enjoyment (measured on a 7-point Likert-scale), compared to the pre-defined condition.

In terms of enjoyment a significant overall difference between a pre-defined scenario and an own-game scenario is expected, with participants enjoying the own-game condition better than the pre-defined condition. This hypothesis is based on the literature described in Section 3.4.3, which claims that personalisation and choice have a positive effect on enjoyment.

2. The main effect mentioned in Hypothesis 1 will be stronger when after controlling for motivational type (as measured by the AGQ) and gender.

The same effect may be stronger when controlling for motivational type and gender, since both of these factors have the potential to influence enjoyment, the first one because of the strong relationship between motivation and enjoyment and the second because gender studies have revealed that female participants respond differently to game-based environments than their male counterparts. For example, Goldstein (1994); Provenzo (1991) provide evidence for the fact that genders differ in game-playing in terms of interest, use, and performance; Kafai (1996) found differences in game-design in terms of characters created, feedback, and narrative development; and (Malone and Lepper, 1987) found a difference between genders in terms of the motivational appeal of games.

3. When participating in the pre-defined condition, participants who score highly on avoidance will experience an increased sense of enjoyment (measured on a 7-point Likert-scale) compared to the own-game condition.

Besides the main effect of condition described above, some interaction effects are also expected, in particular between aspects of the control factor motivation and the experimental condition. First of all, we expect an interaction effect between avoidance goals and condition, where participants who score highly on avoidance, enjoy a pre-defined task more than an own-game task. The same reasoning used for performance applies: people scoring highly on avoidance goals tend to *be disorganized, lack self-determination, and have a fear of failure*. As a result, we expect that these people may feel overwhelmed by the freedom and design-aspect of an own-game task, compared to the safety of a pre-defined task. This may cause them to enjoy the own-game task less than the pre-defined task.

4. When participating in the own-game condition, participants who score highly on approach will experience an increased sense of enjoyment (measured on a 7-point Likert-scale) compared to the pre-defined condition.

Similarly, we expect an interaction effect between approach goals and condition, where participants who score highly on approach enjoy an own-game task more than a pre-defined task. Once again, the same reasoning we used for performance applies: people scoring highly on approach goals tend to *show persistence while studying*,

achieve higher grades, and have an overall need for achievement. The own-game task gives these people additional opportunities for achievement, both personal and social, which can cause them to enjoy the own-game task more than the pre-defined task.

5. When participating in the own-game condition, participants with a low motivation score will experience an increased sense of enjoyment (measured on a 7-point Likert-scale) compared to the pre-defined condition.

As with performance, we expect an interaction effect between motivation size and condition, where poorly motivated participants, i.e. participants with a small motivation size, enjoy an own-game task more than a pre-defined task; but where the opposite is not necessarily true of highly motivated participants, due to a ceiling-effect. That is, if a student is already highly motivated and enjoying game-building assignments, one cannot expect to improve much on this as it may already be at maximum. However, there is a lot more room for improvement with poorly motivated students. A second factor that may be of influence was raised by Squire (2004). He observed that students who are performing well in the current educational context can be resistant to innovations, possibly because they feel they might lose their place at the top of the pecking order. On the other hand, students who are not reaching their potential in the current educational context might have a more open mind with regard to innovations and show increased enjoyment or even performance in the new context.

6. A participant's gender is related to their relative sense of enjoyment (measured on a 7-point Likert-scale) of the two conditions (i.e. own-game and pre-defined).

Finally, an interaction effect between gender and condition is expected, the nature of which is difficult to determine. There are numerous publications demonstrating difference between genders in game-based environments, some of which were mentioned under Hypothesis 2. However, it is difficult to formulate an exact hypothesis based on these publications with regard to which gender would enjoy an own-game task better and which would prefer a pre-defined task. Instead, the two-way hypothesis as formulated above is maintained, with the idea of deriving the direction from the data obtained.

Just like performance, the analysis used for enjoyment is incremental, which means the factors get introduced in order, starting with no factors and followed immediately by the one most likely to be of influence. As explained above, motivation is expected to fit this profile. Gender was also mentioned as a possible predictor, so it is introduced

Table 6.1: Breakdown of Participants by Gender and Degree.

		Required for Degree		Total
		<i>Yes</i>	<i>No</i>	
Gender	<i>Female</i>	14	0	14
	<i>Male</i>	73	3	76
Total		87	3	90

in the third analysis. Time, the most important predictor in the performance analysis, does not play any role in the enjoyment analysis because we believe there is no reason to believe it has any influence on task enjoyment. Although a large time expenditure may *indicate* more enjoyment, or more frustration for that matter, it is unlikely that this effect also works in the opposite direction.

6.5 Method

6.5.1 Design

The study was conducted as a within-group, or dependent, design with two conditions: (1) pre-defined and (2) own-game.

6.5.2 Participants

Ninety undergraduates participated in this study. All participants were taking a second-year Informatics course in *Processing Natural and Formal Languages* at the University of Edinburgh during the academic year 2006–2007. There was significant overlap with the participants in the 2005–2006 group used in Study 2 described in Chapter 5. The participants ranged in age from 18 to 40 with a mean age of 20. Of the 90 participants, 84% were male and 16% female. Furthermore, 97% of participants were required to take the course as part of their degree and 3% were not. Table 6.1 shows a detailed breakdown.

6.5.3 Materials

The materials used in Study 3 comprised sixteen parts in total, eight of which were used in Study 3a and eight in Study 3b.

6.5.3.1 Study 3a

In Study 3a the following materials were used:

FSA Game Engine: A purpose-built game engine in Python allowed for games like the one in Figure 6.2 to be built. The game-engine was based on the FSA module for Python by Oliver Steele (Steele, www) and further developed by Laura Hutchins-Korte as part of Study 3a.

API: The API of the game engine provided detailed documentation of all the features available in the game-engine (see Appendix F).

Pre-Defined Hand-out: A 7-page hand-out described Parts A and B of the pre-defined game-building task. In other words, it contained an introduction to the game-engine and the main pre-defined game-building task participants were asked to perform. The latter included the list of requirements the games had to satisfy (see Appendix E).

Own-Game Hand-out: A 7-page hand-out described Parts A and B of the own-game game-building task. The introduction to the game-engine, or Part A, is identical to the introduction to the game-engine in the *Pre-Defined Hand-out*, but the main game-building task, or Part B, describes an own-game task rather than a pre-defined one. Part B also included the list of requirements the games had to satisfy (see Appendix E).

AGQ: The AGQ was presented to participants through a web-form.

Theory Questions: Part C of the assignment was also presented to participants through a web-form. Part C was the final part of the game-building task, in which theory questions about FSAs and Regular Expressions (REs) were asked. The form also contained a question asking participants how much time they had spent on the game-building task.

Playable Demo: A playable demo of the pre-defined game was available to participants in Group A.

Marking Script and Guide: A semi-automated marking script together with a detailed marking guide formalize the performance measure (see Appendix G).

6.5.3.2 Study 3b

In Study 3b the following materials were used:

TM Game Engine: A purpose-built game engine in Python allowed for games like the one in Figure 6.5 to be built. The game-engine was based on the TM module for Python by Rogério Reis and Nelma Moreira (Reis and Moreira, 2004) and further developed by Laura Hutchins-Korte as part of Study 3b.

API: The API of the game engine provided detailed documentation of all the features available in the game-engine (see Appendix I).

Pre-Defined Hand-out: A 9-page hand-out described Parts A and B of the pre-defined game-building task. It contained an introduction to the game-engine and the main pre-defined game-building task participants were asked to perform. The latter included the list of requirements the games had to satisfy (see Appendix H).

Own-Game Hand-out: A 9-page hand-out described Parts A and B of the own-game game-building task. The introduction to the game-engine, or Part A, is identical to the introduction to the game-engine in the *Pre-Defined Hand-out*, but the main game-building task, or Part B, describes an own-game task rather than a pre-defined one. Part B also included the list of requirements the games had to satisfy (see Appendix H).

AGQ: The AGQ was presented to participants through a web-form.

Theory Questions: Part C of the assignment was presented to participants through a web-form. Part C was the final part of the game-building task, in which theory questions about TMs were asked. The form also contained a question asking participants how much time they had spent on the game-building task.

Partial TM: Participants were given a small TM to start them off and a larger one for use in the game-engine introduction and in the game-building task if they chose to do so.

Partial Wrapper: Participants were also given an initial wrapper, which contained the initial dictionaries used by the game-engine to interpret the symbol on the TM-tapes of the TM they were given.

Marking Guide: A semi-automated marking script together with a detailed marking guide formalize the performance measure (see Appendix G).

6.5.4 Procedure

At the start of Study 3, participants were divided into two groups of equal size semi-randomly. That meant that while each student did have a 50–50 chance of ending up in group A or B, it was also ensured that the groups were balanced with respect to the choice made in Study 2 (for those students who took part in Study 2). As such, Group A contained the same number of own-game choosers as Group B, the same number of pre-defined game choosers as Group B, the same amount of competition choosers as Group B, and the same amount of participants for which no choice was recorded during Study 2.

In Study 3a, Group A was assigned to the pre-defined scenario, while Group B did the own-game scenario. In Study 3b these roles were reversed such that Group B was assigned to the pre-defined scenario, while Group A did the own-game scenario. The materials given to Group A in Study 3a comprised: FSA game-engine, API, pre-defined hand-out, web form 1, web form 2, and the playable demo. Group B was given the following: FSA game-engine, API, own-game hand-out, web form 1, and web form 2. In both Group A and Group B, web form 1 was presented to the participants before any of the other materials and before they knew to which group they had been assigned. In Study 3b the materials given to Group A comprised: TM game-engine, API, own-game hand-out, web form, partial TM, and partial wrapper. Group B was given the following: TM game-engine, API, pre-defined hand-out, web form, partial TM, and partial wrapper.

Participants were given two weeks to complete each of the game-building tasks each of which counted for 1/16 of their final mark. The set guideline for the amount of time to be spent on each of the assignments was 25 hours. During the weeks in which participants were working on the game-building tasks a newsgroup regularly answered by course staff was available to those who needed help with the task.

At the end of Study 3a, participants handed in the following: (a) the answers to Part A in the form of a Python source file, (b) their game, or solution to Part B, also in the form of a Python source file, and (c) the answers to Part C by answering the questions on the web-form. At the end of Study 3b, they handed in: (a) the answers to Part A in the form of a TM source file (an extension of the *partial TM*) plus dictionaries specified in a Python source file (an extension of the partial wrapper), (b) their game, or solution to Part B, also in the form of a TM source file plus dictionaries specified in a Python source file, and (c) the answers to Part C by answering the questions on the

web-form.

The materials which the participants had handed in were then marked according to the semi-automatic marking script and marking guidelines by two independent markers who were not otherwise involved with the research project. The total mark of either assignment was composed of three parts: (1) the marks for the introduction to the game-engine accounted for 40%, (2) the marks on the game itself accounted for 30%, and (3) the marks on a set of theory questions account for another 30%.

6.5.4.1 Analyses

The analyses of performance and enjoyment in the own-game condition versus performance and enjoyment in the pre-defined condition were conducted incrementally. That is, after the initial comparison, predictors or factors will be added, one at a time, starting with the one we expect to have the greatest effect on performance. Justification for these expectations are provided in Section 6.4.

For the performance data, the analysis was initiated with a repeated-measures t-test, with condition as an independent variable, resulting in Performance Model 1. All subsequent analyses also used condition as an independent variable. For Performance Model 2, a one-way repeated measures ANCOVA with two covariates was conducted. The times spent on the tasks of either condition served as covariates. Performance Model 3 was also the result of a one-way repeated measures ANCOVA, but with seven covariates for Performance Model 3a, four for Performance Model 3b, and five for Performance Model 3c. In addition to the times spent on the tasks of either condition, Analysis 3a uses the 5-way motivational type as covariates. Similarly, Analysis 3b uses the 2-way motivational type as covariates and Analysis 3c the 3-way motivational type. See Section 6.2.4 for details of the motivational types.

For the final performance analysis, a two-way mixed ANCOVA with seven, four, or five covariates was used, resulting in Performance Model 4. Besides the independent variable and covariates already mentioned, Analysis 4 introduced the independent variable gender.

For the enjoyment data, the analysis was also initiated with a repeated-measures t-test, with condition as an independent variable, resulting in Enjoyment Model 1. All subsequent analyses also used condition as an independent variable. Enjoyment Model 2 was the result of a one-way repeated measures ANCOVA but with seven covariates for Enjoyment Model 2a, four for Enjoyment Model 2b, and five for Enjoyment Model 2c. All of the covariates modelled motivational type. Analysis 2a used a 5-way moti-

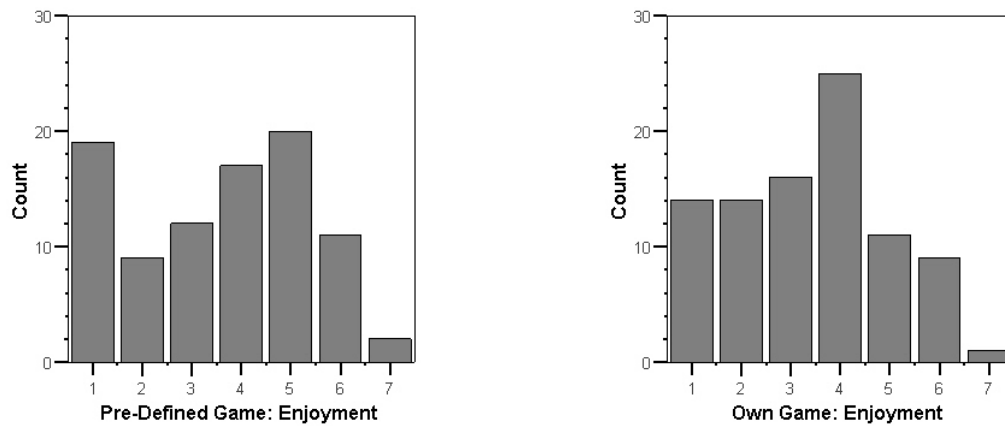


Figure 6.7: Comparison of enjoyment by condition.

vational type, Analysis 2b a 2-way motivational type, and analysis 2c a 3-way motivational type.

For the final enjoyment analysis, a two-way mixed ANCOVA with five, two, or three covariates was used, resulting in Enjoyment Model 3. Besides the independent variable and covariates already mentioned, Analysis 3 introduced the independent variable gender.

In the remainder of this chapter, each of the predictors will be referred to by its name and cardinality. The cardinality indicates how many separate variables the predictors is composed of. For example, *time(2)* is composed of two separate variables: (1) the time spent on the assignment in condition 1 and (2) the time spent on the assignment in the condition 2.

6.6 Results

6.6.1 Raw Data

6.6.1.1 Enjoyment

Figure 6.7 and Table 6.2 show the enjoyment of participants over conditions. In all of the analyses in this section and the one following, we will use the within-subjects variable *condition* to refer to the pre-defined versus own game conditions.

Table 6.2: Participants' enjoyment across conditions.

Enjoyment of ...		Statistic	Std. Error
Pre-Defined Game Task	Mean	3.57	0.19
	Std. Dev.	1.79	
Own-Game Task	Mean	3.40	0.167
	Std. Dev.	1.58	

6.6.1.2 Performance

Figure 6.8 and Table 6.3 show the marks earned over conditions for both Study 3a and 3b, where A2 is the game-building task used Study 3a (FSAs) and A3 the game-building task used in Study 3b (TMs).

Initial Check On average, participants in Group A, who wrote a pre-defined game in Study 3a and their own game in Study 3b earned slightly higher marks on the two task together ($M = 119.32, SE = 4.00$) than participants in Group B ($M = 115.07, SE = 4.34$), who wrote their own game in Study 3a and a pre-defined game in Study 3b. This difference was not significant $t(88) = 0.718, p > .45$.

6.6.1.3 Motivation (Achievement Goals)

Figure 6.9 shows the raw results from the AGQ. Scores on mastery-approach, performance-approach, mastery-avoidance, and performance-avoidance ranged from 3 to 21, while scores work-avoidance ranged from 4 to 28.

6.6.1.4 Time Spent

Figure 6.10 shows the amount of time spent by Groups A and B on both of the game-building tasks, where A2 is the game-building task used Study 3a and A3 the game-building task used in Study 3b. Students were expected to spend 25 hours on the assignment, so 20–30 hours was chosen as the middle category. It was estimated that a large number of students would be able to complete the task in less than 10 hours, so a lower cut-off of 5 was chosen instead. To balance this 15-hour interval, another 15-hour interval was specified from 30–45 hours.

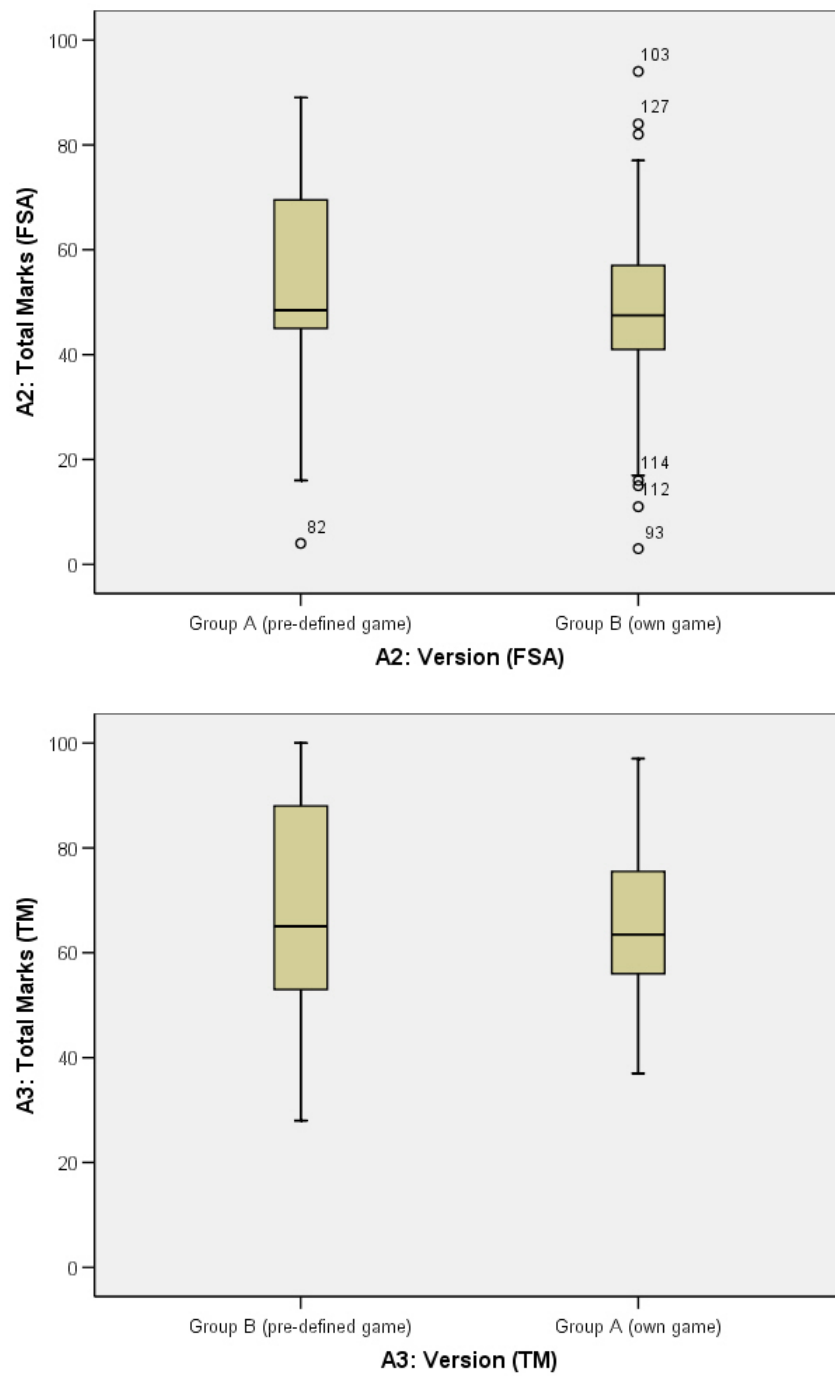


Figure 6.8: Comparison of participants' raw performance across groups and tasks.

Descriptive Statistics

Abbr.		Median	Mean	Std. Dev.
MAp	<i>Mastery-Approach</i>	12.00	15.04	3.753
PAp	<i>Performance-Approach</i>	12.00	11.89	4.241
PAv	<i>Performance Avoidance</i>	12.00	11.59	4.426
MAv	<i>Mastery Avoidance</i>	12.00	11.90	3.721
WAv	<i>Work Avoidance</i>	13.50	13.20	4.439

Correlations: Kendall's τ

	MAp	PAp	PAv	MAv	WAv
MAp	1	0.171*	0.027	0.080	-0.319**
PAp		1	0.331**	0.239**	-0.093
PAv			1	0.347**	-0.038
MAv				1	0.057
WAv					1

** Correlations significant at the 0.01-level.

* Correlations significant at the 0.05-level.

Figure 6.9: AGQ Descriptives and Correlations (based on 90 data-points).

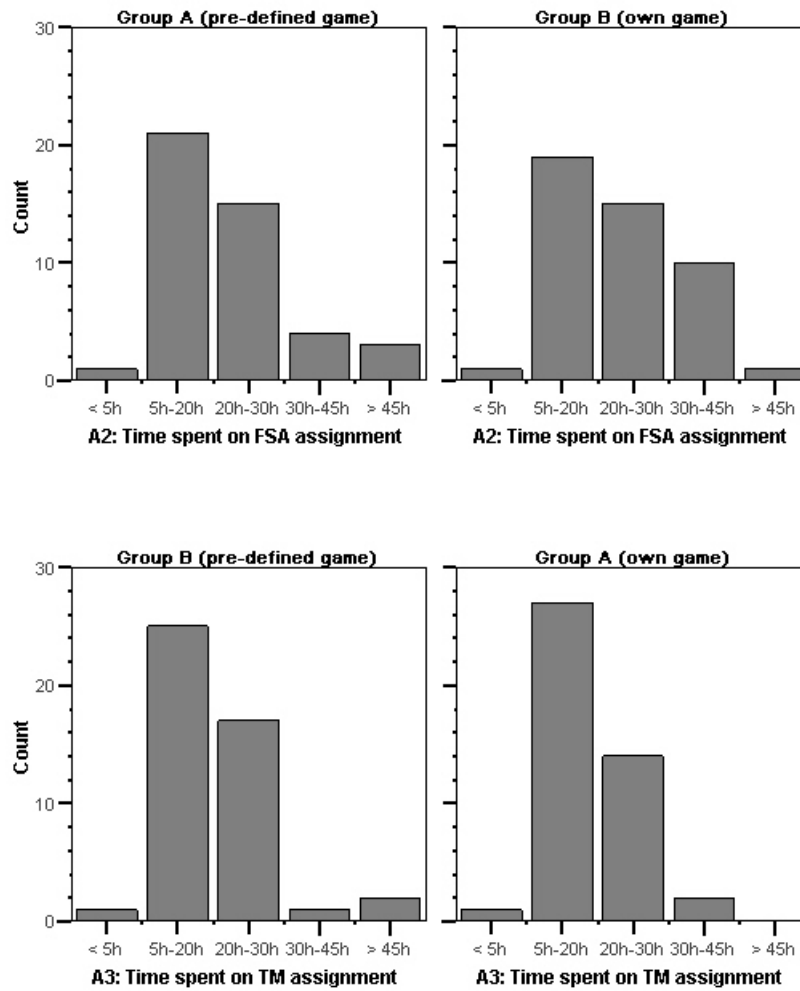


Figure 6.10: Time spent on assignments by conditions and tasks.

Table 6.3: Participants' performance across groups and tasks.

Group (version)			Statistic	Std. Error
FSA Marks	Group A (Pre-Defined Game)	Mean	53.62	2.39
		Std. Dev.	16.93	
	Group B (Own Game)	Mean	47.31	2.65
		Std. Dev.	19.08	
TM Marks	Group A (Own Game)	Mean	65.20	2.33
		Std. Dev.	15.79	
	Group B (Pre-Defined Game)	Mean	66.88	2.85
		Std. Dev.	19.74	

Initial Check Because time spent is measured on a 5-point scale with each point representing a range of at least five hours, calculating the amount of time spent on both tasks together results in the following scale:

1. 0–10
2. 5–25
3. 10–40
4. 25–50
5. 40–60
6. 50–75
7. 60–90
8. 75 and up
9. 90 and up

This scale cannot be used in any parametric or non-parametric test, because there is overlap between the categories, thus violating the assumption of a continuous distribution. So instead of comparing the time spent on the two tasks together, two Wilcoxon Rank-Sum (Mann-Whitney equivalent) tests will be carried out. The first test compared the average amount of time Group A spent on the task of Study 3a against the average amount of time Group B spent on the same task. Similarly, the second test compared the average amount of time Group A spent on the task of Study 3b against the average amount of time Group B spent on the same task. Carrying out two individual tests like this could increase the Type-I error rate, but since we are only looking to confirm the null-hypotheses it does not decrease the significance of our results. In fact, it actually increases it. Results of the individual Wilcoxon Rank-Sum tests, which were used because of the ordinal level of the dependent variable, follow.

In Study 3a, participants in group A, who were asked to write a pre-defined game, did not differ significantly in the amount of time they spend on their game-building task ($Mdn = 2.5$) from those in group B, who were asked to write their own game ($Mdn = 3.0$), $U = 930.00$, $p > .45$.

In Study 3b, participants in group A, who were asked to write their own game, did not differ significantly in the amount of time they spend on their game-building task ($Mdn = 2.0$) from those in group B, who were asked to write a pre-defined game ($Mdn = 2.0$), $U = 935.00$, $p > .45$.

6.6.2 Pre-Defined Game vs. Own Game: Performance

This section presents the results of a series of comparisons between the performance of participants in an own-game task and their performance in a pre-defined task.

Starting off with a t-test using experimental condition (pre-defined/own) as the within-subjects variable, the transition to a General Linear Model (GLM) is made in the analysis of Model 2 to accommodate the introduction of covariates. The first factor, which comprises two covariates, to be added to the GLM is time(2). Note that the cardinality of a factor indicates the number of covariates it comprises. Motivation(2,3,5) was added next. The three cardinalities of motivation(2,3,5) refer to three separate motivation measures. As described in Section 6.2.4, motivation(5) is composed of (1) mastery-approach, (2) mastery-avoidance, (3) performance-approach, (4) performance-avoidance, and (5) work-avoidance. Motivation(2) is composed of (1) avoidance and (2) approach. Motivation(3) is composed of (1) mastery-performance,

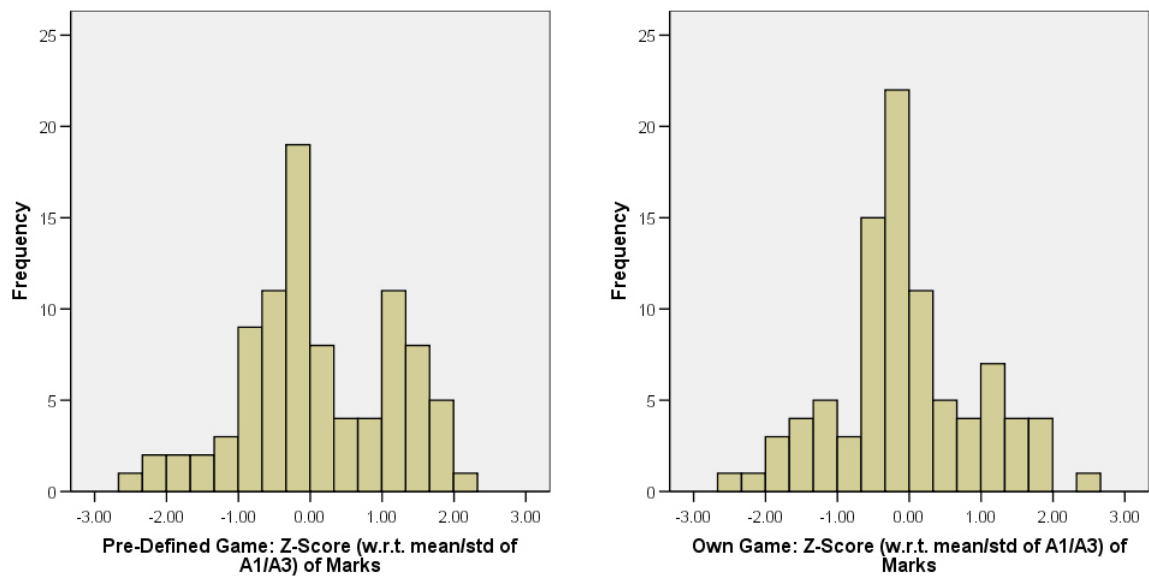


Figure 6.11: Comparison of normalized performance (i.e. Z-score) by condition.

(2) approach-avoidance, and (3) total motivation.

Each motivation measure has its own analysis and results in its own separate model. Finally, for Model 4 the between-subjects variable *gender* was added. All the analyses for Models 1–3 are performed as a repeated measures design and Model 4 as a mixed design.

6.6.2.1 Model 1: No Factors

Figure 6.11 shows the normalized data on the marks earned in the own game condition and the pre-defined condition. The marks were normalized with respect to the mean and standard deviation of the FSA assignment and the TM assignment, in order to obtain a participant's performance relative to that of their peers *on the same assignment*. In this and all subsequent performance analyses, these normalized data were used.

To compare the performance of the participants in the two conditions a dependent t-test was used. On average, participants received a higher (normalized) mark for a pre-defined game ($M = 0.11, SE = 0.11$) than for an own game ($M = -0.07, SE = 0.10$). This difference was not significant $t(89) = 1.34, p > .05$. Figure 6.12 shows the estimated parameters for Model 1.

Model	Dependent Variable (Performance on ...)	Parameter	95% CI for B			
			B	SE	Lower	Upper
1	Pre-Defined Task	Constant	0.111	0.110	-0.108	0.330
	Own Game Task	Constant	-0.720	0.103	-0.277	0.132
2	Pre-Defined Task	Constant	-0.701	0.405	-1.505	0.103
		Pre-Def: Time Spent	0.465**	0.139	0.189	0.741
	Own Game Task	Own-Game: Time Spent	-0.154	0.150	-0.454	0.145
		Constant	-0.872*	0.391	-1.650	-0.095
		Pre-Def: Time Spent	0.197	0.134	-0.070	0.464
		Own-Game: Time Spent	0.110	0.145	-0.179	0.399

* $p < .05$, ** $p < .01$.

Figure 6.12: Performance: Parameter Estimates for Models 1 and 2. Significant parameters are printed in **bold**.

6.6.2.2 Model 2: Time(2)

The factor time(2) comprised two separate variables: (a) the time spent on the assignment in the pre-defined condition and (2) the time spent on the assignment in the own-game condition. Using time(2) as a covariate in a repeated measures General Linear Model (GLM) comparing the performance of the participants in the two conditions produced the following results.

There is a main effect of time spent on a pre-defined game on performance, $F(1, 87) = 11.235, p < 0.01, r = .34$. The direction of the β -values in Figure 6.12 show that participants who spent more time on their pre-defined game tended to receive higher marks for their games. There is no such relationship between the time spent on an own-game and the marks received for the games, neither are there any interaction effects.

6.6.2.3 Model 3: Time(2) and Motivation(2,3,5)

In Section 6.2.4 three different ways of extracting motivation information from the AGQ-data were described. This section presents separate results for each of them.

Model 3a: Time(2) and Motivation(5) The factor motivation(5) comprised five separate variables: (1) mastery-approach, (2) performance-approach, (3) mastery-avoidance, (4) performance-avoidance, and (5) work-avoidance. Using time(2) and motivation(5) as covariates in a repeated measures GLM comparing the performance of the partici-

Model	Dependent Variable (Performance on ...)	Parameter	95% CI for B			
			B	SE	Lower	Upper
3a	Pre-Defined Task	Constant	-1.318	0.877	-3.064	0.428
		Pre-Def: Time Spent	0.439**	0.142	0.156	0.722
		Own-Game: Time Spent	-0.157	0.161	-0.478	0.164
		Mastery Avoidance	0.076*	0.034	0.008	0.145
		Mastery Approach	0.010	0.035	-0.059	0.080
		Performance Approach	0.015	0.030	-0.045	0.074
		Performance Avoidance	-0.033	0.030	-0.093	0.027
		Work Avoidance	-0.012	0.028	-0.068	0.044
	Own Game Task	Constant	-0.675	0.819	-2.305	0.955
		Pre-Def: Time Spent	0.170	0.133	-0.094	0.434
		Own-Game: Time Spent	0.228	0.150	-0.072	0.527
		Mastery Avoidance	-0.004	0.032	-0.068	0.059
		Mastery Approach	-0.022	0.033	-0.067	0.063
		Performance Approach	0.039*	0.028	-0.017	0.095
		Performance Avoidance	-0.063	0.028	-0.119	0.007
		Work Avoidance	-0.005	0.026	-0.057	0.047
3b	Pre-Defined Task	Constant	-1.421*	0.618	-2.560	-0.192
		Pre-Def: Time Spent	0.470**	0.139	0.195	0.746
		Own-Game: Time Spent	-0.169	0.150	-0.467	0.130
		Approach	0.015	0.018	-0.021	0.051
		Avoidance	<i>0.015</i>	0.016	-0.018	0.047
	Own Game Task	Constant	-0.636	0.588	-1.806	0.533
		Pre-Def: Time Spent	0.206	0.132	-0.056	0.468
		Own-Game: Time Spent	0.118	0.143	-0.166	0.402
		Approach	0.021	0.017	-0.014	0.055
		Avoidance	-0.036*	0.016	-0.067	-0.005
3c	Pre-Defined Task	Constant	-1.631*	0.628	-2.879	-0.382
		Pre-Def: Time Spent	0.475**	0.137	0.202	0.749
		Own-Game: Time Spent	-0.202	0.151	-0.502	0.098
		Approach-Avoidance	0.003	0.015	-0.026	0.032
		Mastery-Performance	-0.025	0.016	-0.057	0.007
		Motivation Total	<i>0.019</i>	0.010	0.000	0.038
	Own Game Task	Constant	-0.687	0.605	-1.891	0.517
		Pre-Def: Time Spent	0.207	0.133	-0.056	0.471
		Own-Game: Time Spent	0.110	0.145	-0.179	0.399
		Approach-Avoidance	-0.027	0.014	-0.055	0.000
		Mastery-Performance	-0.006	0.015	-0.037	0.025
		Motivation Total	<i>-0.007</i>	0.009	-0.025	0.012

* $p < .05$, ** $p < .01$.

Figure 6.13: Performance: Parameter Estimates for Models 3a–c. Significant parameters are printed in **bold**. Parameters for which the overall ANCOVA showed a significant interaction effect with the condition, i.e. pre-defined or own game, are printed in *italics*.

pants in the two conditions produced the following results.

There is a main effect of performance avoidance on performance, $F(1, 80) = 4.800, p < 0.05, r = .24$. The direction of the β -values in Figure 6.13 show that participants who scored low on performance avoidance tended to receive higher marks for their own-game than they did for their pre-defined game.

Similar to Model 2, there is a main effect of time spent on a pre-defined game on performance, $F(1, 80) = 8.590, p < 0.01, r = .31$. The direction of the β -values in Figure 6.13 show that participants who spend more time on their pre-defined game tend to receive higher marks for their games. There is no such relationship between the time spent on an own-game and the marks received for their games, neither are there any interaction effects.

Model 3b: Time(2) and Motivation(2) The factor motivation(2) comprised two separate variables: (1) approach, and (2) avoidance. Using time(2) and motivation(2) as covariates in a repeated measures GLM comparing the performance of the participants in the two conditions produced the following results.

Similar to Models 2 and 3a, there is a main effect of time spent on a pre-defined game on performance, $F(1, 85) = 10.858, p < 0.01, r = .34$. The directions of the β -values in Figure 6.13 show that participants who spend more time on their pre-defined game tend to receive higher marks for their games. There is no such relationship between the time spent on an own-game and the marks received for the games.

There was a significant interaction effect between the experimental condition and the participant's score on avoidance, $F(1, 85) = 5.797, p < 0.05, r = .25$. The directions of the β -values in Figure 6.13 show that there is a significant *negative* effect of avoidance on performance on an own-game task, but a non-significant *positive* effect on performance on a pre-defined task.

Model 3c: Time(2) and Motivation(3) The factor motivation(3) comprised three separate variables: (1) approach-avoidance, (2) mastery-performance, and (3) motivation size. Using time(2) and motivation(3) as covariates in a repeated measures GLM comparing the performance of the participants in the two conditions produced the following results.

Similar to Models 2, 3a–b, there is a main effect of time spent on a pre-defined game on performance, $F(1, 84) = 11.907, p < 0.01, r = .35$. The directions of the β -values in Figure 6.13 show that participants who spend more time on their pre-defined

game tend to receive higher marks for their games. There is no such relationship between the time spent on an own-game and the marks received for the games.

There was a significant interaction effect between the experimental condition and the participant's motivation size, $F(1, 84) = 4.197, p < 0.05, r = .22$. The directions of the β -values in Figure 6.13 show that there is a near-significant ³ *positive* effect of motivation size on performance on a pre-defined task, but a non-significant ⁴ *negative* effect on performance on an own-game task.

6.6.2.4 Model 4: Motivation(3), Time(2), and Gender

Using time(2) and one of motivation(2,3,5) as covariates and gender as a between-subjects variable in a repeated measures GLM comparing the performance of the participants in the two conditions, produced no new meaningful results compared to Models 3a–c.

Furthermore, the assumption of homogeneity of regression slopes could not be met, because the between-subjects interaction effect between *gender* and the first component of *time(2)* is significant ($F(1, 79) = 5.61, p < .01$). This failure to meet the assumption of homogeneity of regression slopes raises serious concerns about the main analysis.

For this reason, and because the addition of *gender(1)* as a predictor did not change any of the results compared to the previous analysis, it was decided to remove *gender(1)* as a predictor. This makes Models 3a–c our final models.

Because Model 4 was discarded, its estimated parameters are not included. They can be found in Appendix K instead.

6.6.3 Pre-Defined Game vs. Own Game: Enjoyment

This section presents the results of a series of comparisons between the participants' enjoyment of an own-game task and their enjoyment of a pre-defined task.

Starting off with a t-test using the experimental condition (pre-defined/own) as the within-subjects variable, the transition to a General Linear Model (GLM) is made in the analysis of Model 2 to accommodate the introduction of covariates. The first factor to be added to the GLM is motivation(2,3,5). As in Section 6.6.2, the three cardinalities of motivation(2,3,5) refer to three separate motivation measures; each of which has its

³Confidence lies between 90% and 95%.

⁴Confidence is less than 90%.

own analysis resulting in its own separate model. Finally, for Model 3 the between-subjects variable *gender* was added.

With regard to the normality of the enjoyment data, we found that although the Kolmogorov-Smirnov test gives significant results for both the own game ($D(91) = 0.16, p < .05$) and pre-defined game conditions ($D(91) = 0.16, p < .05$), neither kurtosis nor skewness of either condition shows a significant difference at $p < .01$. Furthermore, the ANCOVA at the heart of the GLM is a very robust test which, given enough degrees of freedom, has been proven to give reliable results regardless of conditions. This includes situations in which the assumption of normality is not met, such as (Lunney, 1970). The reason our data does not meet the normality requirement is not even so much a matter of distribution, but the fact that it is measured on a very coarse scale. For these reasons the ANCOVA should still be able to provide reliable results.

The analyses for Models 1 and 2 are performed as a repeated measures design and Model 3 as a mixed design.

6.6.3.1 Model 1: No Factors

Figure 6.7 and Table 6.2 show the raw data on the enjoyment of the own game condition and the pre-defined condition. To compare the enjoyment of the participants in the two conditions a dependent t-test was used.

On average, participants did not enjoy any one condition significantly more than the other. Figure 6.14 shows the estimated parameters for Model 1.

6.6.3.2 Model 2: Motivation(2,3,5)

In Section 6.2.4 three different ways of extracting motivation information from the AGQ-data were described. This section presents separate results for each of them.

Model 2a: Motivation(5) The factor motivation(5) comprised five separate variables: (1) mastery-approach, (2) performance-approach, (3) mastery-avoidance, (4) performance-avoidance, and (5) work-avoidance. Using time(2) and motivation(5) as covariates in a repeated measures GLM comparing the participants' enjoyment of the two conditions produced no significant results.

No significant main effects or interaction effects were found. Figure 6.14 shows the estimated parameters for Model 2.

Model	Dependent Variable (<i>Enjoyment on . . .</i>)	Parameter	B	SE	95% CI for B	
					Lower	Upper
1	Pre-Defined Task	Constant	3.567	0.188	3.192	3.941
	Own Game Task	Constant	3.400	0.166	3.070	3.730
2a	Pre-Defined Task	Constant	2.262	1.389	-0.500	5.025
		Mastery Avoidance	0.027	0.062	-0.095	0.150
		Mastery Approach	0.054	0.061	-0.068	0.177
		Performance Approach	0.036	0.053	-0.070	0.141
		Performance Avoidance	0.015	0.054	-0.091	0.122
		Work Avoidance	-0.032	0.050	-0.132	0.069
	Own Game Task	Constant	3.543**	1.291	1.107	5.980
		Mastery Avoidance	-0.073	0.054	-0.181	0.035
		Mastery Approach	0.014	0.054	-0.093	0.122
		Performance Approach	-0.049	0.047	-0.143	0.044
		Performance Avoidance	0.094	0.047	0.000	0.188
		Work Avoidance	0.003	0.044	-0.085	0.091
2b	Pre-Defined Task	Constant	1.745*	0.869	0.017	3.473
		Approach	0.055	0.032	-0.009	0.118
		Avoidance	0.015	0.029	-0.043	0.073
	Own Game Task	Constant	3.529**	0.786	1.966	5.092
		Approach	-0.016	0.029	-0.074	0.041
		Avoidance	0.013	0.026	-0.039	0.065
2c	Pre-Defined Task	Constant	1.630	0.930	-0.220	3.479
		Approach-Avoidance	-0.019	0.026	-0.070	0.033
		Mastery-Performance	-0.010	0.029	-0.067	0.046
		Motivation Total	0.036*	0.017	0.002	0.071
	Own Game Task	Constant	3.750**	0.839	2.081	5.419
		Approach-Avoidance	0.012	0.024	-0.035	0.059
		Mastery-Performance	0.020	0.026	-0.031	0.071
		Motivation Total	-0.005	0.016	-0.036	0.026

* $p < .05$, ** $p < .01$.

Figure 6.14: Enjoyment: Parameter Estimates for Models 1 and 2a–c. Significant parameters are printed in **bold**. Parameters for which the overall ANCOVA showed a significant interaction effect with the condition, i.e. pre-defined or own game, are printed in *italics*.

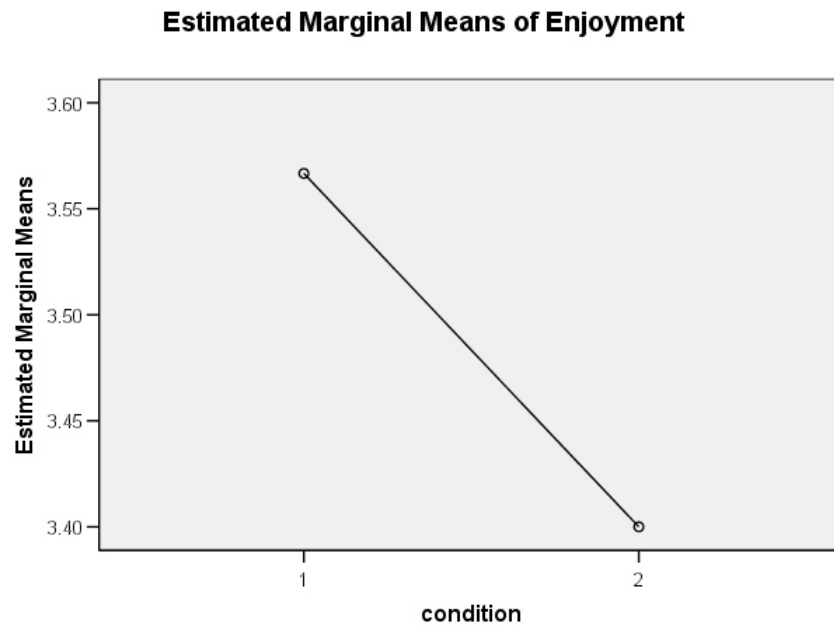


Figure 6.15: Enjoyment of Conditions – controlled for motivation(3). Condition 1 is the pre-defined task and Condition 2 is the own game task.

Model 2b: Motivation(2) The factor motivation(2) comprised two separate variables: (1) approach, and (2) avoidance. Using time(2) and motivation(2) as covariates in a repeated measures GLM comparing the participants' enjoyment of the two conditions produced the following results.

There was a significant interaction effect between the experimental condition and the participant's approach score, $F(1, 87) = 4.199, p < 0.05, r = .21$. The directions of the β -values in Figure 6.14 show that there is a near-significant⁵ positive effect of approach score on enjoyment of a pre-defined task, but a non-significant⁶ negative effect on enjoyment of an own-game task.

Model 2c: Motivation(3) The factor motivation(3) comprised three separate variables: (1) approach-avoidance, (2) mastery-performance, and (3) motivation size. Using time(2) and motivation(3) as covariates in a repeated measures GLM comparing the participants' enjoyment of the two conditions produced the following results.

There was a significant main effect of condition, $F(1, 86) = 4.476, p < 0.05, r =$

⁵Confidence lies between 90% and 95%.

⁶Confidence is less than 90%.

.22. The profile plot in Figure 6.15 shows that participants generally enjoyed a pre-defined task better ($EMM = 3.57$) than an own-game task ($EMM = 3.40$) where EMM is the estimated marginal mean.

There was also a significant interaction effect between condition and motivation size, $F(1, 86) = 4.897, p > 0.05, r = .23$. The directions of the β -values in Figure 6.14 show that there is a significant *positive* effect of motivation size on enjoyment of a pre-defined task, but a non-significant *negative* effect on enjoyment of an own-game task.

6.6.3.3 Model 3: Motivation(2,3,5) and Gender

Using motivation(2,3,5) as covariates and gender as a between-subjects variable in a repeated measures GLM comparing the performance of the participants in the two conditions, produced the following results.

Model 3a: Motivation(5) and Gender No significant main effects or interaction effects were found using motivation(5). Figure 6.16 shows the estimated parameters for Model 3a.

Model 3b: Motivation(2) and Gender There was a significant main effect of condition, $F(1, 86) = 4.301, p < 0.05, r = .22$. The profile plot in Figure 6.17 shows that participants generally enjoyed an own-game task better ($EMM = 3.53$) than a pre-defined task ($EMM = 3.47$) where EMM is the estimated marginal mean.

Furthermore, there was a significant interaction effect between approach score and condition, $F(1, 86) = 4.206, p < 0.05, r = .22$. The directions of the β -values in Figure 6.16 show that there is a near-significant ⁷ *positive* effect of approach score on enjoyment of a pre-defined task, but a non-significant *negative* effect on enjoyment of an own-game task.

Finally, there was a non-significant interaction effect between condition and gender, $F(1, 86) = 1.49, p > .05, r = .13$. The profile plot in Figure 6.18 shows that there is a tendency for female participants to enjoy the own-game task better than male participants. However, there was also a tendency for male participants to enjoy the pre-defined task better than female participants.

⁷Confidence lies between 90% and 95%.

Model	Dependent Variable		B	SE	95% CI for B	
	Parameter	Lower			Upper	
3a	C1: Enjoyment	Constant	2.255	1.394	-0.518	5.028
		Mastery Avoidance	0.026	0.062	-0.097	0.149
		Mastery Approach	0.064	0.063	-0.062	0.190
		Performance Approach	0.027	0.055	-0.082	0.136
		Performance Avoidance	0.016	0.054	-0.091	0.124
		Work Avoidance	-0.030	0.051	-0.131	0.071
		Gender = Female	-0.373	0.567	-1.502	0.756
		Gender = Male	<i>n/a^a</i>	.	.	.
	C2: Enjoyment	Constant	3.552**	1.226	1.112	5.992
		Mastery Avoidance	-0.072	0.054	-0.180	0.036
		Mastery Approach	0.003	0.056	-0.107	0.114
		Performance Approach	-0.040	0.048	-0.136	0.056
		Performance Avoidance	0.093	0.047	-0.002	0.187
		Work Avoidance	0.001	0.045	-0.087	0.090
		Gender = Female	0.440	0.499	-0.554	1.433
		Gender = Male	<i>n/a^a</i>	.	.	.
3b	C1: Enjoyment	Constant	1.810*	0.880	0.061	3.560
		Approach	0.055	0.032	-0.009	0.118
		Avoidance	0.014	0.029	-0.044	0.072
		Gender = Female	-0.295	0.515	-1.318	0.728
		Gender = Male	<i>n/a^a</i>	.	.	.
	C2: Enjoyment	Constant	3.445**	0.795	1.865	5.024
		Approach	-0.016	0.029	-0.074	0.041
		Avoidance	0.014	0.026	-0.038	0.067
		Gender = Female	0.379	0.465	-0.544	1.303
		Gender = Male	<i>n/a^a</i>	.	.	.
3c	C1: Enjoyment	Constant	1.665	0.935	-0.194	3.524
		Approach-Avoidance	-0.018	0.026	-0.070	0.034
		Mastery-Performance	-0.014	0.029	-0.072	0.044
		Motivation Total	0.037*	0.017	0.002	0.071
		Gender = Female	-0.343	0.527	-1.390	0.704
		Gender = Male	<i>n/a^a</i>	.	.	.
	C2: Enjoyment	Constant	3.702**	0.841	2.030	5.375
		Approach-Avoidance	0.012	0.024	-0.035	0.059
		Mastery-Performance	0.025	0.026	-0.027	0.077
		Motivation Total	-0.005	0.016	-0.036	0.026
		Gender = Female	0.465	0.474	-0.477	1.407
		Gender = Male	<i>n/a^a</i>	.	.	.

* $p < .05$, ** $p < .01$.

Figure 6.16: Enjoyment: Parameter Estimates for Models 3a–c. Significant parameters are printed in **bold**. Parameters for which the overall ANCOVA showed a significant interaction effect with the condition, i.e. pre-defined or own game, are printed in *italics*.

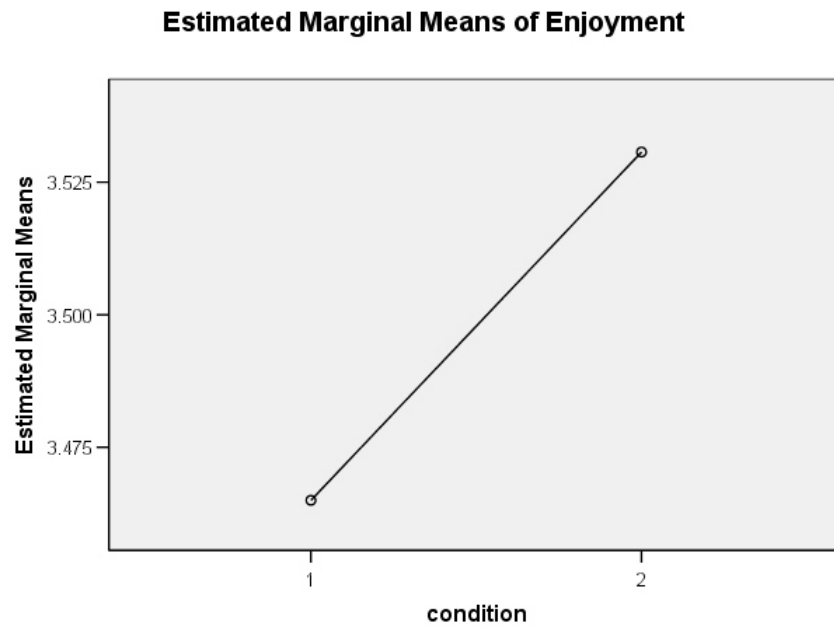


Figure 6.17: Enjoyment of Conditions – controlled for motivation(2) and gender. Condition 1 is the pre-defined task and Condition 2 is the own game task.

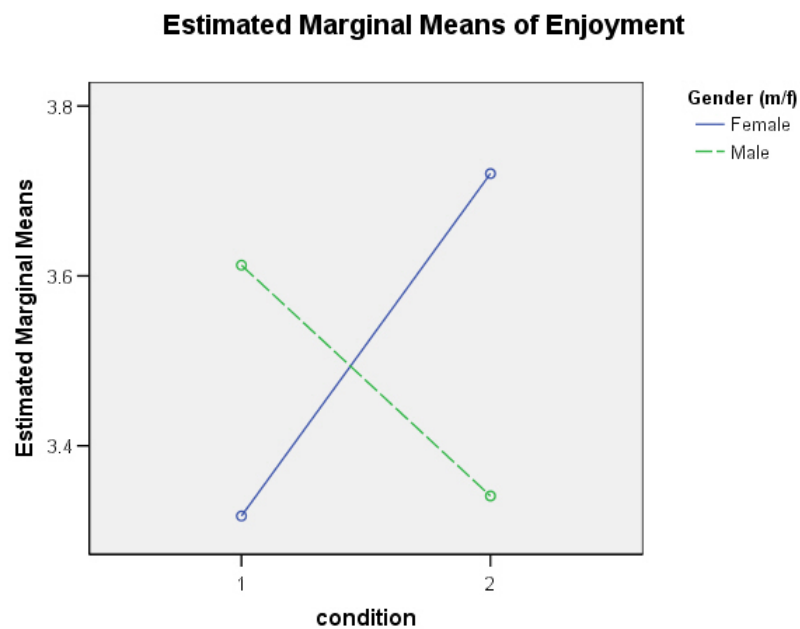


Figure 6.18: Enjoyment of Conditions by Gender – controlled for motivation(2). Condition 1 is the pre-defined task and Condition 2 is the own game task.

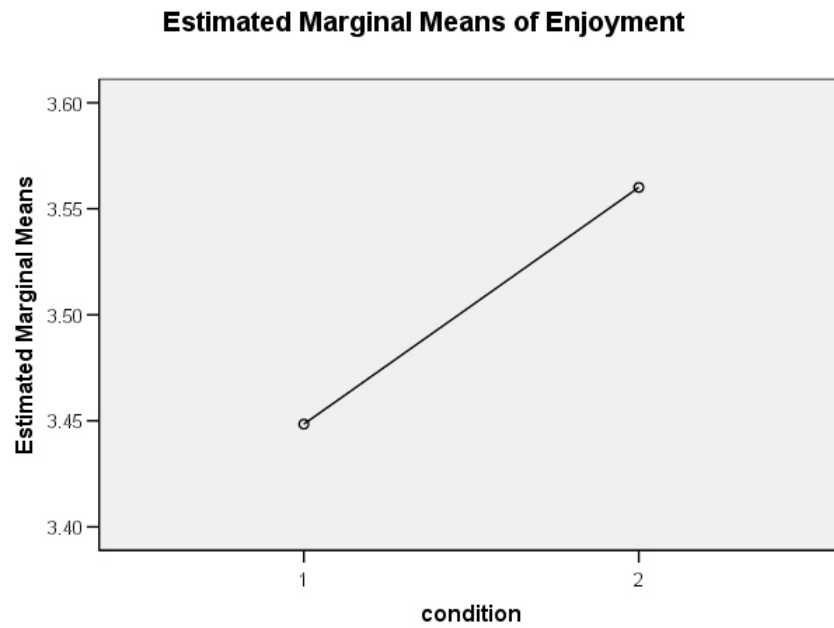


Figure 6.19: Enjoyment of Conditions – controlled for motivation(3) and gender. Condition 1 is the pre-defined task and Condition 2 is the own game task.

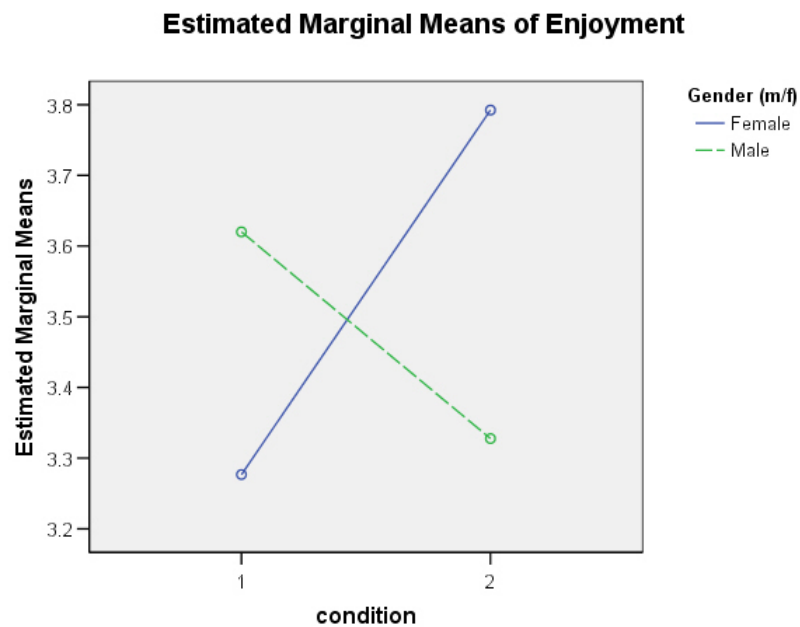


Figure 6.20: Enjoyment of Conditions by Gender – controlled for motivation(3). Condition 1 is the pre-defined task and Condition 2 is the own game task.

Model 3c: Motivation(3) and Gender There was a significant main effect of condition, $F(1, 85) = 5.72, p < .05, r = .25$. The profile plot in Figure 6.19 shows that participants generally enjoyed an own-game task better ($EMM = 3.56$) than a pre-defined task ($EMM = 3.45$) where EMM is the estimated marginal mean. Note that this effect is the exact opposite of the one we found in our previous analysis with motivation(3), before controlling for the effect of gender.

There was also an interaction-effect between condition and motivation size, $F(1, 85) = 5.02, p < .05, r = .24$. The directions of the β -values in Figure 6.16 show that there is a significant *positive* effect of motivation size on enjoyment of a pre-defined task, but a non-significant *negative* effect on enjoyment of an own-game task.

Finally, there was a non-significant interaction effect between condition and gender, $F(1, 85) = 2.07, p > .05, r = .15$. The profile plot in Figure 6.20 shows that there is a tendency for female participants to enjoy the own-game task better than male participants. However, there was also a tendency for male participants to enjoy the pre-defined task better than female participants.

6.7 Discussion

The results of Study 3 confirm some of the hypotheses posed at the beginning of the chapter, although some results also contradicted the predictions. We recall the hypotheses specified in Section 6.4 and discuss each one in turn.

6.7.1 Pre-Defined Game vs. Own Game: Performance

1. When participating in the own-game condition, participants will achieve a performance (measured using marks out of 100) equal to their performance in the pre-defined condition.
2. When participating in the own-game condition, participants will achieve a higher performance (measured using marks out of 100) compared to the pre-defined condition, after controlling for *the amount of time spent on a game-building task* (measured on a 5-interval scale) and a participant's *motivational type* (as measured by the AGQ).

No support was found for the main effect of condition predicted in Hypothesis 1, either before, or after controlling for time and motivational type. The fact that the tasks

were carried out as assessed assignments, may have had a unfavourable influence on motivation or performance. More likely though, our results can be seen as additional evidence that increasing personalisation and choice does not necessarily result in increased overall performance, adding to work like that of Lepper and Malone (1987); Parker and Lepper (1992).

3. Participants who score highly on avoidance will achieve a higher performance (measured using marks out of 100) in the pre-defined condition than in the own-game condition.

There was an interaction effect between avoidance and condition, where participants with high avoidance performed better than those with low avoidance on the pre-defined task, while participants with low avoidance performed better than those with high avoidance on the own-game task. This interaction effect provides evidence in support of Hypothesis 3. As mentioned in Section 6.4, the motivation of participants with a high avoidance score is based on a fear of failure. This fear of failure may have caused these participants to feel overwhelmed by the freedom and design-aspect of an own-game task, compared to the safety of a pre-defined task, thus limiting their performance.

4. Participants who score highly on approach will achieve a higher performance (measured using marks out of 100) in the own-game condition than in the pre-defined condition.

No support was found for an interaction effect between approach and condition. The reason for this discrepancy between the hypotheses and the results is hard to explain. It is possible that the own-game task does not incite additional motivation in participants with high approach. In much the same way, it is possible that it does increase motivation, but this motivation does not translate to increased performance. Other, external factors may have been of influence as well. For example, the fact that the tasks were carried out as assessed assignments, may have limited the potential motivation increase. This is a well-known phenomenon (Deci *et al.*, 1999). Further research is required to determine the influence(s) at work here.

5. Participants with a low motivation score will achieve a higher performance (measured using marks out of 100) in the own-game condition than in the pre-defined condition.

There was an interaction effect between motivation size and condition, where participants with a high motivation score performed better than those with a low motivation score in the pre-defined condition, while participants with a low motivation score performed better than those with a high motivation score in the own-game condition.

While the part of the result dealing with poorly motivated students is as expected, the part of the result dealing with highly motivated students is not. We hypothesized that there would be no significant difference in performance for highly motivated participants in the own-game condition compared to the pre-defined condition. However, the results show that highly motivated participants enjoyed the own-game condition less than the pre-defined condition. The only explanation that can be offered here is that students who are performing well in the current educational context, and would therefore likely have a high overall motivation score, can be resistant to innovations (Squire, 2004). It is possible this resistance is reflected in their performance. Alternatively, the participants' lower competence – as judged by their lower performance – may be the source of this resistance.

6. A participant's gender is not related to their performance (measured using marks out of 100) by condition.

It is additionally noted that no support was found for an effect of gender on performance, which is in line with the predictions made in the hypotheses. Furthermore, while the time spent on the pre-defined task was found to have an effect on the performance in that task, the same was not true for the time spent on the own-game task. At first, this finding seems highly surprising. It can mean one of two things: (1) at least some participants put time into an own game which is not reflected in their performance, or (2) at least some participants are able to perform well in the own-game condition, without investing a proportionate amount of time into the task. Whilst the former initially seems the more likely proposition, we would expect an effect like this to show up in a comparison between the time spent on an own-game task and that spent on a pre-defined task, but it does not. In fact there are slightly fewer people spending more than 45 hours on an own-game task than there are on a pre-defined task. Unfortunately, the evidence of the comparison is not strong enough to support the second proposition either. Further research would be needed to determine which of the factors is of influence.

Finally, not predicted by the hypotheses, but not very surprisingly, a main effect of performance avoidance on performance was found, with participants with high perfor-

mance avoidance generally scoring lower than those with low performance avoidance. This is in fact one of the characteristics associated with people scoring highly on performance avoidance.

6.7.2 Pre-Defined Game vs. Own Game: Enjoyment

1. When participating in the own-game condition, participants will experience an increased sense of enjoyment (measured on a 7-point Likert-scale), compared to the pre-defined condition.
2. The main effect mentioned in Hypothesis 1 will be stronger when after controlling for motivational type (as measured by the AGQ) and gender.

After controlling for gender and motivational type, there was a significant main effect of condition on enjoyment, where participants generally enjoyed the own-game condition more than the pre-defined condition. This confirms Hypothesis 2.

However, before controlling for gender and motivation, no significant main effect of condition on enjoyment was found. This is taken to be evidence against the claim that, by definition, the introduction of personalisation and choice increases overall enjoyment of a task, at least in the context of assessed assignments in the undergraduate theoretical computer science curriculum.

In fact, after controlling the results for motivational type only, a significant effect was found which was the exact opposite of that predicted in Hypothesis 1. That is, participants generally enjoyed the pre-defined condition more than the own-game condition. This result is more difficult to interpret, but the interaction effect between condition and gender, as discussed below, may shed some light on this issue.

3. When participating in the pre-defined condition, participants who score highly on avoidance will experience an increased sense of enjoyment (measured on a 7-point Likert-scale) compared to the own-game condition.

There was an interaction effect between avoidance and condition, where participants with a high avoidance score enjoyed the pre-defined condition more than those with a low avoidance score, while those with a low avoidance score enjoyed the own-game condition more than those with a high avoidance score. This interaction effect confirms Hypothesis 3.

The three attributes which are consistently associated with avoidance goals are: (1) fear of failure, (2) disorganisation, and (3) lack of self-determination. Therefore,

people with a high avoidance score may have felt overwhelmed by the freedom and design-aspect of an own-game task, compared to the safety of a pre-defined task. This could have caused them to enjoy the own-game condition less.

4. When participating in the own-game condition, participants who score highly on approach will experience an increased sense of enjoyment (measured on a 7-point Likert-scale) compared to the pre-defined condition.

For enjoyment, one difference between the own-game condition and the pre-defined condition was expressed through an interaction effect between approach and condition, where participants with high approach enjoyed the pre-defined task more than the own-game task, those with low approach scores enjoyed the own-game task more than the pre-defined task. This is exactly the opposite of our prediction expressed in the hypotheses. Although this result is unexpected, it possible that the same effect as was observed by Squire (2004), mentioned in Section 6.4, is at work here. Participants who are in their element in a traditional educational context would generally be more approach motivated than those who are not reaching their full potential. Squire observes that the former are much more resistant to innovations in the curriculum than the latter. This could be reflected in their enjoyment of the own-game building task for those students who do well in a traditional formal education context.

5. When participating in the own-game condition, participants with a low motivation score will experience an increased sense of enjoyment (measured on a 7-point Likert-scale) compared to the pre-defined condition.

There was an interaction effect between motivation size and condition, where highly motivated participants enjoyed the pre-defined condition more than those who were poorly motivated, while poorly motivated participants enjoyed the own-game condition more than those who were highly motivated. This interaction effect confirms Hypothesis 5.

While the part of the result dealing with poorly motivated students is as expected, the part of the result dealing with highly motivated students is not. We hypothesized that there would be no significant difference in enjoyment for highly motivated participants in the own-game condition compared to the pre-defined condition. However, the results show that highly motivated participants enjoyed the own-game condition less than the pre-defined condition. The only explanation that can be offered here is the same one that was quoted for the previous hypothesis: students who are performing

well in the current educational context, and would therefore likely have a high overall motivation score, can be resistant to innovations.

6. A participant's gender is related to their relative sense of enjoyment (measured on a 7-point Likert-scale) of the two conditions (i.e. own-game and pre-defined).

When controlling for motivation and gender, a non-significant interaction effect between gender and condition was found, where there was a tendency for female participants to enjoy the own-game task better than their male counterparts. In turn, male participants tended to enjoy the pre-defined task better than their female counterparts. Although the effect is non-significant and therefore likely due to chance, we do consider it here because it is also possible that the non-significance is due to the fact that participants were predominantly male. Out of the 90 participants, only 14 were female; 76 were male.

However, even though the effect is not significant, it can still help explain why there was a main effect of condition in the opposite direction from what the hypotheses predicted in the absence of the control variable gender. After all, male participants tended to like the own-game task less and there are a lot of them compared to the number of female participants. This could easily tip the effect towards that of the male participants, because there are not enough data-points to support that of their female counterparts.

Female students may feel less affinity than their male counterparts with a games-industry which has traditionally focussed on boys Cassell and Jenkins (1998). This could explain their lower enjoyment of the pre-defined task. The higher enjoyment of the own-game task is more surprising, but can possibly be explained by the fact that the task gave female participants a rare opportunity to make their the highly-male-oriented informatics undergraduate environment more female oriented, The fact that this opportunity is utilized is demonstrated by the games created by female participants in the current study. For example, a goal in one of these games involves growing flowers, while another game is set around the TV-show 'The X-Factor', which caters for a predominantly female audience.

One last factor which might have been of influence is the fact the majority of video games are played by male gamers (Greefield *et al.*, 1994; Provenzo, 1991). This could cause them to be disillusioned by the game-building task which does not match up to their current game-standards. While they are happy enough use the 'sub-optimal system' to copy a pre-defined game, they might get frustrated during the own-game

task because they can never write a game which is as good as what they play at home. Female participants are less likely to have the same expectations of the game-building task, because of their more limited experience with video games. Although it is possible that female informatics students, like those who participated in our studies, could be an exception to this ‘rule’ and have plenty of experience with video games.

Recalling the question at the centre of this study – whether or not an ‘own-game’ scenario differs from a ‘pre-defined’ scenario in its effect on performance and/or enjoyment – the results provide evidence in favour of a positive answer to both these questions.

6.7.3 Validity

In both the design and the execution of this study, every reasonable effort was made to ensure the validity of our results. However, the measures of *performance* and *motivation* do raise some questions with respect to construct validity.

The **performance measure** used a participant’s performance on the complete assignment. This means their marks are composed of three parts: (1) the marks for the introduction to the game-engine which all participants are expected to be able to complete without difficulty account for 40%, (2) marks on the game itself account for 30%, and (3) marks on a set of theory questions about the same modelling skill as the game-building task. These last marks account for another 30% of the total mark. So even though performance does include marks on theory questions, the marks are still somewhat biased towards the game-building marks and tell us more about how a participant did on the game-building task and introduction, than about how they did on other questions about the same modelling skill.

In order to address this issue, an attempt was made to run the analyses using just the 30% of the marks that represent the performance on the third part of the assignment only. Unfortunately though, this data was not normally distributed in the case of the first assignment. This means that ‘performance’ as used in this study (a) is a slightly more complex measure than we ideally would have liked to use and (b) should therefore be interpreted as combination of performance on the game-building task itself and performance on a number of non-game-related modelling questions.

As discussed in Chapter 5, the **motivational framework** of achievement goals and the accompanying questionnaire (AGQ) do not give a complete motivational profile. As such, the framework reveals only a fraction of the full complexity of a person’s

motivation and focusses mainly on a person's fears and ambitions using a fairly crude set of distinctive classes of achievement goals. Different motivational frameworks focus on different aspects of motivation and would have possibly shone an entirely different light on our performance and enjoyment data.

Furthermore, both Performance Hypothesis 5 and Enjoyment Hypothesis 5 use a concept of *high/low motivation* which is distilled from the achievement goal data. As mentioned above, the achievement goal framework does not give a complete motivational profile and is thus not able to produce a general concept of *high/low motivation*. Therefore, the concept of *high/low motivation* used in the hypotheses, and throughout the rest of this thesis, can only be interpreted to in the context of the achievement goal framework.

Finally, there was a validity issue with regard to **task equivalence** between the own-game and pre-defined scenario in Study 2 described in Chapter 5. This same issue is also present in Study 3: the pre-defined task is not a minimal implementation of the list of requirements provided in the assignment. However, as with Study 2, every effort was made to make the pre-defined game a minimal implementation of the requirements, *while still being an actual game rather than an incoherent collection of interactive objects*. Furthermore, for all of the 'additional' programming involved in the pre-defined game it was possible to recycle code from essential parts of the game. In other words, no new concepts were used on top of the minimal implementation of the set of requirements.

6.7.4 Ethical Considerations

Many of the same ethical concerns we came across in Chapter 5 are equally valid in this study. Section 5.4.5.1 provides an overview of the ethical concerns of Study 2. However, Study 3 does differ significantly from Study 2 with respect to the ethical importance of task-equivalence.

First of all, in Study 2 participants could choose the their preferred task, which addresses – at least in part – the issue of task equivalence. In Study 3 on the other hand, participants are assigned to one task or the other, thus potentially creating an ethical concern. However, this study was conducted as counter-balanced design, which also averts the issue of task equivalence albeit in a different manner.

In the counter-balanced design, all participants are exposed to both the own-game and the pre-defined game. Half of them complete the own-game task first and the other

half complete the pre-defined task first. An initial check was also performed before the main analyses to ensure that on average, those participants who completed the own-game task first performed equally well as those who completed the pre-defined task first. The initial check showed no significant difference between the two groups (Section 6.6.1.2).

6.8 Summary

This chapter showed that certain participants perform better on an own-game task, than they do on a pre-defined task. Evidence of the opposite was also found: certain participants perform better on a pre-defined task, than they do on an own-game task. Both avoidance score and motivation size were found to be useful discriminators in this respect.

This chapter also showed that certain participants enjoy an own-game task better than a pre-defined task. Evidence of the opposite was also found: certain participants enjoy a pre-defined task better than an own-game task. Both approach score and motivation size were found to be useful discriminators in this respect.

Finally, the chapter provided evidence against the claim that the introduction of personalisation and choice lead to an increased overall performance. Furthermore, it was shown that personalisation and choice did not necessarily increase overall enjoyment. When the results were controlled for gender and motivation, a significant main effect of condition did show up with the own-game task being more enjoyable than the pre-defined one.

The next chapter concludes this thesis by reflecting on its findings, considering possible implications for TCS education, and suggesting directions for future research.

Chapter 7

Conclusions

In this thesis a novel approach to theoretical computer science (TCS) education has been presented and evaluated. In this final chapter, we will summarize the primary contributions of the thesis and revisit the thesis questions posed in Chapter 1 to determine to what extent they have been answered. Finally, some suggestions for future work are made, followed by the overall conclusions.

7.1 Contributions and Findings

The **contributions** of this thesis are three-fold. First of all, an *overview of the literature* relevant to learning by game-building was presented. This overview revealed the interdisciplinary nature of the approach, bringing together research from a number of different fields including theoretical computer science, programming, motivation, and education.

Second, a *novel approach to theoretical computer science education* was presented in which students master the modelling-skills on the TCS curriculum by completing game-building tasks. Several examples were included and the pragmatics of using these tasks in the classroom were discussed.

Third, an *accompanying methodology* was specified to test the hypothesis that the success of the approach depends on the amount of control available in the task. Two experimental conditions were formulated which were equal in all aspects except for the amount of control available in the task.

The **findings** of this thesis can be summed up in four items. First of all, Study 1 showed that university lecturers perceive student motivation to be lower on the material

in TCS than on their other informatics subjects. Each of the three reasons speculated in the literature for this were backed up by the data collected in Study 1:

1. difficulty of the subject matter
2. lack of enjoyment, frequently referred to as 'boring'.
3. lack of relevance to the university degree and future career

Second, a majority demand for control in game-building tasks was observed in Study 2 where by slight abuse of terminology, demand was defined to mean the following:

There is a demand for personalizable tasks among informatics undergraduates, if at least some of them choose these tasks over non-personalizable tasks when given a choice between the two. The size of the demand depends on the number of students preferring personalizable tasks over non-personalizable tasks.

In light of this definition, the majority demand observed in Study 2 can also be described as follows: When participants were presented with two game-building tasks, one of which was personalizable and one of which was not, more than 50% chose the personalizable task over the non-personalizable one. Furthermore, almost 50% of own-game choosers entered their game into the peer-reviewed competition. Additionally, the analysis of the relationship between the participants' choice and motivational type revealed the following:

- pre-defined game choosers score *higher* on *mastery-avoidance* than own-game choosers
- own-game only choosers score *higher* on *mastery avoidance* than own-game plus competition choosers
- own-game only choosers score *higher* on *work avoidance* than own-game plus competition choosers
- own-game only choosers score *lower* on *mastery approach* than own-game plus competition choosers
- own-game only choosers score *lower* on *performance approach* than own-game plus competition choosers

Third, Study 3 revealed an overall increased enjoyment of an own-game task, compared to a pre-defined task, after controlling the results for motivational type and gender. Furthermore, several interaction effects between the control variables were observed:

- **highly motivated participants enjoyed the pre-defined task better**
- there is a tendency for participants with high approach to enjoy the pre-defined task better
- there is a tendency for female participants to enjoy the own game task better

Finally, Study 3 did not find an overall effect of personalization on performance. However, we did find that participants with a low performance avoidance score performed better overall. Furthermore, two interesting interaction effects were observed:

- **participants with low avoidance perform better on the own game task**
- there is a tendency for highly motivated participants to perform better on the pre-defined task

7.2 Discussion

This section discusses to what extent the findings described above answer the thesis questions posed in Chapter 1.

Question 1: *Are students really less motivated for TCS than they are for their other informatics subjects? If so, why?*

Study 1 allows us to answer a tentative ‘yes’ to the first part of these questions with the three reasons provided in the literature as a justification, because this represents the view of the international group of university lecturers who participated in the study. However, the ‘yes’ is a tentative one because it is based on observed motivation, rather than actual motivation. Therefore, an additional study using an international group of informatics *students* could provide additional insight into the particulars of the problem.

The fact that our participants were lecturers did allow us to obtain additional information that could not have been provided. For example, several of our participants

found TCS hard or not enjoyable as a student, but nevertheless ended up teaching it for a living and enjoying it as well. This suggests that students' likes and dislikes are not set in stone and that our approach has potential to succeed. Furthermore, lecturers were able to reflect on different teaching techniques on the same subject and provide their opinion of why TCS is important to teach and which topics are the most important within TCS.

Question 2: Can we use game-building as an approach to TCS education? If so, how?

Chapter 4 provided a positive answer to this question in the form of a detailed overview of the approach including several examples. The one caveat is its restriction to the modelling skills on the TCS curriculum – such as Finite State Automata (FSAs), Turing Machines (TMs), or the Calculus of Communicating Systems (CCS) – and properties of the resulting models.

Question 3: *Is there a demand for personalizable tasks among informatics undergraduates?*

Study 2 enables us to answer 'yes', as observed by the fact that more than 50% of participants chose the personalizable task over the non-personalizable task.

Question 4: *Is there a motivational difference between students who choose personalization and those who do not?*

Again, the findings of Study 2 enable us to answer 'yes'. Furthermore, the correlation between choice and the variables that make up a student's motivational type provided initial evidence of a link between motivational type and the experimental conditions 'own-game' and 'pre-defined game'.

However, the results of this study do depend heavily on the motivational framework of achievement goals and the measure that was used, the Achievement Goal Questionnaire (AGQ). This framework reveals only a fraction of the full complexity of a person's motivation and focusses mainly on a person's fears and ambitions using a fairly crude set of distinctive classes of achievement goals. Different motivational frameworks focus on different aspects of motivation and would have possibly shone an entirely different light on our performance and enjoyment data.

Furthermore, Study 2 was conducted in an assessed assignment setting, which could have significantly affected the participants' choice of task. For example, participants who are afraid of receiving a low mark, may choose whichever task they

perceive to be the easy option, rather than the one they would have chosen if no marks were awarded for the task.

Finally, Study 2 does not tell us anything about the demand for game-building tasks themselves, because all experimental conditions were concerned with building a game. Personalization and competition choice were the only isolated variables in this study.

Question 5: *What is the effect of increased personalizability on enjoyment and performance?*

Study 3 showed us that although the own game task is enjoyed more than the pre-defined game task after controlling for motivational type and gender, there is no effect of personalization on performance. While the former was predicted by the literature, the latter was unexpected, as the literature generally predicts increased enjoyment of a personalizable task without the use of control variables. The fact this study too was conducted in an assessed assignment setting might serve as an explanation for this unexpected finding. Deci (1971) and other showed that the presence of extrinsic motivation can decrease the intrinsic motivation already present. As a result, any intrinsic motivation incited by the addition of personalization could be negated by the marks that are awarded for completion of the game-building task.

Furthermore, the performance used a participant's performance on the complete assignment. This means their marks are composed of three parts: (1) the marks for the introduction to the game-engine which all participants are expected to be able to complete without difficulty account for 40%, (2) marks on the game itself account for 30%, and (3) marks on a set of theory questions about the same modelling skill as the game-building task. These last marks account for another 30% of the total mark. So even though performance does include marks on theory questions, the marks are still somewhat biased towards the game-building marks and tell us more about how a participant did on the game-building task and introduction, than about how they did on other questions about the same modelling skill.

Additionally, Study 3 did not tell us anything about a participants enjoyment and performance on game-building tasks in general, because both experimental conditions were concerned with building a game. Personalization/choice was the only isolated variable in this study.

Finally, personalization/choice was measured on two levels: (1) no choice – the pre-defined game task and (2) a large but bounded amount of choice – i.e. the own game task. However, Section 3.5 showed that previous research suggests there is a

motivational difference between a condition with a moderate amount of choice and conditions with a large or non-existent amount of choice. Study 3 does not provide any insight into the condition with moderate choice. This condition seems inherently suitable for those students who find the own-game task too overwhelming.

Question 6: *For which kinds of students do these factors compare favourably and for which do they compare unfavourably?*

Study 3 showed that in terms of enjoyment, highly motivated participants enjoy the pre-defined task better. In terms of performance, it showed that participants with low avoidance perform better on the own game task.

The first result in particular is an interesting one because it was somewhat unexpected, although a similar effect was found by (Squire, 2004) as well. Squire found that students who did well in a traditional formal educational setting, were resistant to new approaches and in some cases actually opted out of them in favour of a traditional reading group.

In our particular case, the first result could indicate similar resistance to new approaches from highly motivated students. This does not provide any insight into the cause of this resistance though. For example, does the resistance represent a fear of straying outside the comfort zone where the highly motivated student is currently top of the pack? Or does it represent an annoyance with having to deal with something that 'distracts' from the actual learning content? And if the latter holds true, does this mean that poorly motivated students view the personalisation as an extrinsic motivator, or are they cultivating an intrinsic motivation for TCS itself?

In Study 1 we learned that some of the students who dislike TCS or find it very difficult, may go on to become a university lecturer of the subject and enjoy it as well. Does this mean they have acquired a previously not existent intrinsic motivation for TCS? Research does suggest that the better we become at something, the more we tend to enjoy it, provided the increased competence is accompanied by a sense of autonomy (Deci and Ryan, 1985; Vallerand and Reid, 1984; Ryan, 1982). This could mean that if personalization, or any other approach for that matter, manages to create an initial boost in performance, it could snowball into an ever growing intrinsic motivation for the subject.

Once again, the fact that the study was conducted in an assessed assignment setting, may have affected the results of this study. For example, participants who are afraid of receiving a low mark, may have not have enjoyed the own game task as much as they

would have, had there not been the pressure of a mark in the equation. Furthermore, only a small percentage of the variance in performance and enjoyment between the two experimental conditions was explained by our model. This raises questions about the possibility of other factors, besides motivation and gender, that are of influence here. What are they and how do they affect performance and enjoyment in the experimental conditions? Further research is needed to answer these questions, which brings us to the next section.

7.3 Future Work

As well as attempting to answer the central questions, this thesis has also *raised* a number of further questions and it has brought up new issues of interest.

The Effect of Motivation First of all, there was the issue of context and the fact that our studies were conducted in an assessed assignment setting. This might have influenced our results in several different ways as described in the previous section. It would be interesting to investigate how the results differ when the experiment is conducted in an environment without assessment. In general, the overall enjoyment of either condition would be expected to be higher, because of the absence of the most obvious extrinsic motivation. This same absence is also expected to influence the choice participants make in terms of own game versus pre-defined game, as the former may seem less daunting without the prospect of a mark at the end of it. Other effects that are perhaps more difficult to predict may be revealed between a participant's motivational type and their enjoyment and performance in the two experimental conditions.

Students Profiles Another interesting issue that arose from the results was the possibility of unmeasured factors having an influence on performance and enjoyment of the conditions with and without personalization. Discovering these factors and exploring their relationship with performance, enjoyment, and personalization would be a useful direction for further research.

Furthermore, only two levels of personalisation/choice were used in our studies. As mentioned in the discussion of Question 5, it would be interesting to add a third level of choice positioned between the two levels used in this study. This condition would represent a moderate amount of choice. We would hypothesize it to be most successful for those students who find the amount choice of an own-game task too overwhelming.

This brings us to the more general direction of defining student profiles and the kind of environments in which they thrive. Our study focussed mainly on the motivational aspect of these student profiles and its relationship with the level of available personalization, but one could imagine similar studies to be conducted for countless other aspects of either a student profile, or the task they are asked to perform.

With recent interest in formative instead of summative assessment, a wide range of opportunities has opened up for tailoring education to the individual. To master a certain topic, students could be allowed to choose from several different tasks which all incorporate the same learning objectives, but differ in either presentation, kind of task, available personalisation, or otherwise. This could allow more students to remain engaged learners for longer or even become re-engaged after a period of disengagement. However, in a setting like this one, it would be interesting to investigate to what extent students choose the task best suited to their profile.

This move towards tailoring learning towards the individual is of course relevant not only to theoretical computer science, but also to undergraduate education university wide and potentially beyond.

Link with Academic Performance In the current study, we would have liked to analyze the students' performance and enjoyment on the game-building tasks with respect to their general academic performance, using data from other courses as well as data from the course for which the students completed the game-building tasks. However, the required data were not obtained due to ethical concerns.

Had this data been available, it could have given us valuable insight into the relationship between a student's general academic performance, their motivational type, and their enjoyment and performance on the two game-building tasks. Furthermore, it could have provided support for our conjecture that stronger students tend to be resistant to new approaches.

In future studies of a similar nature, more extensive permission could be obtained from participants in order to allow for these analyses to be performed. More extensive permission would have also allowed us to extract answers to exam questions on the topics covered by the game-building task. This information could be used in future studies in an analysis investigating transfer of the experimental conditions.

Motivational Effect of Game-Building An issue which was raised several times in this thesis is the fact that our studies do not provide any information about the motiva-

tional effect of game-building itself. Both experimental conditions used are specified in terms of game-building. It would be interesting to compare the students' performance and enjoyment on a game-building task with their performance and enjoyment on a task from a more traditional approach. This would allow for the motivational effect of game-building to be isolated from the motivational effect of personalization, as measured in the studies described in this thesis.

Applications Finally, we consider the potential application domains of the approach. In this thesis it was demonstrated that learning by game-building can be applied to the modelling skills in TCS, like Finite State Automata (FSAs), Turing Machines (TMs), etc. However, we believe the potential of the approach is not limited to TCS, but can be applied to other informatics subjects as well. In fact, the approaches similar to the one described in this thesis are not uncommon in novice programming courses for languages like Python (PyGame), PROLOG (Adventure in PROLOG), or Java (RoboCode).

Topics outside informatics are potentially more difficult to capture with our approach. Although in principle, any modelling framework can be used as a domain for learning by game-building, a programming overhead could easily arise when the approach is used in a discipline other than informatics or an other computing science. Informatics students already possess the basic programming skill required for game-building tasks as described in this thesis, which students of other disciplines may not. Having to acquire this skill before being able to start on the game-building task would likely take more time than can be justified by the learning gains of the game-building task.

However, this does not mean the approach is inherently unsuitable for topics outside informatics. It does mean that a lot more thought will need to go into designing an interface which allows student to create the games for these topics. The interface should require minimal programming or no programming at all. Some examples of interfaces like these are Scratch (Maloney *et al.*, 2004), Stagecast Creator (Louca and Zacharia, 2007), and the Neverwinter Nights Aurora toolkit (Good and Robertson, 2006). Although these toolkits were not designed to teach any particular modelling framework through game-building, they do show that games can be created with minimal or no programming.

7.4 Conclusions

This thesis proposed a novel approach to theoretical computer science (TCS) education at undergraduate level. In this approach modelling framework is mapped onto a game-building framework and the student learns about the former by creating a game of their choice using the latter. It was hypothesized that the success of this approach would depend on the amount of personalization available in the game-building task. To test this hypothesis, two experimental conditions were formulated: (1) own-game and (2) pre-defined game. In the former, participants are asked to write a game of their own, whereas in the latter, they are asked to copy a pre-defined game.

A majority of students chose the own-game condition over the pre-defined condition in a game-building task in which participants were given this choice. In addition to the choice of own-game versus pre-defined game, participants could also opt to enter a peer-reviewed competition. Using the motivational data which was obtained from the participants at the start of the experiment, it was observed that that own-game choosers have lower mastery-avoidance than pre-defined game choosers. Furthermore, competition choosers have higher performance- and mastery-approach than those who opted out of the competition. However, these same competition choosers did score lower than their non-competition counterparts on mastery- and work-avoidance.

Finally, it was shown that although the own-game condition is enjoyed more than the pre-defined game condition after controlling for a participant's motivational type and gender, no such effect exists in terms of performance. A number of interaction effects were observed for both enjoyment and performance. The most important one for enjoyment shows that participants with high (positive) motivation tend to enjoy the pre-defined condition more. However, participants with low avoidance performed better in the own-game condition.

The emphasis of this thesis was on *undergraduate* education. For the first time, support was provided for the much used claim that at undergraduate level, theoretical computer science education suffers more than average from a lack of intrinsic motivation among students. Furthermore, within the area of TCS this thesis has concentrated on the *modelling skills* (finite state automata, push-down automata, Turing machines, value-passing CCS, etc.). It was shown that these make up the majority of the curriculum at undergraduate level.

To conclude, this thesis has focussed on capturing the profiles of those students who benefit from a learning environment with increased control and those who do not.

As such, it has taken an important step towards evaluating educational approaches in terms of their success on different types of students, instead of measuring their overall success.

Bibliography

- Adams, L., Kasserian, J., Yearwood, A., Perfetto, G., Bransford, J., and Franks, J. (1988). The effects of facts versus problem-oriented acquisition. *Memory and Cognition*, **16**, 167–175.
- Aldrich, C. (1995). *Learning by Doing: A Comprehensive Guide to Simulations, Computer Games and Pedagogy in e-Learning and Other Educational Experiences*. John Wiley and Sons.
- Amabile, T., Hill, K., Hennessey, B., and Tighe, E. (1994). The work preference inventory: Assessing intrinsic and extrinsic motivational orientations. *Journal of Personality and Social Psychology*, **66**(5), 950–967.
- Anderson, L. W., Krathwohl, D. R., and Bloom, B. S. (2001). *A Taxonomy for Learning, Teaching, and Assessing: A Revision of Bloom's Taxonomy of Educational Objectives*. New York: Longman.
- Archer, J. (1994). Achievement goals as a measure of motivation in university students. *Contemporary Educational Psychology*, **19**, 430–446.
- Bell, T., Witten, I. H., and Fellows, M. (1998). *Computer Science Unplugged: off-line activities and games for all ages*. Lulu.
- Bloom, B. S. (1956). *Taxonomy of Educational Objectives, Handbook I: The Cognitive Domain*. New York: David McKay Co Inc.
- Boekaerts, M. (1986). The measurement of state and trait motivational orientation: Refining our measures. In J. H. L. van den Bercken, E. E. J. D. Bruyn, and T. C. M. Bergen, editors, *Achievement and Task Motivation*. Lisse: Swets and Zeitlinger.
- Bonakdarian, E. and White, L. (2004). Robocode throughout the curriculum. *Journal of Computing Sciences in Small Colleges*, **19**(3), 311–313.

- Bransford, J. D. and Schwarz, D. L. (1999). Rethinking transfer: A simple proposal with multiple implications. *Review of Research in Education*, **24**, 61–100.
- Brophy, J. E. (1983). Conceptualizing student motivation. *Educational Psychologist*, **18**, 200–215.
- Bruggeman, J. (1986). *The effects of modeling and inspection methods upon problem solving in a computer programming course*, page 1821A.
- Card, O. S. (1995). How software companies die. *Windows Sources*.
- Cassell, J. and Jenkins, H. (1998). *Chess for Girls? Feminism and Computer Games*, chapter 1, pages 2–45. MIT Press.
- Clements, D. and Merriman, S. (1988). *Componential developments in logo programming environments, in Teaching and Learning Computer Programming: Multiple Research Perspectives*, pages 13–54. Erlbaum: Hillsdale, NJ.
- Conway, M., Audia, S., Burnette, T., Cosgrove, D., Christiansen, K., Durbin, R. D. J., Gossweiler, R., Kogi, S., Long, C., Mallory, B., Miale, S., Patten, K. M. J., Pierce, J., Schochet, J., Staak, D., Stearns, B., Stoakley, R., Sturgill, C., Viega, J., White, J., Williams, G., and Pausch, R. (2000). Alice: Lessons learned from building a 3d system for novices. In *Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 486–493.
- Conway, M. J. (1997). *Alice: Easy-to-Learn 3D Scripting for Novices*. Ph.D. thesis, Faculty of the School of Engineering and Applied Science, University of Virginia.
- Cordova, D. and Lepper, M. (1996). Intrinsic motivation and the process of learning: Beneficial effects of contextualization, personalization, and choice. *Journal of Educational Psychology*, **88**, 715–730.
- Csikszentmihalyi, M. (1975). *Beyond Boredom and Anxiety*. San Francisco: Jossey-Bass.
- DeCharms, R. (1971). From pawns to origins: Toward self-motivation. *Psychology and Educational Practice*, pages 380–407.
- Deci, E. and Ryan, R. (1991). A motivational approach to self. In R. Dienstbier, editor, *Nebraska Symposium on Motivation*, volume 38, pages 237–288.

- Deci, E. L. (1971). Effect of externally mediated rewards on intrinsic motivation. *Journal of Personality and Social Psychology*, **18**, 105–115.
- Deci, E. L. and Ryan, R. M. (1985). *Intrinsic Motivation and self-determination in human behavior*. New York: Plenum.
- Deci, E. L., Koestner, R., and Ryan, R. M. (1999). A meta-analytic review of experiments examining the effects of extrinsic rewards on intrinsic motivation. *Psychological Bulletin*, **125**(6), 627–668.
- Dehnadi, S. and Bornat, R. (2004). Dividing the sheep from the goats.
- DiSessa, A. A. (2000). *Changing Minds: Computers Learning and Literacy*. The MIT Press.
- Dowson, M. and McInerney, D. (2001). Psychological parameters of students' social and work avoidance goals. *Journal of Educational Psychology*, **93**, 35–42.
- Eisenberger, R., Pierce, W. D., and Cameron, J. (1999). Effects of reward on intrinsic motivation – negative, neutral, and positive: Comments on deci, koestner, and ryan (1999). *Psychological Bulletin*, **125**(6), 677–691.
- Elliot, A. J. (1999). Approach and avoidance motivation in achievement goals. *Educational Psychologist*, **34**, 169–189.
- Elliot, A. J. and Church, M. (1997). A hierarchical model of approach and avoidance achievement motivation. *Journal of Personality and Social Psychology*, **72**, 218–232.
- Elliot, A. J. and McGregor, H. A. (1999). Test anxiety and the hierarchical model of approach and avoidance achievement motivation. *Journal of Personality and Social Psychology*, **76**, 628–644.
- Elliot, A. J. and McGregor, H. A. (2001). A 2 x 2 achievement goal framework. *Journal of Personality and Social Psychology*, **80**(3), 501–519.
- Elliot, A. J. and Thrash, T. M. (2001). Achievement goals and the hierarchical model of achievement motivation. *Educational Psychology Review*, **13**(2).
- Finney, S. J., Pieper, S. L., and Barron, K. E. (2004). Examining the psychometric properties of the achievement goal questionnaire in a general academic context. *Educational and Psychological Measurement*, **64**(2), 365–382.

- Ganster, D. C., Hennessey, H. W., and Luthans, F. (1983). Social desirability response effects: Three alternative models. *The Academy of Management Journal*, **26**(2), 321–331.
- Garris, R., Ahlers, R., and Driskell, J. (2002). Games, motivation and learning: A research and practice model. *Simulation and Gaming*, **33**(4), 441–467.
- Gee, J. P. (2003). *What Video Games have to teach us about Learning and Literacy*. Palgrave Macmillan.
- Gick, M. and Holyoak, K. (1980). Analogical problem solving. *Cognitive Psychology*, **12**, 306–355.
- Goldstein, J. (1994). *Sex Differences in Toy Use and Video Game Play*, pages 110–129. New York: Cambridge University Press.
- Good, J. and Robertson, J. (2006). Learning and motivational affordances in narrative-based game authoring. In *Proceedings of the Narrative and Interactive Learning Environments Conference (NILE 2006)*.
- Greefield, P. M., Brannon, G., and Lohr, D. (1994). Two-dimensional representation of movement through three-dimensional space: The role of video game expertise. *Journal of Applied Developmental Psychology*, **15**(1), 87–104.
- Habgood, M. P. J., E., A. S., and Benford, S. (2005). Intrinsic fantasy: motivation and affect in educational games made by children. In *Proceedings of the AIED 2005 workshop on motivation and affect in educational software*.
- Harackiewicz, J., Barron, K., Carter, S., A., L., and Elliot, A. (1997). Predictors and consequences of achievement goals in the college classroom: maintaining interest and making the grade. *Journal of Personality and Social Psychology*, **73**, 1284–1295.
- Harel, I. (1991). *Children Designers: Interdisciplinary Constructions for Learning and Knowing*. Praeger/Greenwood.
- Harel, I. and Papert, S. (1991). *Constructionism*. Norwood, NJ: Ablex.
- Harris, A., Yuill, N., and Luckin, R. (in press – 2007). Examining the consistency of mastery and performance goals across group and perceived-ability contexts. *British Journal of Developmental Psychology*.

- Hartness, K. (2004). Robocode: using games to teach artificial intelligence. *Journal of Computing Sciences in Small Colleges*, **19**(4), 287–291.
- Healy, J. M. (1998). *Failure to Connect: How computers affect our children's mind and what we can do about it*. Simon & Schuster.
- Joint Task Force for Computing Curricula 2005 (2005). Computing curricula: 2005 overview report. <http://www.acm.org/education/curricula.html>.
- Julkunen, K. (1989). Situation- and task-specific motivation in foreign-language learning and teaching. *Publications in Education* 6, Univ. of Joensuu, Finland.
- Kafai, Y. B. (1995). *Minds in Play: Computer Game Designs as a Context for Children's Learning*. Lawrence Erlbaum Associates.
- Kafai, Y. B. (1996). *Gender Differences in Children's Construction of Video Games*, chapter 4, pages 90–114. Norwood, N.J.: Ablex Publishing.
- Kahn, K. (2000). Generalizing by removing detail: How any program can be created by working with examples. In H. Lieberman, editor, *Your Wish Is My Command: Programming By Example*. Morgan Kaufmann.
- Kahn, K. (2004). Toontalk: Steps towards ideal computer-based learning environments. In M. Tokoro and L. Steels, editors, *A Learning Zone of One's Own: Sharing Representations and Flow in Collaborative Learning Environments*. Ios Pr Inc.
- Kieran, C. (1986). Logo and the notion of angle among fourth and sixth grade children. In *Proceedings of PME 10*, pages 99–104. City University: London, England.
- Kolb, D. A. (1984). *Experiential Learning: experience as the source of learning and development*. New Jersey: Prentice-Hall.
- Korte, L., Anderson, S., Good, J., and Pain, H. (2007). Learning by game-building: A novel approach to theoretical computer science education. *SIGCSE Bull.*, **39**(3), 53–57.
- Kozen, D. C. (1997). *Automata and Computability*. Springer Science+Business Media, Inc.
- Kuh, G. D. (1995). The other curriculum: out-of-class experiences associated with student learning and personal development. *Journal of Higher Education*, **66**(2), 123–155.

- Lakhani, K. and Wolf, R. (2003). Why hackers do what they do: Understanding motivation and effort in free/open source software projects. MIT sloan working paper, MIT.
- Lave, J. (1988). *Cognition in practice: Mind, mathematics, and culture in everyday Life*. Cambridge University Press.
- Lave, J. and Wenger, E. (1991). *Situated learning: Legitimate peripheral participation*. Cambridge University Press.
- Lepper, M. R. (1973). Undermining children's intrinsic interest with extrinsic reward: A test of the 'overjustification' hypothesis. *Journal of Personality and Social Psychology*, **28**, 129–137.
- Lepper, M. R. and Malone, T. W. (1987). Intrinsic motivation and instructional effectiveness in computer-based education. In R. E. Snow and M. J. Farr, editors, *Aptitude, Learning and Instruction: III. Conative and affective process analyses*, pages 243–286. Hillsdale, NJ: Erlbaum.
- LeWinter, B. (1986). *A study of the influence of Logo on locus of control, attitudes toward mathematics, and problem-solving ability in children in grades 4, 5, 6*, page 1640A.
- Littlefield, J., Delclos, V., Lever, S., Clayton, K., Bransford, J., and Franks, J. (1988). Learning logo: Method of teaching, transfer of general skills, and attitudes toward computers. In R. E. Mayer, editor, *Teaching and Learning Computer Programming: Multiple Research Perspectives*, pages 111–135. Lea.
- Louca, L. and Zacharia, Z. S. (2007). The use of computer-based programming environments as computer modelling tools in early science education: The cases of textual and graphical program languages. *International Journal of Science Education*, **30**(3).
- Lunney, G. H. (1970). Using analysis of variance with a dichotomous dependent variable: An empirical study. *Journal of Educational Measurement*, **7**(4), 263–269.
- Malone, T. W. (1981). Toward a theory of intrinsically motivating instruction. *Cognitive Science*, **5**(4), 333–369.

- Malone, T. W. and Lepper, M. R. (1987). Making learning fun: A taxonomy of intrinsic motivations for learning. In R. E. Snow and M. J. Farr, editors, *Aptitude, Learning and Instruction: III. Conative and affective process analyses*, pages 223–253. Hillsdale, NJ: Erlbaum.
- Maloney, J., Burd, L., Kafai, Y., Rusk, N., Silverman, B., and Resnick, M. (2004). Scratch: A sneak preview. In *Second International Conference on Creating, Connecting, and Collaborating through Computing*, pages 104–109.
- McGregor, H. A. and Elliot, A. J. (2002). Achievement goals as predictors of achievement relevant processes prior to task engagement. *Journal of Educational Psychology*, **94**, 381–395.
- Merrill, D. (1999). What motivated the mtv generation? some comments on motivation. Unpublished.
- Merritt, D. (1990). *Adventure in PROLOG*. Springer-Verlag Berlin and Heidelberg GmbH and Co.
- Midgley, C., Maehr, M. L., Hruda, L. Z., Anderman, E., Anderman, L., Freeman, K. E., Gheen, M., Kaplan, A., Kumar, R., Middleton, M. J., Nelson, J., Roeser, R., and Urdan, T. (2000). Manual for the patterns of adaptive learning scales (pals). Technical report, Ann Arbor, MI: University of Michigan.
- Miles, M. B. and M., H. A. (1994). *Qualitative Data Analysis: 2nd Edition*. Thousand Oaks, CA: Sage.
- Milojkovic, J. (1984). *Children learning computer programming: Cognitive and motivational consequences*, page 385B.
- Moskal, B., Lurie, D., and Cooper, S. (2004). Evaluating the effectiveness of a new instructional approach. In *SIGCSE '04: Proceedings of the 35th SIGCSE technical symposium on Computer science education*, pages 75–79, New York, NY, USA. ACM.
- Myers, I. B. and Myers, P. B. (1980, 1995). *Gifts Differing: Understanding Personality Type*. Mountain View, CA: Davies-Black Publishing.
- Nelson, M. (www). <http://robocode.sourceforge.net/>.

- Papert, S. (1980). *Mindstorms: Children, Computers, and Powerful Ideas*. Basic Books New York.
- Parker, L. and Lepper, M. (1992). The effects of fantasy contexts on children's learning and motivation: Making learning more fun. *Journal of Personality and Social Psychology*, **62**, 625–633.
- Perkins, D. and Salomon, G. (1992). Transfer of learning. In *International Encyclopedia of Education*. Pergamon Press.
- Peterson, P. (1998). Guillotine. Wizards of the Coast.
- Pfaffman, J. A. (2003). *Manipulating and Measuring Student Engagement in Computer-Based Instruction*. Ph.D. thesis, Vanderbilt University.
- Piaget, J. (1926). *The language and thought of the child*. Harcourt, Brace and Co.
- Piaget, J. (1965). *Science of education and the psychology of the child*. New York: Viking.
- Piaget, J. I. (1969). *The psychology of the child*. Basic Books, New York.
- Pieper, S. L. (2003). *Refining and Extending the 2x2 Achievement Goal Framework: Another Look at Work-Avoidance*. Ph.D. thesis, James Madison University, Harrisonburg, Virginia.
- Prensky, M. (2001). *Digital Game-Based Learning*. McGraw-Hill.
- Provenzo, E. F. (1991). *Video Kids: Making Sense of Nintendo*. Harvard University Press.
- Reiber, L. P., Luke, N., and Smith, J. (1998). Project kid designer: Constructivism at work through play. *Meridian: Middle School Computer Technology Journal*, **1**(1).
- Reis, R. and Moreira, N. (2004). Turing machine simulator in python. <http://www.ncc.up.pt/fado/node3.html>.
- Resnick, M. (1996). *Termites, turtles and traffic lights. Explorations in massive computation*. MIT Press, USA.
- Robertson, J. and Good, J. (2005). Story creation in virtual game worlds. *Communications of the ACM*, **48**(1), 61–65.

- Rodger, S. H., Bressler, B., Finley, T., and Reading, S. (2006). Turning automata theory into a hands-on course. In *37th SIGCSE Technical Symposium on Computer Science Education*, volume 38, pages 379–384. Houston, Texas, USA.
- Ryan, R. M. (1982). Control and information in the intrapersonal sphere: An extension of cognitive evaluation theory. *Journal of Personality and Social Psychology*, **43**, 450–461.
- Ryan, R. M. and Deci, E. L. (2000). Intrinsic and extrinsic motivations: Classic definitions and new directions. *Contemporary Educational Psychology*, **25**(1), 54–67.
- Sipser, M. (1997). *Introduction to the Theory of Computation*. PWS Publishing Company.
- Skinner, B. F. (1953). *Science and Human Behavior*. New York: Macmillan.
- Soloway, E. (1993). Should we teach students to program? *Communications of the ACM*, **36**(10), 21–24.
- Squire, K. (2002). Rethinking the role of games in education. *Game Studies*, **2**(1).
- Squire, K. D. (2004). *Replaying history: learning world history through playing Civilization III*. Ph.D. thesis, School of Education, Indiana University.
- Steele, O. (www). Finite state automata simulator in python. <http://osteele.com/software/python/fsa/FSA.html>.
- Stenning, K. (2002). *Seeing Reason: Image and language in learning to think*. Oxford University Press.
- Stoll, C. (1999). *High Tech Heretic: Why computers don't belong in the classroom*. Doubleday.
- Valente, A. (2003). C-cards: using paper and scissors to understand computer science. In *Proceedings of Kolin Kolistelut - Koli Calling*, pages 84–92.
- Valente, A. (2004). Exploring theoretical computer science using paper toys (for kids). In *Proceedings of the IEEE International Conference on Advanced Learning Technologies (ICALT'04)*, Joensuu, Finland.

- Vallerand, R., Pelletier, L., Blais, M., and Briere, N. (1992). The academic motivation scale: A measure of intrinsic, extrinsic, and amotivation in education. *Educational and Psychological Measurement*, **52**, 1003–1017.
- Vallerand, R. J. and Reid, G. (1984). On the causal effects of perceived competence on intrinsic motivation: A test of cognitive evaluation theory. *Journal of Sport Psychology*, **6**, 94–102.
- Verma, R. M. (2005). Visual and interactive automata theory course emphasizing breadth of automata. In *10th Annual SIGCSE Conference on Innovation and Technology in Computer Science Education*, volume 37, pages 325–329. Monte de Caparica, Portugal.
- White, R. W. (1959). Motivation reconsidered: The concept of competence. *Psychological Review*, **66**, 297–333.
- White, T. M. and Way, T. P. (2006). jfast: a java finite automata simulator. In *Proceedings of the 37th SIGCSE technical symposium on Computer Science Education*, volume 38, pages 384–388.
- Zuckerman, M., Porac, J., Lathin, D., Smith, R., and Deci, E. L. (1978). On the importance of self-determination for intrinsically motivated behavior. *Personality and Social Psychology Bulletin*, **4**, 443–446.

Appendix A

Questionnaire 1

Teaching TCS - Lecturers' Questionnaire

Welcome L.Korte@sms.ed.ac.uk and thank you for your participation in this online questionnaire! Our research is conducted in the context of the PhD project Learning by Game-Building by Laura Hutchins-Korte. With the help of your answers we are hoping to gain a better insight into the motivation of students - both past and present - in studying theoretical computer science at the undergraduate level. In particular, we are hoping to find if any factors have a negative influence on student motivation in this area and to evaluate selected teaching activities that may minimise these factors and improve student motivation.

N.B. There are two parts to the main questionnaire: (A) background information, and (B) theoretical computer science education. Because this is a pilot version, we have added a third part (C) in which you are asked to provide feedback on the questionnaire. You can use the "Save" buttons on the form to save individual questions or the "Save All" and "Save All & Exit" at the bottom of the page to save all your answers. Please note that the latter will also exit this interface. You can of course always log back in later to change your answers and/or answer additional questions using your email address and password. If you have any questions regarding this questionnaire please do not hesitate to contact me at L.Korte@sms.ed.ac.uk.

A. Background Information

- (1) Your name:
- (2) At which university do you currently teach?
- (3) Which undergraduate/postgraduate degrees do you hold? For each degree please specify the subject, kind of degree (BSc, MA, PhD, etc.), and the university it was received from.
- (4) Please describe your research interests in a maximum of 10 lines.
- (5) At which universities have you taught TCS topics at the undergraduate level?
- (6) Education Portfolio: Which TCS topics have you taught at the undergraduate level? Which other computing topics have you taught? Which other subjects have you taught?
- (7) Of the subjects that you have taught, which do you prefer teaching? (Please give any reasons for this preference)

B. Theoretical Computer Science Education

- (8) Do you like teaching TCS topics? If so, why? If not, why not?
- (9) In undergraduate computing degrees, how important do you think it is to teach TCS topics? Please explain your answer.
- (10) Which TCS topics do you feel are the most important to teach at the undergraduate level? Please explain your answer.
- (11) Which TCS topics do you feel are less important to teach at the undergraduate level? Please explain your answer.
- (12) From your experience, how would you rate students' enjoyment of TCS topics compared to other computing topics? Why do you think students like/dislike TCS topics?
- (13) From your experience, how would you rate students' difficulty with TCS topics compared to other computing topics? Why do you think students find TCS topics easy/difficult?
- (14) From your experience, how would you rate students' perceived relevance of TCS topics to their studies compared to other computing topics? Why do you think students find TCS topics relevant/irrelevant to their studies?
- (15) Of the three factors influencing student motivation mentioned in questions 12-14 (i.e. enjoyment, difficulty, and relevance), which do you feel is the largest positive (or least negative) influence? Which do you feel is the largest negative (or least positive) influence?

- (16) What was your opinion of TCS when you were a student? Did you feel you were an exception or the rule at that time?
- (17) Do you think there is a difference between student opinion of TCS then and now? If so, can you describe this difference? Why do you think opinion changed?
- (18) Which teaching activities do you think could make TCS more enjoyable for students?
- (19) Which teaching activities do you think could make TCS easier to understand for students?
- (20) Which teaching activities do you think could improve the perceived relevance of TCS for students?
- (21) Which of the activities you suggested in questions 18-20 have you tried in your teaching and what was your experience with them?
- (22) One approach to teaching TCS topics is to assign students a game-building task through which they learn a particular TCS topic (see Learning by Game-Building for more detail on this approach). Have you tried this approach? If so, have you found it successful; what problems does it raise? If not, do you feel it could be successful, or not? What problems do you think it might raise?
- (23) Do you have any additional remarks/comments that you would like to add?

C. Questionnaire Feedback (pilot version only)

- (24) Were there any questions you found particularly difficult to answer? If so, please specify which questions and explain why you found them difficult to answer.
- (25) Did you have any trouble using the interface? Please describe any problems you had.
- (26) Do you have any additional comments/suggestions regarding this questionnaire?

Appendix B

Questionnaire 2

Teaching TCS - Lecturers' Questionnaire

Welcome L.Korte@sms.ed.ac.uk and thank you for your participation in this online questionnaire! Our research is conducted in the context of the PhD project Learning by Game-Building by Laura Hutchins-Korte. With the help of your answers we are hoping to gain a better insight into the motivation of students - both past and present - in studying theoretical computer science at the undergraduate level. In particular, we are hoping to find if any factors have a negative influence on student motivation in this area and to evaluate selected teaching activities that may minimise these factors and improve student motivation.

*N.B. There are two parts to this questionnaire: (A) background information, and (B) theoretical computer science education. You can use the "Save" buttons on the form to save individual questions or the "Save All" and "Save All & Exit" at the bottom of the page to save all your answers. Please note that the latter will also exit this interface. **It may take as much as 20-30 minutes to fill out this questionnaire,** but remember that you can always log back in later to answer additional questions or change any of your answers using your email address and password. **Please feel free to take as many sessions as you require to complete the questionnaire.** If you have any questions regarding this questionnaire please do not hesitate to contact me at L.Korte@sms.ed.ac.uk.*

A. Background Information

- (1) Your name:
- (2) At which university do you currently teach?
- (3) Which undergraduate/postgraduate degrees do you hold? For each degree please specify the subject, kind of degree (BSc, MA, PhD, etc.), and the university it was received from.
- (4) Please describe your research interests in a maximum of 3 lines.
- (5) At which universities have you taught TCS topics at the undergraduate level?
- (6) Education Portfolio: Which TCS topics have you taught at the undergraduate level in the past 5 years? Which other computing topics have you taught? Which other subjects have you taught?
- (7) Of the subjects that you have taught, which do you prefer teaching? (Please give any reasons for this preference)

B. Theoretical Computer Science Education

- (8) Do you like teaching TCS topics? If so, why? If not, why not?
- (9) In undergraduate computing degrees, how important do you think it is to teach TCS topics? Please explain your answer.
- (10) Which TCS topics do you feel are the most important to teach at the undergraduate level? Please explain your answer.
- (11) Which TCS topics do you feel are less important to teach at the undergraduate level? Please explain your answer.
- (12) From your experience, how would you rate students' enjoyment of TCS topics compared to other computing topics? Why do you think students like/dislike TCS topics?
- (13) From your experience, how would you rate students' difficulty with TCS topics compared to other computing topics? Why do you think students find TCS topics easy/difficult?
- (14) From your experience, how would you rate students' perceived relevance of TCS topics to their studies compared to other computing topics? Why do you think students find TCS topics relevant/irrelevant to their studies?
- (15) Of the three factors influencing student motivation mentioned in questions 12-14 (i.e. enjoyment, difficulty, and relevance), which do you feel is the largest positive (or least negative) influence? Which do you feel is the largest negative (or least positive) influence?

- (16) What was your opinion of TCS when you were a student? Did you feel you were an exception or the rule at that time?
- (17) Do you think there is a difference between student opinion of TCS then and now? If so, can you describe this difference? Why do you think opinion changed?
- (18) Which teaching activities do you think could make TCS more enjoyable for students?
- (19) Which teaching activities do you think could make TCS easier to understand for students?
- (20) Which teaching activities do you think could improve the perceived relevance of TCS for students?
- (21) Which of the activities you suggested in questions 18-20 have you tried in your teaching and what was your experience with them?
- (22) One approach to teaching TCS topics is to assign students a game-building task through which they learn a particular TCS topic (see Learning by Game-Building for more detail on this approach). Have you tried this approach? If so, have you found it successful; what problems does it raise? If not, do you feel it could be successful, or not? What problems do you think it might raise?
- (23) Do you have any additional remarks/comments that you would like to add?

Appendix C

Communicating Objects Assignment (2005–2006)

Inf1B OOP Assignment 3

Games and Object Communication

Hutchins, Korte

Out: February 17th, 2006

Due: March 3rd, 2006

Materials: Download `assignment3.zip` from the informatics 1 website:

```
http://www.inf.ed.ac.uk/teaching/courses/inf1/oop/assignments/  
assignment3.zip
```

Submission: Type the following commands at the UNIX shell prompt to submit this assignment:

```
submit inf1 inf1b A04 answers.txt gamedemo.txt  
cd workspace/Game  
submit inf1 inf1b A04 *.java
```

Research: Laura Korte, whom many of you will know from Inf1B lab-sessions and tutorials, is doing her PhD research on the topic of *knowledge modeling through game building*. She would like to use your submissions to this assignment for her research. Please note that the identity of participants will remain anonymous. Furthermore, any document containing results of this research project will be presented in such a way that no individual can be identified.

If you wish for your assignment to be excluded from this research project, please mention this in your plain-text file `answers.txt`. Also, if you have any further questions, please email Laura at orange.laura@gmail.com.

Optional: If you choose to write your own game, you have the option of (1) having your game published on the Inf1B website, or (2) entering the peer-reviewed competition – which includes option (1). The names of the winners of this competition will be published on the Inf1B website. If you wish to have your game published or you wish to enter the competition, please specify this in your plain-text file `answers.txt`.

The fine print: Late submissions will not be accepted. Collaboration with other students on this assignment is not permitted. If you have any questions on this assignment, please post them to the newsgroup `eduni.inf.course.inf1b` or ask a demonstrator in one of the drop-in labs sessions.

1 Overview

The purpose of this assignment is to teach you how to coordinate the activities of many objects which all interact with each other. You will do this by building your own computer game! You have two options available to you:

Option (A): You can design and build your own computer game. Unlike the other assignments in informatics 1, this option gives you the chance to be creative. You can build any kind of game you want, so long as it fulfills the requirements listed in section 6. You are limited only by your imagination, programming skill, and free time!

Option (B): You can build a game called “The Tower of London”, which is specified in section 7. The amount of programming is the same as option (A), but you don’t have to design the game yourself.

If you choose to design your own game, you may choose to publish it on the Informatics 1 website. Just mention that you want it published in `answers.txt`.

You may also choose to enter your game into the 2006 Inf1 programming competition. Again, just mention it at the top of `answers.txt`. If you enter the competition, your game will be sent out to other Inf1 students to judge. The other students will vote on a winner, and the winner will be announced to the class. All entries to the competition will be posted to the website.

1.1 Downloads

The source code for this assignment can be found at:

```
http://www.inf.ed.ac.ac.uk/teaching/courses/inf1/oop/assignments/  
assignment3.zip
```

You should create a new project named “Game” in eclipse, and import the code into it as usual.

2 The Game Engine

The major challenge to writing any modern game is making good graphics. The games you see on the shelves today are produced very much like movies; whole teams of artists work for years to create the stunning visual effects that you see on the screen.

Since you only have two weeks to complete this assignment (and no teams of paid artists), you’ll have to cheat and use ASCII graphics instead. Nevertheless, it’s possible to make entertaining games without fancy graphics; it just takes a bit more creativity!

You’ll be using a *game engine* which has been developed specifically for Informatics 1. The game engine is a java library which does all the hard work of drawing, animation, and event processing. Games written with this engine are laid out in a `World` which is much like a chessboard. The world is divided up into rows and columns. Each square may be empty, or it may contain an object.

Figure 1 shows a screenshot from “The Tower of London”. Every symbol you see on the screen is an object. The player is represented by a happy face. The ‘c’ character is a cat, the ‘G’ is a guard, and each ‘t’ is a tourist. Some objects (such as the cat and the tourists) move around, while other objects (such as the walls) are fixed. The class of each object determines its behavior, so you can make new kinds of object simply by defining new classes.

Objects can communicate with the player by posting messages to the bottom of the screen. Here, several tourists (the red ones) have been shot by a marauding happy face. The area on the side of the screen shows the *current state* of the player. In “The Tower of London” it shows how many whiskey bottles you have stolen.

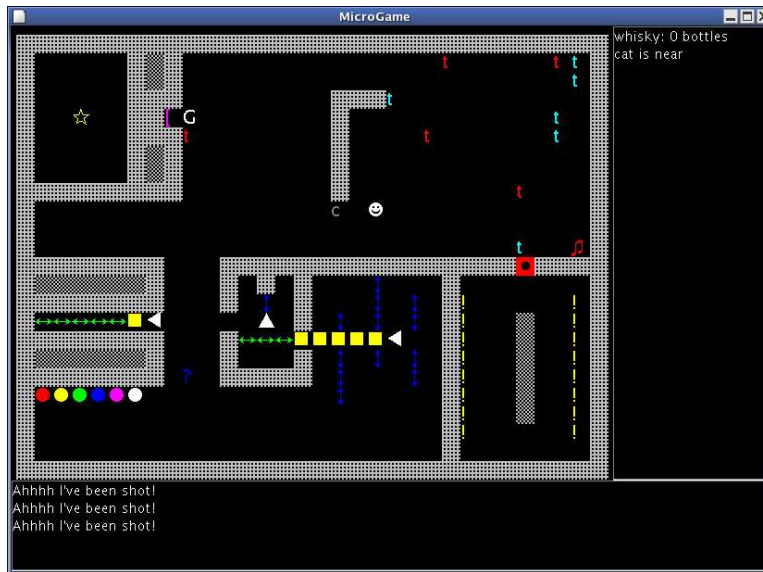


Figure 1: The Tower of London

2.1 Entities

All objects in the world must derive from class `Entity`. Each entity has the following characteristics:

- A **name**. This is generally the name of the class, such as “Tourist” or “Cat”. It is possible to search for entities by name.
- A **symbol** which is displayed on the screen. The symbol is a string containing one character. The symbol can be any character on the keyboard, or one of several special symbols which have been predefined.
- A **color**.
- A **speed**, which defines how fast the entity moves and responds. (see below)

Entities may be either *active* or *inactive*. The game has an internal clock which ticks 40 times a second. Active entities receive regular signals (called *step* events) based on that clock. An entity with a speed of 1 (the fastest) receives a step event every clock tick. An entity with a speed of 10 receives a step event every 10 clock ticks. Bullets travel very fast (they have a speed of 2) while tourists move more slowly (they have a speed of 10).

Active entities must override the following method to respond to step events:

```
public void step() { ... }
```

Inactive entities do not receive regular step events, but that doesn't mean that they can't respond to other events, such as being pushed or shot. These will be discussed in the next section.

2.2 Things

There are two kinds of entity in the world: *Things* and *Floors*. A **Thing** is a solid object. Only one **Thing** can be on the same square at any one time. Things can move around,

bump into each other, and shoot bullets. Things are the only kind of entity that you have to deal with in this assignment.

There are four main actions that a Thing can take:

- `boolean move(Direction d)`: The Thing will try to move in the given direction. Valid directions are `Direction.north`, `Direction.south`, `Direction.east`, and `Direction.west`. The square that the thing is moving to must be empty. If the square is not empty, then `move()` will return `false`.
- `boolean moveTo(int x, int y)` The Thing will try to move to absolute coordinates (x, y) . The lower-left hand coordinate of the world is $(1,1)$. Again, the target location must be empty.
- `boolean push(Direction d, int force)`: The *push* command can be used whenever a Thing wants to move onto a square that's currently occupied by another Thing. This command will first send a *push event* to the other object, and then try to move onto the square. Push events are discussed below.
- `boolean shoot(Direction d, int speed)`: This command will shoot a bullet in the given direction. The bullet will travel at the given speed (the default is 2). When another Thing gets hit by a bullet, it receives a *shot event*.

Things can interact with other Things by responding in different ways when they are pushed or shot. For example, when a boulder is pushed, it should move out of the way. Once the boulder has vacated its square, the Thing that pushed it will automatically move onto the spot where the boulder used to be.

In order to respond to a push event, a Thing must override the `pushed(...)` method. There are three arguments to `pushed()`, as shown in the example below. The `pusher` is a reference to the Thing that does the pushing, while `d` is the direction of the push. (A boulder would need to move in the same direction as the push.) The `force` is specified in the original call to `push()`, as mentioned earlier. (A heavy boulder might require a certain amount of force to move.) By default, the player pushes with force 5.

For example, assume we have a pound coin. In this case, "pushing" the coin is the same as picking it up. Consider the following code:

```
class PoundCoin extends Thing {
    ...
    public void pushed(Thing pusher, Direction d, int force) {
        if (pusher == getPlayer()) {
            MyPlayer p = (MyPlayer) pusher; // downcast to MyPlayer
            p.giveMoney(1); // give one pound to the player
            destroy(); // remove the coin
        }
    }
    ...
}
```

This class checks to see if the player is doing the pushing, and if it is, it gives money to the player and removes the coin. Once the pound coin is gone (having been picked up) the player will move onto the square occupied by the coin. Since the type of `pusher` is `Thing` and not `MyPlayer`, you must *downcast* `pusher` to the appropriate type in order to call a player-specific method on it.

Things can also respond to getting shot by implementing the following method:

```
public void shot(Thing shooter, Direction d) { ... }
```

Here `shooter` is the Thing that shot the bullet, while `d` is the direction that the bullet was traveling when it hit.

2.3 The Player

Each game needs exactly one object which derives from `Player`. The player class for your game is defined in the file `MyPlayer.java`.

The `Player` is a `Thing`, so it can move around, push and shoot. However, the `Player` also has state, and it responds to keyboard events. There are two methods that you may wish to modify, which are described below.

2.3.1 Responding to keyboard events

```
public void keyPressed(int keycode, boolean shift,
                       boolean alt, boolean ctrl)
{ ... }
```

The `keyPressed()` method is called whenever the user presses a key. The `shift`, `alt`, and `ctrl` arguments specify whether any of those keys are currently held down. The `keycode` is a number that indicates which key was pressed; it will be of the form `KeyEvent.VK_???`. For a full list of all keycodes, see the following URL:

<http://java.sun.com/j2se/1.4.2/docs/api/java/awt/event/KeyEvent.html>

By default, the player will move when the arrow keys are pressed, and shoot bullets when `shift + arrow` keys are pressed. You may add additional actions if you wish.

2.3.2 Player State Variables

In a typical game, the player class has a number of *state variables* associated with it, which keep track of health, ammunition, money, or any special items that the player has picked up.

You must override `currentState()` to display the current state of the player. The method should return a string which displays all the state information. For example:

```
public String currentState() {
    return "Health: " + getHealth() + "\n" +
           "Ammo:   " + getAmmunition() + "\n";
}
```

Each line has displays a single piece of information, and ends with `"\n"`, which indicates "end of line".

2.3.3 Additional notes

Because `Player` is a `Thing`, you can also override `pushed()` and `shot()`. However, if you choose to override `step()`, your implementation *must* call `super.step()`, or your player will no longer respond to the keyboard.

2.4 The World

Once you have created some `Things` and a `Player`, you will need to create a `World`, which is stored in the file `MyWorld.java`.

There are two steps to creating a world. First, you must encode the game board as an array of strings. Second, you must implement `makeTile()` so that it decodes those strings to make all of the objects in the game. The game board looks something like the following, although it can be as big, or as small, as you want:

which can be done with the following method:

```
public void showMessage(String message)
```

If a Thing needs to show more than one line of text, or if it needs to ask the user a question, then it should use a *conversation*.

A conversation has two parts. It has a block of text which can be as long as you want. The text is HTML, so you can also use bold, italic, tables etc. to display information. In addition to the text, a conversation may have one or more buttons that the user can click on to make a response, as seen in figure 2.



Figure 2: A conversation

There are two kinds of buttons that a conversation can have. A *return button* will close the conversation window, and return a string as an answer. A *continue button* will jump to another part of the conversation, which may have different text and/or response buttons. The following code snippet shows how to make a knock-knock joke:

```
Conversation joke      = new Conversation("Knock knock...");
Conversation nextline = new Conversation("Anna.");
Conversation punchline = new Conversation("Anna body know a better joke?");

joke.addContinueButton("Who's there?", nextline);
nextline.addContinueButton("Anna who?", punchline);
punchline.addReturnButton("Ha ha!", "funny");
punchline.addReturnButton("That's stupid!", "notfunny");

// open a window entitled "My Joke" which starts the conversation
String result = showConversation("My Joke", joke);
if (result == "funny") ...
```

There are three parts to the joke – the start, the middle, and the punch line. Each part of the conversation continues on to the next part. Once the punch line has been delivered,

the user has two choices. Clicking on the <Ha! Ha!> button will end the conversation and return the string "funny", while clicking on the <That's stupid!> button will return the string "notfunny". The object can then test this answer and decide what to do next.

Important Note: The markers will mark your game by running a demo that you record. The demo records all keystrokes, but it does *not* record conversations. To enable your game to be marked, you *must* put an asterisk (*) by the correct response in each conversation, as seen in figure 2.

2.5.2 Random Numbers

The function `randomInt(n)` will return a random number between 0 (inclusive) and n (exclusive). The function `Direction.random()` will return a random direction.

You should *not* use the standard Java libraries to generate random numbers, because you will not be able to record a proper demo.

2.5.3 Special Symbols

The `Symbols` class contains a number of predefined symbols, such as circles, boxes, stars, arrows, etc. that you can use in your game. See `Symbols.java` for details.

2.5.4 Finding Entities by Name

Some Entities in your game may need to interact with other Entities that are far away. For instance, a doorbell may cause a door somewhere else to open, or a guard may call out to other guards for reinforcements. In these cases, you will need to find the other Entities in order to send them a message.

The command `findEntities(name)` will search for all entities in the world which have the given name. It returns an *iterator*. Iterators in Java are much like lists in Haskell — they allow a group of objects to be processed in a loop. You can use iterators as follows:

```
...
Iterator iter = findEntities("Whatsit");
while (iter.hasNext()) {
    Whatsit what = (Whatsit) iter.next(); // downcast
    what.doSomething();
}
...
```

This example will call the method `doSomething()` on every entity named "Whatsit" in the world. The *name* of an entity is the string returned by `getName()`. It is usually, but not necessarily, the same as the class name. As in the coin example, you must downcast to the appropriate type.

3 Recording a Demo of your Game

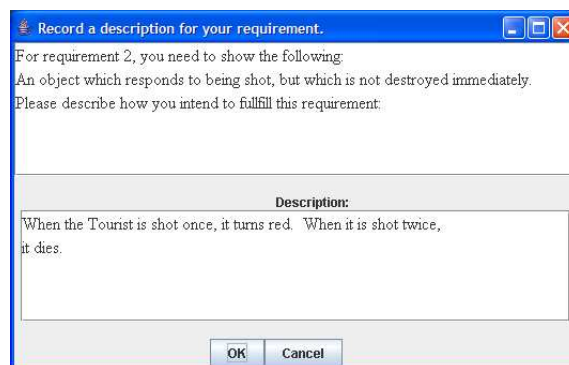
We would like to encourage you to put interesting plots and difficult puzzles in your game. Unfortunately, the markers won't have time to solve all of the puzzles that every student submits. Instead, you must record a demo of your game that the markers can play in order to evaluate it. In addition, there are certain requirements that your game must fulfill, and you will use the recorder to document how your game fulfills those requirements.

When you launch your game, it opens two windows — the game window, and a recorder window, which is shown below.



Press <Record> to start recording a demo, and then play the game as normal. As you play the game, you will interact with certain objects that fulfill certain requirements.

Right before an important interaction, you should press one of the buttons labeled “Begin R n ” where n is the number of the requirement that the interaction will fulfill. Pressing this button will pop up the following dialog, where you are asked to enter a short description of how the interaction fulfills the requirement.



All of these descriptions will be logged to the recorder window, so you can see what you have entered. Moreover, after you have completed a requirement, the button for that requirement will change to “Done R n ”, so you know which ones are still left. The <Play> button will play back the demo, and pause the game at the start of each requirement.

When you have played to the end of the game, press the <Save> button, and save it under the name “gamedemo.txt”. The demo file is an ordinary text file, so if you want to edit some of your descriptions without recording a whole new demo, then you can do so with any text editor.

The recorder records all keystrokes, but it does *not* record conversations. To enable your game to be marked, you must also put an asterisk (*) by the correct response in each conversation, as described in section 2.5.1.

That’s it for the introduction, now on to the actual assignment...

4 Reflection on your Learning (5 points)

In this section you will be answering some questions as a reflection on your learning. There are no right or wrong answers. There are only answers that are true of you, and answers that are not true of you. For each of the following twelve statements, your answer should consist of a number between 1 (not at all true of me) and 7 (very true of me). Please save your answers in a file named **answers.txt**.

1	My goal this semester is to get better grades than most of the other students.	1	2	3	4	5	6	7
2	I just want to avoid doing poorly compared to other students this semester.	1	2	3	4	5	6	7
3	Completely mastering the material in my courses is important to me this semester.	1	2	3	4	5	6	7
4	I am afraid that I may not understand the content of my courses as thoroughly as I'd like.	1	2	3	4	5	6	7
5	It is important for me to do well compared to other students this semester.	1	2	3	4	5	6	7
6	I want to learn as much as possible this semester.	1	2	3	4	5	6	7
7	The fear of performing poorly is what motivates me.	1	2	3	4	5	6	7
8	The most important thing for me this semester is to understand the contents in my courses as thoroughly as possible.	1	2	3	4	5	6	7
9	I worry that I may not learn all that I possibly could this semester.	1	2	3	4	5	6	7
10	I want to do better than other students this semester.	1	2	3	4	5	6	7
11	I am definitely concerned that I may not learn all that I can this semester.	1	2	3	4	5	6	7
12	My goal this semester is to avoid performing poorly compared to other students.	1	2	3	4	5	6	7

5 Introductory Questions (35 points)

In this section you will be asked to implement *blocks*, *sliders*, *pushers*, and *paint tins*. These are all examples of objects you could put in your game later on. But first, let me explain what these four different object are.

Blocks are squares (`Symbol.blackSquare`) that can be pushed around the screen by the player (or by *pushers*!). When a block is pushed against a wall, nothing happens. When a block is pushed against other blocks, all of the blocks will move in the direction that the player (or pusher) is pushing them.

Implementation: Blocks should should respond to a *push* event by pushing the object next to them. The force of the original push should be transfered to the new object. (*Hint:* It doesn't matter exactly what object is next to them, they just need to push it in the right direction! If the next object is a block, then that block will push the next one, and so on down the line. Walls are immovable because they don't do anything when they are pushed.)

Sliders are similar to blocks, but they can only be pushed along one axis. Naturally, they come in two kinds: horizontal sliders (`Symbol.leftRightArrow`), and vertical sliders (`Symbol.upDownArrow`). If something tries to push a horizontal slider up or down, nothing will happen. The same holds if something tries to push a vertical slider to the left or or to the right. However, push a horizontal slider left or right, or a vertical slider up or down, and they will move exactly like blocks.

Pushers do not wait for the player to push them, they move by themselves. Pushers can only move in one direction and thus come in four difference kinds: north pushers (`Symbols.upTriangle`), south pushers (`Symbols.downTriangle`), east pushers (`Symbols.rightTriangle`), and west pushers (`Symbols.leftTriangle`). Each time unit, the pusher will move exactly one square. Like the player, pushers can push groups of blocks and/or sliders until they hit an obstacle.

Implementation: Pushers respond to a *step* event by pushing in the appropriate direction. Pushers have speed 30, and should push with force 10.

Paint tins are circles (`Symbol.blackCircle`) which have a particular color. When the player sticks its little smiley head into a paint tin, it will turn the same color as the paint in the tin. Furthermore, it will stay that way until it is dipped into a paint tin with a different colour.

Implementation: A paint tin responds to a *push* event by changing the color of the pusher. A paint tin can only be used once, so it should die after being pushed.

For each of the following questions, find the appropriate class in `PusherLib`, and fill in the body of the appropriate method.

- a. Implement blocks. *(10 points)*
- b. Implement sliders. *(8 points)*
- c. Implement pushers. *(7 points)*
- d. Implement tins of paint. *(10 points)*

6 Option A – Build Your Own Game. (60 points)

Remember that you should select either Option A or Option B, but not both.

In this section you will be implementing your own game. You can make whatever you want, provided it satisfies the requirements specified below. You will need to record proof that you have satisfied all requirements using the *behaviour recorder*, which is documented in section 3.

Name	Description	Credit
R1	An object which moves around the screen randomly. Its movement can be completely random or partly random.	5 points
R2	An object which responds to being shot, but which is not destroyed immediately.	5 points
R3	An object which modifies a <i>new</i> state variable of the player. This must be a new variable created by you, the student, not one which is present in the original <code>assignment3.zip</code> .	5 points
R4	An object which depends on a <i>new</i> state variable of the player.	5 points
R5	Two communicating objects: one sending a signal, the other receiving it. Neither of the objects should be the player.	10 points
R6	An object which depends on the location of the player.	10 points
R7	An object which modifies the location of the player.	5 points
R8	A conversation which depends on the state of the player.	5 points
R9	A conversation which modifies the state of the player.	5 points
R10	A victory object: if the player pushes or picks up this object, then the player should be informed that he/she has won the game.	5 points

Section 8 contains additional instructions on how to submit your game, and what you should or should not do.

Section 9 contains some examples of game elements satisfying each of the ten requirements. Remember that the list above specifies the *minimum* requirements. Feel free to add as many extra objects as you like!

7 Option B – The Tower of London. (60 points)

Remember that you should select either Option A or Option B, but not both.

If you don't want to design your own game, you can build the one described below. This game satisfies all the requirements listed for Option A. However, this option is not the easy way out! You still need to record proof that you have satisfied all requirements using the *behaviour recorder*, which is documented in section 3. It is up to you to figure out which objects satisfy which requirements.

The game is set in the Tower of London. Your job is to get past the guard and steal the crown jewels. You'll have to find some whiskey to bribe the guard, figure out a way to get rid of the cat, and dodge the tourists. A picture of the game is shown below:



As you can see, the different objects in the game are all numbered. The behaviour of each of these objects is described in detail in the list below:

1. These are tourists which move around the screen randomly. A tourists should turn red the first time it is shot, and die the second time it is shot.
2. This is a cat, which follows the player around. If the cat is more than 3 squares away, it tries to run after the player. The cat can run faster than the player, so there is no way to get away from it by running. However, the cat always runs directly towards the player, so it can be trapped behind walls.
3. This is a guard. When you bump into him, he will start a conversation with you. If you have the cat with you, the guard will tell you to get rid of the cat. If you don't have the cat with you, he will offer to step aside in exchange for a bottle of whiskey.
Hint: You can add a state variable to the player which shows whether or not the cat is near. The cat should set this variable to "false" whenever it is more than 3 squares away, and "true" whenever it gets close.

4. This is a portal which transports the player into the chamber which holds the crown jewels.
5. This is a doorbell. When the player pushes the doorbell, the door opens.
6. This is the door to the tourist shop, where you can steal whiskey.
7. These are bottles of whiskey.
8. This a box containing the crown jewels. If the player picks it up, the game is over.
9. This a pusher (see Section 5).
10. This a block (see Section 5).
11. This a slider (see Section 5).
12. This is a **Signpost** object which displays the following message: “Welcome to the Tower of London! Your job is to steal the crown jewels from the vault. The vault is protected by a guard who hates cats and loves whiskey. Be sure to avoid the tourists!”

8 Further Notes

- When you are recording proof for requirements R3, R4, R8, and R9, the variable in question must be visible in the *current state* section of the game window, as described in Section 2.3.2
- Be sure to use the functions described in Section 2.5.2 to generate random numbers, or you will be unable to record your game properly.
- Since the result of conversations are not recorded, you must put an asterisk (“*”) next to the responses that the marker should select, as described in Section 2.5.1
- It possible to create a single object which satisfies more than one requirement, provided that you record the appropriate proof using the behaviour recorder.
- You are allowed to use the classes you wrote in Section 5 in your game.
- Do *not* attempt to modify the game engine. There is a separate reference guide on the Infl web site which contains a detailed list of all classes and methods provided. Do not call any methods that are not documented either in this guide or in the reference guide.
- Remember to put a **Signpost** at the beginning of your game which describes the game and its objective.
- Remember to document your code and when you introduce a new field, to implement methods to update it appropriately.

8.1 Deliverables

Regardless of whether you complete option (A) or option (B), you must submit the following files:

- Your response to the learning questions, in a file named `answers.txt`.
- All of the java files that you modified, which should be.
 - `MyWorld.java`
 - `MyPlayer.java`
 - `MyLib.java`
 - `PusherLib.java`
- A recorded demo of how to play the game, called `gamedemo.txt`.

9 Ideas and Examples

Name	Example
R1	A creature which wanders around. It might wander completely randomly, or alternate between moving randomly and moving towards the player.
R2	A creature which when shot once, will turn red; when shot twice, will disappear.
R2	A wall that bullets bounce off of.
R3	A health potion which adds 10 to a variable called 'health'. When the player is shot or touches an enemy, 'health' decreases by 10.
R3	A box of ammunition adds 10 to a variable called 'ammo'. When the player shoots, 'ammo' decreases by 1.
R4	A door which will open if the player holds the key. The door depends on a boolean variable called 'hasKey'. Until the player has found the key, the value of 'hasKey' is false .
R5	A doorbell and a door: when the doorbell is punched, the door opens.
R5	A number of guards: when one guard is shot, all of the other guards wake up and attack.
R5	A golden idol and a temple: when you steal the golden idol, the gate starts to close.
R5	A tripwire and gun: when you step on the tripwire, a gun shoots.
R6	A proximity alarm: when you come within a 2 block radius of the golden idol, it goes off. When you move out of the 2 block radius, it stops again.
R6	A thief which always runs away from the player.
R6	A "smart" creature that only shoots when it's lined up with the player.
R7	An electric fence: if the player runs into it, (s)he get flung back four squares.
R7	A secret portal: if you go into it, you end somewhere completely different in the game.
R8	The creature wants to sell a password to the player, but if the player does not have enough money, the deal is off - i.e., the creature depends on the player variable 'money'.
R9	During the conversation, the player buys something from the creature - i.e., the player variable 'money' decreases.
R10	A pot of gold: if the player reaches it, the game is complete.

Appendix D

Communicating Objects Assignment (2006–2007)

Inf1B Assignment 4 (OOP)

Object Interactions

Hutchins, Korte

Out: February 16th, 2007

Due: March 2nd, 2007

Materials: Download `assignment4.zip` from the informatics 1 website:

```
http://www.inf.ed.ac.uk/teaching/courses/inf1/oop/assignments/  
assignment4.zip
```

Submission: Type the following commands at the UNIX shell prompt to submit this assignment:

```
cd workspace/Game  
zip a4.zip oop4/*.java  
submit inf1 inf1b oop4 a4.zip  
submit inf1 inf1b oop4 myWorld.xml  
submit inf1 inf1b oop4 answers.txt demo.txt
```

Research: Laura Korte, whom many of you will know from Inf1B lab-sessions and tutorials, is doing her PhD research on the topic of *learning by game-building*. She would like to use your submissions to this assignment for research purposes. Please note that the identity of participants will remain anonymous. Furthermore, any document containing results of this research project will be presented in such a way that no individual can be identified.

If you wish for your assignment to be excluded from this research project, please mention this in "answers.txt". Also, if you have any further questions, please email Laura at orange.laura@gmail.com.

The fine print: If your code does not compile, 30% will be deducted off of your final score. Bad indentation will be penalized, up to a maximum of 10 points off the total score. Late submissions will not be accepted. Collaboration with other students on this assignment is not permitted. If you have any questions on this assignment, please post them to the newsgroup `eduni.inf.course.inf1b` or ask a demonstrator in one of the drop-in lab sessions.

1 Introduction

This is the second half of a two-part assignment on computer games. For this assignment, you must choose one of two options. For option (A), you are given a pre-defined game to create called “Edinburgh Castle”. For option (B), you have the opportunity flex your creative skills, and design your own game, on whatever topic you like! Both options are equally difficult, and will be marked in exactly the same way.

1.1 Inf1B Game Programing Competition

If you select option (B), you have two additional choices. You may (1) choose to publish your game on the Inf1B website, and you may (2) choose to enter it in the Inf1B game programming competition. (You must publish your game in order to enter it in the competition.) For the competition, we will set up a voting form where you can vote for the best game in the class. Winners will be announced before the end of term. In addition to the peer-judged competition, the markers will also be awarding “markers choice” awards to games that they feel show exceptional creativity.

To publish your game on the website, you should give it a name, write up a brief description, and make a screenshot of the game. You can use a program called “gimp” on the DICE machines to take the screenshot. (Choose `file->acquire->screenshot`.) Put the name and description in a file called `description.txt`, and save the screenshot as `screenshot.jpg`. Submit those two files along with the rest of the assignment and we will make it available for other students to download and play.

To enter your game in the competition, just say so at the top of the file `description.txt`. If you do choose to enter your game in the programming competition, be aware that writing a clever game, even a simple one, takes a *lot* of time. I suggest that you start on this assignment immediately. There is no D&A assignment due next week, so you have two weeks of uninterrupted time.

1.2 Marking and Submission

Options A and B will both be marked in the same way. Your game must fulfill 10 different requirements. To prove that your game fulfills them, you must record a demo of yourself playing the game from start to finish. As you play the game, you can describe how each requirement is fulfilled. *Note that you must still record a demo, even if you choose to write the pre-defined game.*

We would like to encourage you to put interesting plots and difficult puzzles in your game. One advantage of recording a demo is that the markers do not have to actually solve your puzzles in order to mark them. You are free to make your game as clever or as difficult as you wish, without worrying about whether or not the markers can figure it out.

Java files: You must download a new version of the game engine for this assignment; the previous version will not work. All java files that you write should be put into a package named “oop4”, and then zipped together for submission.

Modifying the game engine: You are not allowed to modify the game engine in any way, including any of the files in the “engine”, “gui”, or “widgets” subdirectories.

Reusing code: Feel free to reuse any of the classes that you wrote in the previous assignment or in Lab4 as part of your game. However, you cannot use those classes to fulfill any requirements. In addition, you should copy any java files that you reuse from the “oop3” or “lab4” package to “oop4” so you can submit them together.

2 Reflection on Your Learning (5 points)

Before you submit your game, you must answer some questions as a reflection on your learning. There are no right or wrong answers. There are only answers that are true of you, and answers that are not true of you. For each of the following 16 statements, your answer should consist of a number between 1 (not at all true of me) and 7 (very true of me). Please save your answers in a file named "answers.txt".

(1)	My goal this semester is to get better grades than most of the other students.	1	2	3	4	5	6	7
(2)	I just want to avoid doing poorly compared to other students this semester.	1	2	3	4	5	6	7
(3)	Completely mastering the material in my courses is important to me. this semester.	1	2	3	4	5	6	7
(4)	I really don't want to work hard in my classes this semester.	1	2	3	4	5	6	7
(5)	I'm afraid that I may not understand the content of my courses as thoroughly as I'd like.	1	2	3	4	5	6	7
(6)	It is important for me to do well compared to other students this semester.	1	2	3	4	5	6	7
(7)	I want to learn as much as possible this semester.	1	2	3	4	5	6	7
(8)	The fear of performing poorly is what motivates me.	1	2	3	4	5	6	7
(9)	I want to do as little work as possible this semester.	1	2	3	4	5	6	7
(10)	The most important thing for me this semester is to understand the content in my courses as thoroughly as possible.	1	2	3	4	5	6	7
(11)	I worry that I may not learn all that I possibly could this semester.	1	2	3	4	5	6	7
(12)	I want to do better than other students this semester.	1	2	3	4	5	6	7
(13)	I want to get through my courses by doing the least amount of work possible.	1	2	3	4	5	6	7
(14)	I am definitely concerned that I may not learn all that I can this semester.	1	2	3	4	5	6	7
(15)	My goal this semester is to avoid performing poorly compared to other students.	1	2	3	4	5	6	7
(16)	I look forward to working really hard this semester in my coursework.	1	2	3	4	5	6	7

3 Requirements (95 points)

Your game must provide entities which fulfill the following requirements:

Name	Description	Credit
R1	An entity which depends on the position of the player.	10 points
R2	An entity which modifies a <i>new</i> state variable (i.e. field) of the player. This must be a new state variable which has been added by you, and the state must be visible in the player status area on the right-hand side of the screen.	10 points
R3	An entity which depends on a <i>new</i> state variable (i.e. field) of the player.	10 points
R4	A conversation which modifies the state of the player.	10 points
R5	A conversation which depends on the state of the player.	10 points
R6	An entity which creates another entity.	10 points
R7	An entity which sends a signal to another entity of the same class. This should be a new signal that has been added by you. You cannot use an existing event mechanism such as being pushed, shot, etc.	10 points
R8	An entity which sends a signal to another entity of a different class. This should be a new signal that has been added by you. You cannot use an existing event mechanism such as being pushed, shot, etc.	10 points
R9	An entity which depends on the state of another entity. The other entity cannot be the player, and its state should be clearly visible as a change in symbol or color.	10 points
R10	A victory object: if the player pushes or picks up this object, then the player should be informed that he/she has won the game.	5 points

An entity can be used to fulfill more than one requirement. However, the entities that fulfill requirements R2 and R3 must be different from those that fulfill R4 and R5.

Section 6.3 describes how to use conversations. Section 6.5 describes how to send signals between objects. You may also wish to review OOP Lab 4 to see how to send signals and add new entities to the board.

4 Option (A) – Edinburgh Castle

If you don't want to design your own game, you can make a game called "Edinburgh Castle". Note that you still have to create the game board yourself, and record a demo. You must also figure out which entities satisfy which requirements. This option merely saves you the trouble of designing a game, it does not save you the work of implementing it.

The game board for "Edinburgh Castle" is shown below. A playable version of the game is also available on the Inf1 website, if you are uncertain about any details.



The various entities found in this game are as follows, although exact details are omitted. In general, you should play a sound and/or print a message whenever something interesting happens.

- The signpost at the beginning gives an introduction to the game.
- The signpost in the chapel describes Margaret's chapel, and mentions that its a good place to ditch the cat.
- The ticket agent (T) offers to sell the player a student ticket to the castle for £87.50, as part of a conversation. The player has no money, and cannot accept this offer.
- The cat (c) will run after the player if it is more than 3 squares away. It always runs in the direction of the player, but can move around some obstacles.
- The guard (G) hates cats, and will complain if the cat is within a radius of 8 squares. If the cat is far away, he will ask for a dram of whisky. If the player has whisky and offers to drink it with him, then he will move aside and let the player pass.
- The player (class MyPlayer – happy face) can be either sober or drunk. After drinking with the guard, it becomes drunk. After walking 25 paces, it becomes sober again. While drunk, the player cannot walk in a straight line; it has a 1 in 3 chance of moving in a random direction, rather than the direction that it is supposed to go in. The player dies immediately with a gruesome death message if it is shot.
- The whisky shop owner (happy face) will advertise his whiskies whenever the player touches him.
- The whisky bottles (!) cannot be picked up unless the player has £25 to pay. If the player has enough money, they will subtract £25 and give 1 bottle of whisky.
- The tourists (t) move around randomly. 75% of the time, they continue walking in whatever direction they have been moving. 25% of the time, they change to a random direction. They also change direction if they bump into a wall or other immovable object.

Tourists have two states, happy and frightened. If they are happy, they are colored cyan, and move around slowly at speed 6. If they are frightened, they are colored

white, and run around madly at speed 2. After being frightened for 4 seconds, they will change to happy again. If a frightened tourist bumps into a happy tourist, the happy tourist becomes frightened.

When a tourist is shot, it dies and drops a 5-pound note. As it dies, it screams, thus causing all tourists within a radius of 5 squares to become frightened.

When a tourist bumps into the player, it pops up an annoying conversation about castles if it is happy, or an annoying conversation about madmen if it is frightened.

- A five-pound note (£) will give the player £5.
- The security sensors (grey tile) will alert the security walls if the player steps on them.
- The security walls will send out a constant stream of bullets if they are triggered by the security squares.
- The Honours of Scotland (star) will show a “You Win” conversation.

5 Option (B) – Making Your Own Game

If you make your own game, you may choose to satisfy the requirements in whatever way you want. Make sure to put a signpost at the beginning of the game to introduce the setting, and tell the player how to play the game, and what they are supposed to do. Here are some ideas to get you thinking:

- (R1) Enemies which move towards the player, or guns which shoot when the player is within range.
- (R2) Any item which can be picked up: ammunition, health, invincibility potion, money, keys, etc.
- (R3) A door which requires a key, or a purchase which requires money.
- (R4) A shopkeeper or vending machine.
- (R5) A shopkeeper or vending machine.
- (R6) An enemy which reproduces, or a monster which drops treasure.
- (R7) A guard which calls for help, or a lift made up of several entities that move together.
- (R8) A button which opens a door, or a sensor which sets off a trap.
- (R9) A car which stops at a stoplight, or a guard that can open closed doors.
- (R10) A pot of gold, a prince/princess, or the exit.

6 Game Engine

This section includes some additional information on the game engine that you need to know.

6.1 Random Numbers

The function `randomInt(n)` will return a random number between 0 (inclusive) and n (exclusive). The function `Direction.random()` will return a random direction.

You should *not* use the standard Java libraries to generate random numbers, because you will not be able to record a proper demo.

6.2 Getting the Player

The expression `getPlayer()` will retrieve a reference to the current player. However, `getPlayer()` returns a reference of type `Player`. You must use a class named `MyPlayer` which extends `Player` in order to fulfill some of the requirements. To get a reference of type `MyPlayer`, you will have to downcast, as shown in the following example:

```
MyPlayer p = (MyPlayer) getPlayer();
```

6.3 Messages and Conversations

There are three ways in which entities can communicate with the user. They can play print a message to the message area at the bottom of the screen, they can play a sound, or they can start a **Conversation**. Simple messages that consist of a single line of text can be shown with the following command:

```
public void showMessage(String message)
```

If an entity needs to show more than one line of text, or if it needs to ask the user a question, then it should use a *conversation*.

A conversation has two parts. It has a block of text which can be any length. The text is HTML, so it can include bold, italic, tables etc. to display information. In addition to the text, a conversation may have one or more buttons that the user can click on to make a response.

There are two kinds of buttons that a conversation can have. A *return button* will close the conversation window, and return a string as an answer. A *continue button* will jump to another part of the conversation, which may have different text and/or response buttons. The following code snippet shows how to make a knock-knock joke:

```
Conversation joke1 = new Conversation();
Conversation joke2 = new Conversation();
Conversation joke3 = new Conversation();

joke1.setText("Knock knock...");
joke1.addReturn("Go Away!", "away");
joke1.addContinue("Who's there?", joke2);

joke2.setText("Anna.");
joke2.addReturn("Go Away!", "away");
joke2.addContinue("Anna who?", joke3);

joke3.setText("Anna body know a better joke?");
joke3.addReturn("Ha Ha!", "funny");
joke3.addReturn("That's stupid!", "stupid");

String response = showConversation("The Joker Speaks:", joke1);

if (response.compareTo("funny") != 0) {
    getPlayer().damaged(this);
}
```

There are three parts to the joke – the start, the middle, and the punch line. Each part of the conversation continues on to the next part. Once the punch line has been delivered, the user has two choices. Clicking on the <Ha! Ha!> button will end the conversation and return the string "funny", while clicking on the <That's stupid!> button will return the string "stupid". An entity can then test this answer and decide what to do next. In this case, the player will get damaged if he or she does not like the joke.

Important note: when comparing strings in Java, you should use `compareTo` rather than `==`. The `==` operator tests for reference equality, rather than value equality, and will thus give the wrong answer.

6.4 Playing Sounds

The following command will play a sound clip:


```
public void playSound(String filename)
```

Note that under linux, there is an annoying half-second delay before the sound clip will start playing. This is because Java does *not* use the built-in system mixer on linux machines, but instead uses an old (and slow) java-based mixer. There is no way to solve this problem, short of re-implementing a mixer by hand. Under windows, java uses the system mixer, and there is thus no delay.

The file name must be the name of a sound file in the “sounds” subdirectory. A number of sounds are already provided for you to use, but you are welcome to add your own. If you do add your own, be sure to include your sounds in `a4.zip` when you submit your game. Do *not* submit any sounds that are already provided. In order to save memory, you should save your sounds at low quality; 22kHz mono is adequate for an ASCII-text game.

You can make sounds with any program you want. I personally recommend a program called Audacity, which runs under both linux and windows, and is freely available for download. You can also download sound effects, although free sound effects are somewhat difficult to find. If you choose to download sound clips, be careful to read the fine print and make sure that it is legal to republish them. I can recommend the following sites:

- www.therecordist.com/pages/downloads.html
- www.partnersinrhyme.com/pir/PIRsfx.shtml
- freesound.iua.upf.edu/

You will not receive any credit for sound effects, so do not spend too much time trying to make good ones. You can add clever sounds after your game has been completed.

6.5 Sending Signals to Other Entities

In order to satisfy requirements R7 and R8, you need to send a signal to another entity, without using pushes, bullets, or any other built-in event of the game engine. Typically this means that one entity must signal another one that is far away. For instance, a doorbell may cause a door somewhere else to open, or a bomb may send a “bombed” event to nearby entities when it explodes. See OOP Lab 4 for details.

In these cases, you will need to find the other entities in order to send them a message. The command `findEntities(class)` will search for all entities in the world which have the given class. The `findEntities` method returns an iterable collection, which is much like a list in Haskell. Java 1.5 provides a special syntax for dealing with iterators called the extended for loop, which is shown below.

```
for (Entity e : getBoard().findEntities(Spider.class)) {
    ((Spider) e).doSomething();
}
```

The above loop will find all instances of the `Spider` class on the current board, and will call the `doSomething()` method on all of them. Notice that you must downcast from type `Entity` to type `Spider`.

7 Using the Editor

The editor is not very sophisticated, but it should provide everything you need to make a game. To start the editor, select `File->Edit World` from the main menu. Save your game as “myWorld.xml”.

To place a new entity on the board, select an entity from the list on the left, then hold down `<Shift>` and left-click on a game square. Remember that each tile can hold one Thing, and any number of Effects.

To modify an entity, click on the entity to select it. All of the editable attributes of the entity are shown in the lower left corner. The “editable attributes” of an entity are those attributes which have `get/set` methods defined. Changing an attribute in the editor will call the appropriate `set` method on the entity.

You can change one of the attributes by typing in some text, and then typing `<Enter>`. Attributes can be integers, floats, booleans, strings, directions, or colors. All of the standard colors (See `java.awt.Color`) are represented by strings – e.g. “red”, “cyan”, “magenta”, etc. Valid directions are “north”, “south”, “east”, “west”, and “here”.

If you repeatedly click on the same square, it will select all of the entities on that square in sequence, including both things and effects.

To delete an entity, select it and press `<Delete>`.

To select a region of the game board, hold down the left mouse button and drag. The editor supports the standard operations of cut (`Ctrl-X`), copy (`Ctrl-C`), paste (`Ctrl-V`), and delete.

To create text labels, lay down a horizontal row of `basic.Letter` entities, and then select the first one. When a letter is selected, hitting a key will change the text of that letter to the key, and then move the selection to the next letter.

To change the size of a board, click on `Edit->Board Properties`. If you choose to make your own board classes by inheriting from `Board`, you can also set the class of the board here as well. Change the board class to `basic.DarkBoard` to see an example of a custom board class.

7.1 Multiple Boards

Your game may have multiple boards, although it is not required to. Each board must have a unique name. You can add a new board with `Edit->Add Board`, and switch between boards with the `Board` menu. The starting board for the game is the one with the player on it, so only one board should have a player.

You can connect boards together by using the `basic.Portal` entity. If the player steps onto a portal, it will be transported to the matching portal of the same color on a different board. Alternately, each board may have other boards to the north, south, east and west. If the player walks off the edge of the current board, it will switch to the new board. Both portals and board edges refer to other boards by name, so make sure that you type in the names correctly.

7.2 Potential Problems

If your entity classes don’t show up on the entity list, make sure that they have a `public` constructor with no arguments.

If your attributes don’t show up, make sure that the types of the `get/set` methods match, and that the methods are declared `public`.

8 Recording a Demo of your Game

To record a demo, click on `Demo->Show Demo Control`. That control will open up the window shown below:



Press <Record> to start recording a demo, and then play the game as normal. As you play the game, you will interact with certain objects that fulfill certain requirements.

Right before an important interaction, press the 'p' key to pause the game, then click on one of the buttons labeled "Begin Rn" where *n* is the number of the requirement that the interaction will fulfill. Pressing this button will pop up a dialog box, where you are asked to enter a short description of how the interaction fulfills the requirement.

Once you enter a description, the button will change from "Begin Rn" to "End Rn". Once the interaction is over, you should press "End Rn" to finish the requirement.

All of the descriptions will be logged to the recorder window, so you can see what you have entered. After you have completed a requirement (both the begin and the end), the button for that requirement will change to "Done Rn", so you know which ones are still left.

The <Play> button will play back the demo, and pause the game at the start of each requirement. When you have played to the end of the game, choose Demo->Save from the main menu, and save it under the name "demo.txt". The demo file is an ordinary text file, so if you want to edit some of your descriptions without recording a whole new demo, then you can do so with any text editor.

9 Good Luck, and Have Fun!

Appendix E

Finite State Automata Assignments

Inf2A: Assignment 1

Adventure Games through FSAs

– Version A –

ANDERSON, KORTE

Week 3 (2 October 2006)

If you have any questions regarding this assignment, please post them to the newsgroup `eduni.inf.course.inf2a`, or ask a demonstrator during one of the lab sessions. Please note that your code should compile in < 1 minute on a standard DICE machine, or your assignment will not be marked. Most games will satisfy this requirement *if* the `minimize` function is used appropriately. Please also remember to always document your code appropriately. The deadline for this assignment is **Friday 13 October, 4pm**. You should spend a maximum of 25 hours on this assignment; remember that you have work for other courses as well. Happy programming!

*In this assignment you will be writing an adventure game using finite state automata and their combinators. Section 1 of the assignment consists of two parts. Part 1 introduces you to the Python module FSA and the task of programming adventure games. In Part 2 you will be programming an adventure game. Note that questions which have a * at the start are only there for you to check if your answers are correct. No answer is expected and consequently, no marks are given for the questions starting with *. However, checking is important since if you go ahead to the next section without having passed the checks for a particular section, your chances of success are lower.*

You should put your answers in two files (`Intro.py` and `Game.py`). Only files with those exact names will be accepted by the `submit` program. You should put all your answers to the programming questions in Section 1.1. into `Intro.py`. Similarly, you should put your game, according to the specification in Section 1.2. into `Game.py`. Put both files into a single directory and from that directory, submit them using the following command:

```
submit inf2 inf2a A1 Intro.py Game.py
```

Laura Korte, whom many of you will know from Inf1B lab-sessions and Inf1B and Inf2A tutorials, is doing research on the topic of knowledge modeling through game building. She would like to use your submissions to this assignment for research purposes. Please note that the identity of participants will remain anonymous. Furthermore, any document containing results of this research project will be presented in such a way that no individual can be identified. If you wish for your assignment to be excluded from the research, please mention this as a comment at the top of your `Intro.py` file. Also, if you have any further questions, please email Laura at `orange.laura@gmail.com`.

1 Programming

1.1 Game Preliminaries (40 marks)

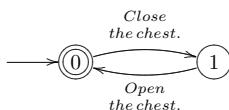
For this assignment, you will be using the Python module `FSA`. You can find it and its API in the main menu of the Assignment 1 website, which is linked from the Informatics 2A website. Download the module and include the following statement at the beginning of your program:

```
import FSA
```

In questions 1–4 you will be building an increasingly large FSA, which models a simple game. In question 5 you will learn how to incorporate information about the game environment into the FSA. Save all of your answers to these questions in a single file and call it `Intro.py`. Remember that you can test your FSAs interactively using the following command:

```
>> python -i Intro.py
```

1. First we will model a room with nothing in it but a chest. The chest can be opened and closed. For now, the goal will be to have the chest open. Here is the FSA, which we will refer to as `chest`, that corresponds to this description:



And here is the 5-tuple that corresponds to the FSA:

```
{{0, 1}, {"Open the chest.", "Close the chest."}, {(0 → 1, "Close the chest."), (1 → 0, "Open the chest.")}, 0, {0}}
```

Remember that the 5-tuple contains the following information: $\{\text{States, Alphabet, Transitions, Start State, Final States}\}$.

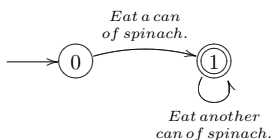
- (a) Create an equivalent FSA in Python using only `FSA.singleton`, `FSA.concatenation` and `FSA.closure` and call it `chest`.
- (b) If you use the `view()` method to examine your FSA, you will find that it looks quite different from the one printed above. Use the `FSA.minimize` function to remedy this and call the result `min_chest`. Use `view()` to check your result.

As you know, the operators on FSAs introduce many states that increase the size of the automaton. Most of these states can be eliminated to create an equivalent FSA. As you do this practical you will notice that as you construct more complex automata, they begin to perform more slowly. The `minimize` function eliminates the redundant states and after applying it to your automaton you will find it will operate much more quickly.

- (c) * Check if your answer is correct using the `checkQ1()` method.

2. Next, we will extend the room with a large collection of cans of spinach hidden inside the chest. A player needs the spinach to become strong enough to get to the next level. The player can eat the spinach when the chest is open. It should be impossible to eat the spinach when the chest is closed.

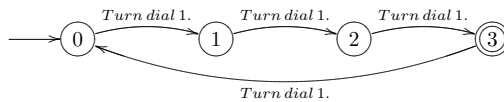
Here is the FSA, which we will refer to as `spinach`, which models the behaviour of the spinach:



- (a) Create `spinach` using only `FSA.singleton`, `FSA.concatenation`, `FSA.closure`, and possibly `FSA.minimize`.

- (b) * Check if your version of `spinach` is correct using the `checkQ2a()` method.
 - (c) Using both `chest` and `spinach`, create an FSA in Python with the behaviour specified above using `FSA.singleton`, `FSA.concatenation`, `FSA.closure`, `FSA.preInterleave`, and possibly `FSA.minimize`. Call it `spinach_in_chest`. *Remember*, you can not take the `spinach` out of the chest if it is closed!
 - (d) * Check if your version of `spinach_in_chest` is correct using the `checkQ2c()` method.
3. The next thing we will add to our room is a combination lock. It consists of three dials, each of which can be turned (clockwise) to the values 0, 1, 2 and 3. The secret combination is 3-0-1 (i.e. dial 1 is turned to 3, dial 2 is turned to 0, and dial 3 is turned to 1). This combination lock will be on the door which gets you to the next level. The initial configuration of all three dials is 0.

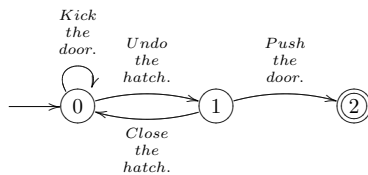
Here is the FSA, which we will refer to as `dial1`, which models the behaviour of the first of the three dials:



Notice that state 3 is the final state, reflecting that this is the correct position for the first dial.

- (a) Create an FSA in Python for `dial1` using only `FSA.singleton`, `FSA.concatenation`, `FSA.closure`, and possibly `FSA.minimize`.
 - (b) Do the same for the other two dials. Call them `dial2` and `dial3`.
 - (c) Use `dial1`, `dial2` and `dial3` and the appropriate function from the FSA module to create the combination lock which should be called `cLock`. *Remember that you can use the `minimize` function when your FSA is starting to get slow!*
 - (d) * Check if your version of `cLock` is correct using the `checkQ3c()` method.
 - (e) Create an FSA for a room which has in it: (1) a chest full of cans of spinach, and (2) a 3-dial combination lock. Call this room `level1`. *Note that `spinach_in_chest` and `cLock` are independent objects!*
 - (f) * Check if your version of `level1` is correct using the `checkQ3e()` method.
4. Our final addition to the game will be another room, or in more general terms: another game level. This second room will be a simple one, with only a door to the next level in it. The player can reach the room with the action `nextLevel1`, if (s)he has (a) eaten some spinach, and (b) turned the dials to the right combination. The Python FSA module contains a function called `composeLevels` which performs level composition on an arbitrary number of levels with an intermediate `nextlevel` action (see also the API).

Here is the FSA, which we will refer to as `level2`, which models the behaviour of the second level:



- (a) Create an FSA in Python for `level2` using only `FSA.singleton`, `FSA.concatenation`, `FSA.closure`, `FSA.union`, and possibly `FSA.minimize`.
- (b) * Check if your version of `level2` is correct using the `checkQ4a()` method.

- (c) Create an FSA in Python for the composition of `level1` and `level2` using `composeLevels` and call it `game1`.
- (d) Create a second FSA with the specified behaviour using any of the FSA combinators, except `composeLevels`. Call it `game2`.
- (e) * Check if the two game FSAs are equivalent using the `equivalent` function.
- (f) * Check if your versions of `game1` and `game2` are correct using the `checkQ4cd()` method.

In the next part of the assignment, you will use the FSAs you have created as the basis for a playable game. You could try play the game you just wrote as it is, using to following command: `FSA.play(game1, 'Introduction', 'Congratulations!', '>>')` (type `Goodbye!` to exit). However, since no descriptive information has been added yet, the game can be rather confusing. For example, the player has no idea that the spinach is inside the chest, so imagine his surprise when it suddenly disappears when he closes the chest!

In question 1a you should have used two different calls to `singleton`: `singleton('Open the chest.')` and `singleton('Close the chest.')`. A call to `singleton` *always* results in an FSA with two states. We will now supply some information for each of these states in both of the calls: `singleton('Open the chest.', 'You see a chest. It is closed.', 'You see a chest. It is open. There are several cans of spinach in it.')` and `singleton('Close the chest.', 'You see an open chest with spinach in it.', 'You see a closed chest.')`

Notice that a list of states and their description appears every time you call the method `view()`.

5. (a) Create a second FSA in Python modelling `chest`, but this time use the new calls to `singleton`. Call the new FSA `i_chest_d`.
- (b) After minimization, you can look at your FSA using the method `view()`. You will notice that there seems to be some redundant information in the state descriptions. Why is that? Write your answer as a comment at the top of your `Intro.py`.
- (c) Using your answer to the previous question, create a third FSA in Python modelling `chest`. Call the new FSA `i_chest`. It should give the same descriptive information as `i_chest_d`, but without redundancy.

1.2 The Game - Version A (30 marks)

Now that you have experimented with building a small adventure game using the FSA building blocks, you are ready to start programming a complete adventure game. The game you will be writing is called 'The Weirdest Link'. Here is the introduction:

Welcome to the Weirdest Link! In this game, your goal is to raise enough money to buy a yacht so you can fulfill your dream of sailing around the world. You have decided to raise the money by entering yourself in the popular tv show The Weirdest Link. In the 1st level, you will have to fill out the entry exam and mail it. In the 2nd level you will be on the show trying to win enough money, and in the 3rd you will hopefully get to sail around the world!

As mentioned in the introduction, the game consists of 3 different levels. In the first level the players needs to answer 5 questions on the entry exam sheet and then mail it when (s)he is done. Because it is a paper exam, the questions can be answered in any order. Also, if a mistake is made, the player can erase an answer and then try again. In the second level the player is on the TV show 'The Weirdest Link' in which (s)he has to answer 5 questions in sequence. If a mistake is made, the player goes back to the beginning. Once the player has answered all 5 questions correctly, (s)he can either bank the money that you have earned and go the next level, or start again from the beginning. In the third level the player is trying to buy a yacht on which (s)he can sail around the world. First the computer needs to be turned on, then the player needs to go the the eBay site to make a bid on a

yacht. Once the yacht is bought, (s)he leaves for the harbour, hires a sailing instructor (because our player is ignorant when it comes to sailing), and sets off for the ocean! There is a playable version of the game available from the Assignment 1 website, which is linked from the Informatics 2A website, so you can better understand the exact behaviour required for our game 'The Weirdest Link'.

Your game will be marked according a number of requirements. This is the complete list of requirements:

(Behaviour 1) It takes the player at least 15 actions to reach the goal.
(Behaviour 2) It is always possible to reach the goal.
(Behaviour 3) Your game contains infinite paths.
(Behaviour 4) Your game contains at least 1 'long' cycle (i.e. at least 5 states before a repeat).
(Behaviour 5) Within at least 1 level it is always possible to reach the initial state (of the level) with 1 step. Note that it is not required that the initial state can be reached from the *final* states of the level.

(Structure 1) Your game consists of at least 3 levels.
(Structure 2) your game has at least 1 union of size ≥ 5 .
(Structure 3) Your game has at least 1 interleave of size ≥ 5 .
(Structure 4) Your game has at least 1 precondition interleave.
(Structure 5) Your game has at least 1 nested precondition interleave (i.e. of the form (FSA1 |> (FSA2 |> FSA 3))). NB. These *cannot* be the same ones used for structural constraint 4!

Furthermore, marks will be deducted if your game does not satisfy the following requirements:

(Negative 1) Your code should be well documented.
(Negative 2) You should provide at least one information string – for initial state and/or final state – with each call to `singleton`.
(Negative 3) Your code should compile in less than 1 minute on a standard DICE machine.
(Negative 4) Your variables should have informative names.
(Negative 5) Your levels are composed using `composeLevels`.

Please remember that you should not spend more than 25 hours on this assignment!

6. Implement the adventure game in Python using the combinators in the FSA module, store it in a variable and call it `gameFSA`. Your game should satisfies all the given behavioural, structural, and negative requirements. Call the file that contains your game `Game.py`.
7. Copy the introduction of the 'The Weirdest Link' into a variable and call it `intro`. You should also copy the message of congratulations (which will be shown to you when you finish playing the game), store it into a variable and call it `congrats`.
8. * Check that your game is playable with the following statement:
`FSA.play((gameFSA,intro,congrats,"Yes? "))`
9. * Check that your game satisfies all of the behavioural constraints using the `behaviour1()` – `behaviour5()` methods. Note that you will have to invoke some of the methods on the actual game FSA, and others on the FSAs representing the individual levels.

2 Submitting your Assignment

When you have finished sections 1 and 2, you are ready to submit this part of your assignment. Please note that your code should compile in < 1 minute on a standard DICE machine, or your assignment will not be marked. Most games will satisfy this requirement *if* the `minimize` function is used appropriately. Please also remember to always document your code appropriately and check to see that you have two files called `Intro.py` and `Game.py`. Only files with those *exact* names will be accepted by the `submit` program. Put these three files into a single directory and submit them using the following command from a terminal window:

```
submit inf2 inf2a A1 Intro.py Game.py
```

Remember that the questions on this handout are only worth 70/100 marks and that you still need to answer some theory questions on the Assignment 1 website for the remaining 30 marks!

THE END

Plagiarism

There are some general principles in preparing any work:

- You should complete coursework yourself, using your own words, code, figures, etc.
- Acknowledge your sources for text, code, figures etc that are not your own.
- Take reasonable precautions to ensure that others do not copy your work and present it as their own.

If you produce your own unassisted work then you are in no danger of plagiarising. However, if your work requires some help or collaboration then you should follow the guidelines below.

It is acceptable to use general ideas picked up, for example, in discussion with others, or in textbooks, without acknowledgement. In general, if you write the work you submit, in its entirety, yourself then you do not need to include an acknowledgement. An exception is when a really pivotal idea comes from an external source, in which case you should acknowledge that source.

There are occasions on which you will wish to include work written by somebody else within your own work. For example, you may be quoting a passage from a book within an essay, or you may be unable to finish a program yourself and require a piece of code written by someone else. On such occasions, you should copy the original work verbatim together with a full and explicit acknowledgement of the original source. The beginning and end of the copied material must be clearly indicated:

- In the case of copied text (in an essay or written answer to a question), you should enclose the text within quotation marks and explicitly indicate the source of the work close to where the copied material appears.
- In the case of code, you should insert a comment right before the start of the code you have copied or collaborated on, indicating who you have gotten it from or who you have collaborated with in developing it, and then another comment right after the end of the relevant code.

These rules about acknowledging the source of work that is not fully your own, apply irrespective of the type of work it is (prose, code, etc.).

If you collaborate on the preparation of all or part of a piece of work then the collaboration must be explicitly acknowledged, and the extent of the collaboration should be stated as fully as possible by all parties involved.

It is always unacceptable to attempt to disguise someone else's work in order to present it as your own, for example by making cosmetic changes to text, or by changing variable names or altering comments in code.

When in doubt, state explicitly what you have done and the sources of any help (including fellow students, tutors or textbooks) you receive. This will enable your work to be marked appropriately, and will ensure that you avoid suspicion of plagiarism.

In order to prevent others from copying your work and presenting it as their own, set the protection on your practical work directories appropriately. It is also sensible to ensure that any practical submissions that you print are collected promptly from the printers. By all means discuss your general ideas with other students, but do not distribute or lend your solutions to them.

This text was adapted from "Guidelines on Plagiarism" at <http://www.inf.ed.ac.uk/admin/ITO/DivisionalGuidelinesPlagiarism.html>.

Inf2A: Assignment 1

Adventure Games through FSAs

– Version B –

ANDERSON, KORTE

Week 3 (2 October 2006)

If you have any questions regarding this assignment, please post them to the newsgroup `eduni.inf.course.inf2a`, or ask a demonstrator during one of the lab sessions. Please note that your code should compile in < 1 minute on a standard DICE machine, or your assignment will not be marked. Most games will satisfy this requirement *if* the `minimize` function is used appropriately. Please also remember to always document your code appropriately. The deadline for this assignment is **Friday 13 October, 4pm**. You should spend a maximum of 25 hours on this assignment; remember that you have work for other courses as well. Happy programming!

*In this assignment you will be writing an adventure game using finite state automata and their combinators. Section 1 of the assignment consists of two parts. Part 1 introduces you to the Python module FSA and the task of programming adventure games. In Part 2 you will be programming an adventure game. Note that questions which have a * at the start are only there for you to check if your answers are correct. No answer is expected and consequently, no marks are given for the questions starting with *. However, checking is important since if you go ahead to the next section without having passed the checks for a particular section, your chances of success are lower.*

You should put your answers in two files (`Intro.py` and `Game.py`). Only files with those exact names will be accepted by the `submit` program. You should put all your answers to the programming questions in Section 1.1. into `Intro.py`. Similarly, you should put your game, according to the specification in Section 1.2. into `Game.py`. Put both files into a single directory and from that directory, submit them using the following command:

```
submit inf2 inf2a A1 Intro.py Game.py
```

Laura Korte, whom many of you will know from Inf1B lab-sessions and Inf1B and Inf2A tutorials, is doing research on the topic of knowledge modeling through game building. She would like to use your submissions to this assignment for research purposes. Please note that the identity of participants will remain anonymous. Furthermore, any document containing results of this research project will be presented in such a way that no individual can be identified. If you wish for your assignment to be excluded from the research, please mention this as a comment at the top of your `Intro.py` file. Also, if you have any further questions, please email Laura at `orange.laura@gmail.com`.

1 Programming

1.1 Game Preliminaries (40 marks)

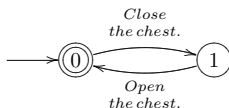
For this assignment, you will be using the Python module `FSA`. You can find it and its API in the main menu of the Assignment 1 website, which is linked from the Informatics 2A website. Download the module and include the following statement at the beginning of your program:

```
import FSA
```

In questions 1–4 you will be building an increasingly large FSA, which models a simple game. In question 5 you will learn how to incorporate information about the game environment into the FSA. Save all of your answers to these questions in a single file and call it `Intro.py`. Remember that you can test your FSAs interactively using the following command:

```
>> python -i Intro.py
```

1. First we will model a room with nothing in it but a chest. The chest can be opened and closed. For now, the goal will be to have the chest open. Here is the FSA, which we will refer to as `chest`, that corresponds to this description:



And here is the 5-tuple that corresponds to the FSA:

```
{{0, 1}, {"Open the chest.", "Close the chest."}, {(0 → 1, "Close the chest."), (1 → 0, "Open the chest.")}, 0, {0}}
```

Remember that the 5-tuple contains the following information: $\{\text{States, Alphabet, Transitions, Start State, Final States}\}$.

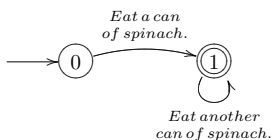
- (a) Create an equivalent FSA in Python using only `FSA.singleton`, `FSA.concatenation` and `FSA.closure` and call it `chest`.
- (b) If you use the `view()` method to examine your FSA, you will find that it looks quite different from the one printed above. Use the `FSA.minimize` function to remedy this and call the result `min_chest`. Use `view()` to check your result.

As you know, the operators on FSAs introduce many states that increase the size of the automaton. Most of these states can be eliminated to create an equivalent FSA. As you do this practical you will notice that as you construct more complex automata, they begin to perform more slowly. The `minimize` function eliminates the redundant states and after applying it to your automaton you will find it will operate much more quickly.

- (c) * Check if your answer is correct using the `checkQ1()` method.

2. Next, we will extend the room with a large collection of cans of spinach hidden inside the chest. A player needs the spinach to become strong enough to get to the next level. The player can eat the spinach when the chest is open. It should be impossible to eat the spinach when the chest is closed.

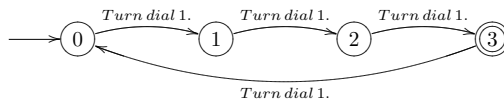
Here is the FSA, which we will refer to as `spinach`, which models the behaviour of the spinach:



- (a) Create `spinach` using only `FSA.singleton`, `FSA.concatenation`, `FSA.closure`, and possibly `FSA.minimize`.

- (b) * Check if your version of `spinach` is correct using the `checkQ2a()` method.
 - (c) Using both `chest` and `spinach`, create an FSA in Python with the behaviour specified above using `FSA.singleton`, `FSA.concatenation`, `FSA.closure`, `FSA.preInterleave`, and possibly `FSA.minimize`. Call it `spinach_in_chest`. *Remember*, you can not take the `spinach` out of the chest if it is closed!
 - (d) * Check if your version of `spinach_in_chest` is correct using the `checkQ2c()` method.
3. The next thing we will add to our room is a combination lock. It consists of three dials, each of which can be turned (clockwise) to the values 0, 1, 2 and 3. The secret combination is 3-0-1 (i.e. dial 1 is turned to 3, dial 2 is turned to 0, and dial 3 is turned to 1). This combination lock will be on the door which gets you to the next level. The initial configuration of all three dials is 0.

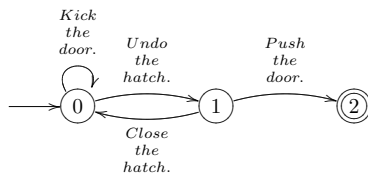
Here is the FSA, which we will refer to as `dial1`, which models the behaviour of the first of the three dials:



Notice that state 3 is the final state, reflecting that this is the correct position for the first dial.

- (a) Create an FSA in Python for `dial1` using only `FSA.singleton`, `FSA.concatenation`, `FSA.closure`, and possibly `FSA.minimize`.
 - (b) Do the same for the other two dials. Call them `dial2` and `dial3`.
 - (c) Use `dial1`, `dial2` and `dial3` and the appropriate function from the FSA module to create the combination lock which should be called `cLock`. *Remember that you can use the `minimize` function when your FSA is starting to get slow!*
 - (d) * Check if your version of `cLock` is correct using the `checkQ3c()` method.
 - (e) Create an FSA for a room which has in it: (1) a chest full of cans of spinach, and (2) a 3-dial combination lock. Call this room `level1`. *Note that `spinach_in_chest` and `cLock` are independent objects!*
 - (f) * Check if your version of `level1` is correct using the `checkQ3e()` method.
4. Our final addition to the game will be another room, or in more general terms: another game level. This second room will be a simple one, with only a door to the next level in it. The player can reach the room with the action `nextLevel1`, if (s)he has (a) eaten some spinach, and (b) turned the dials to the right combination. The Python FSA module contains a function called `composeLevels` which performs level composition on an arbitrary number of levels with an intermediate `nextlevel` action (see also the API).

Here is the FSA, which we will refer to as `level2`, which models the behaviour of the second level:



- (a) Create an FSA in Python for `level2` using only `FSA.singleton`, `FSA.concatenation`, `FSA.closure`, `FSA.union`, and possibly `FSA.minimize`.
- (b) * Check if your version of `level2` is correct using the `checkQ4a()` method.

- (c) Create an FSA in Python for the composition of `level1` and `level2` using `composeLevels` and call it `game1`.
- (d) Create a second FSA with the specified behaviour using any of the FSA combinators, except `composeLevels`. Call it `game2`.
- (e) * Check if the two game FSAs are equivalent using the `equivalent` function.
- (f) * Check if your versions of `game1` and `game2` are correct using the `checkQ4cd()` method.

In the next part of the assignment, you will use the FSAs you have created as the basis for a playable game. You could try play the game you just wrote as it is, using to following command: `FSA.play(game1, 'Introduction', 'Congratulations!', '>>')` (type `Goodbye!` to exit). However, since no descriptive information has been added yet, the game can be rather confusing. For example, the player has no idea that the spinach is inside the chest, so imagine his surprise when it suddenly disappears when he closes the chest!

In question 1a you should have used two different calls to `singleton`: `singleton('Open the chest.')` and `singleton('Close the chest.')`. A call to `singleton` *always* results in an FSA with two states. We will now supply some information for each of these states in both of the calls: `singleton('Open the chest.', 'You see a chest. It is closed.', 'You see a chest. It is open. There are several cans of spinach in it.')` and `singleton('Close the chest.', 'You see an open chest with spinach in it.', 'You see a closed chest.')`

Notice that a list of states and their description appears every time you call the method `view()`.

5. (a) Create a second FSA in Python modelling `chest`, but this time use the new calls to `singleton`. Call the new FSA `i_chest_d`.
- (b) After minimization, you can look at your FSA using the method `view()`. You will notice that there seems to be some redundant information in the state descriptions. Why is that? Write your answer as a comment at the top of your `Intro.py`.
- (c) Using your answer to the previous question, create a third FSA in Python modelling `chest`. Call the new FSA `i_chest`. It should give the same descriptive information as `i_chest_d`, but without redundancy.

1.2 The Game - Version B (30 marks)

Now that you have experimented with building an adventure game using the small FSA building blocks, you are ready to start programming your own game. Some examples of games could be: 'NASA Space Expedition', 'Dating Game', 'Knight vs. Dragon', 'Surviving Pollock Halls', etc. Basically, anything goes. Surprise us!

The only restrictions are that you do not reuse code from the introduction and that your game satisfies the following structural and behavioural requirements:

- (Behaviour 1)** It takes the player at least 15 actions to reach the goal.
- (Behaviour 2)** It is always possible to reach the goal.
- (Behaviour 3)** Your game contains infinite paths.
- (Behaviour 4)** Your game contains at least 1 'long' cycle (i.e. at least 5 states before a repeat).
- (Behaviour 5)** Within at least 1 level it is always possible to reach the initial state (of the level) with 1 step. Note that it is not required that the initial state can be reached from the *final* states of the level.

(**Structure 1**) Your game consists of at least 3 levels.

(**Structure 2**) your game has at least 1 union of size ≥ 5 .

(**Structure 3**) Your game has at least 1 interleave of size ≥ 5 .

(**Structure 4**) Your game has at least 1 precondition interleave.

(**Structure 5**) Your game has at least 1 nested precondition interleave (i.e. of the form $(FSA1 \mid > (FSA2 \mid > FSA 3))$). NB. These *cannot* be the same ones used for structural constraint 4!

Furthermore, marks will be deducted if your game does not satisfy the following requirements:

(**Negative 1**) Your code should be well documented.

(**Negative 2**) You should provide at least one information string – for initial state and/or final state – with each call to `singleton`.

(**Negative 3**) Your code should compile in less than 1 minute on a standard DICE machine.

(**Negative 4**) Your variables should have informative names.

(**Negative 5**) Your levels are composed using `composeLevels`.

Please remember that you should not spend more than 25 hours on this assignment!

6. Implement an adventure game in Python using the combinators in the FSA module, store it in a variable and call it `gameFSA`. Your game should satisfy all the given behavioural, structural, and negative requirements. Call the file that contains your game `Game.py`.
7. Write a short introduction what your game is about, how it is played and what the goal is. Insert this piece of prose into a variable and call it `intro`. You should also write a short message of congratulations to any player who finishes the game successfully. Store it into a variable and call it `congrats`.
8. * Check that your game is playable with the following statement:
`FSA.play((gameFSA, intro, congrats, "Yes? "))`
9. * Check that your game satisfies all of the behavioural constraints using the `behaviour1()` – `behaviour5()` methods. Note that you will have to invoke some of the methods on the actual game FSA, and others on the individual level FSAs.

2 Submitting your Assignment

When you have finished sections 1 and 2, you are ready to submit this part of your assignment. Please note that your code should compile in < 1 minute on a standard DICE machine, or your assignment will not be marked. Most games will satisfy this requirement *if* the `minimize` function is used appropriately. Please also remember to always document your code appropriately and check to see that you have two files called `Intro.py` and `Game.py`. Only files with those *exact* names will be accepted by the `submit` program. Put these three files into a single directory and submit them using the following command from a terminal window:

```
submit inf2 inf2a A1 Intro.py Game.py
```

Remember that the questions on this handout are only worth 70/100 marks and that you still need to answer some theory questions on the Assignment 1 website for the remaining 30 marks!

THE END

Plagiarism

There are some general principles in preparing any work:

- You should complete coursework yourself, using your own words, code, figures, etc.
- Acknowledge your sources for text, code, figures etc that are not your own.
- Take reasonable precautions to ensure that others do not copy your work and present it as their own.

If you produce your own unassisted work then you are in no danger of plagiarising. However, if your work requires some help or collaboration then you should follow the guidelines below.

It is acceptable to use general ideas picked up, for example, in discussion with others, or in textbooks, without acknowledgement. In general, if you write the work you submit, in its entirety, yourself then you do not need to include an acknowledgement. An exception is when a really pivotal idea comes from an external source, in which case you should acknowledge that source.

There are occasions on which you will wish to include work written by somebody else within your own work. For example, you may be quoting a passage from a book within an essay, or you may be unable to finish a program yourself and require a piece of code written by someone else. On such occasions, you should copy the original work verbatim together with a full and explicit acknowledgement of the original source. The beginning and end of the copied material must be clearly indicated:

- In the case of copied text (in an essay or written answer to a question), you should enclose the text within quotation marks and explicitly indicate the source of the work close to where the copied material appears.
- In the case of code, you should insert a comment right before the start of the code you have copied or collaborated on, indicating who you have gotten it from or who you have collaborated with in developing it, and then another comment right after the end of the relevant code.

These rules about acknowledging the source of work that is not fully your own, apply irrespective of the type of work it is (prose, code, etc.).

If you collaborate on the preparation of all or part of a piece of work then the collaboration must be explicitly acknowledged, and the extent of the collaboration should be stated as fully as possible by all parties involved.

It is always unacceptable to attempt to disguise someone else's work in order to present it as your own, for example by making cosmetic changes to text, or by changing variable names or altering comments in code.

When in doubt, state explicitly what you have done and the sources of any help (including fellow students, tutors or textbooks) you receive. This will enable your work to be marked appropriately, and will ensure that you avoid suspicion of plagiarism.

In order to prevent others from copying your work and presenting it as their own, set the protection on your practical work directories appropriately. It is also sensible to ensure that any practical submissions that you print are collected promptly from the printers. By all means discuss your general ideas with other students, but do not distribute or lend your solutions to them.

This text was adapted from "Guidelines on Plagiarism" at <http://www.inf.ed.ac.uk/admin/ITO/DivisionalGuidelinesPlagiarism.html>.

Appendix F

API for FSA.py

Module FSA

Methods

Acceptance

accepts(sequence)	returns true or false
--------------------------	-----------------------

Accessors and predicates

isEmpty()	returns true iff the language accepted by the FSA is the empty language
labels()	returns a list of labels that are used in any transition

Reductions

minimized()	returns an equivalent minimal FSA
--------------------	-----------------------------------

Presentation

toDotString()	returns a string suitable as *.dot file for the 'dot' program from AT&T GraphViz
view()	views fsa with a gs viewer, if gs and dot are installed
tuple()	returns the 5-tuple representation of an fsa
displayStates()	pretty-prints the state information of an fsa
displayTransitions()	pretty-prints the transitions of an fsa

Behaviour Checks

behaviour1()	Checks a (game) FSA for behavioural constraint 1.
behaviour2()	Checks a (game) FSA for behavioural constraint 2.
behaviour3()	Checks a (game) FSA for behavioural constraint 3.
behaviour4()	Checks a (game) FSA for behavioural constraint 4.

behaviour5()	Checks a (level) FSA for behavioural constraint 5.
---------------------	--

Assignment Question Checks

checkQ1()	Checks your answer to Question 1.
checkQ2()	Checks your answer to Question 2.
checkQ3()	Checks your answer to Question 3.
checkQ4()	Checks your answer to Question 4.
checkQ5a()	Checks your answer to Question 5a.
checkQ5d()	Checks your answer to Question 5d.
checkQ5f()	Checks your answer to Question 5f.
checkQ6()	Checks your answer to Question 6.

Functions

Construction from FSAs

closure(a)	returns an fsa that accepts sequences composed of zero or more concatenations of sequences accepted by the argument
concatenation(fsas)	returns an fsa that accepts sequences composed of a sequence accepted by the first fsa in fsas, followed by a sequence accepted by the second fsa in fsas etc. Note that the FSAs in fsas should be seperated by commas.
intersection(a, b)	returns an fsa that accepts sequences accepted by both a and b
option(a)	equivalent to union(a, EMPTY_STRING_FSA)
union(fsas)	returns an fsa that accepts sequences accepted by one of the FSAs in fsas. Note that the FSAs in fsas should be seperated by commas.
interleave(fsas)	returns an fsa that accepts a sequence accepted by a, interleaved with a sequence accepted by b
preInterleave(a,b)	returns the <u>pre-condition interleave</u> of a and b.
composeLevels(levels)	returns the <u>level composition</u> of levels (which should be seperated by commas)

A *pre-condition interleave* of fsa a and fsa b accepts a subsets of the normal interleaves of a and b. The additional

restriction is that when interleaving a sequence of accepted by a with a sequence accepted by b, you can only insert an element of b if the already inserted elements of a, form a sequence accepted by a. In other words, the fsa a has to be in one of its final states in order for you to insert an element of b.

Example. a = "x" "y" "z" ("y" "z")*, b = "w"*

The pre-condition interleave of a and b accepts:

- "x" "y" "z" "w" "w"
- "x" "y" "z" "y" "z" "w" "w"
- "x" "y" "z" "w" "w" "w" "y" "z"
- "x" "y" "z" "w" "w" "y" "z" "w" "w" "w"

But does not accept:

- "x" "y" "w" "z"
- "x" "y" "z" "y" "w" "w" "z"

A **level composition** of fsa a and fsa b accepts sequences consisting of a sequence accepted by a, followed by an occurrence of the "nextLevel" label, followed by a sequence accepted by b.

Example. a = "x"*, b = "y"*

The level composition of a and b accepts:

- "nextLevel"
- "nextLevel" "y" "y"
- "x" "nextLevel"
- "x" "x" "nextLevel" "y"

But does not accept:

- "y" "nextLevel" "x"
- "x" "y" "nextLevel"
- "nextLevel" "x" "y"

Predicates

equivalent(a, b)	returns true iff a and b accept the same language
-------------------------	---

Reductions (these equivalent to the similarly-named methods)

minimize(fsa)	returns an equivalent minimal FSA
----------------------	-----------------------------------

Construction from labels

singleton(label)

returns an fsa that accepts singletons whose elements are matched by label. For example, singleton('a') returns an fsa that accepts only the string 'a'. *NB. label should not contain any instances of '~' or '&'*

FSA Constants**EMPTY_STRING_FSA**

is an FSA that accepts the language consisting only of the empty string.

Adapted from the original [FSA.py](#) by Oliver Steele. Last modified on 10/26/2007 10:42:16 by Laura.Korte@ed.ac.uk

Appendix G

Finite State Automata Assignment (Marking Guide)

Dear Marker,

The assignment consists of 3 parts. Therefore there are 3 marking scripts:

Mark-Intro.py
Mark-Game.py
Mark-Theory.py

Besides these 3 scripts, the directory attached also contains the module FSA.pyc, the 'data' directory, the two versions of the assignment, the example game (WeirdestLinkBin.pyc), and example solutions of Intro.py and Game.py. The students were assigned one of the two versions of the assignment. This should not make any difference for the marking process, but in case you are interested, you find the version the student was assigned to on the first line of their .txt file in the data-directory. For example, the following means that the student was assigned to v2, which is version B:

```
GsyWZmxT    v2  
:  
:
```

To mark the submission of student s0233694 in tutorial group T03, copy the following files to the marking directory:

```
/home/submit/submissions/inf2/inf2a/T03/s0233694/A1/Intro.py  
/home/submit/submissions/inf2/inf2a/T03/s0233694/A1/Game.py
```

If you find that you do not have access to the submit directory, please submit a ticket to support telling them that you are an inf2a marker and need permission to the submit directory.

The first thing you should do is make sure the submission was made in time. Check the timestamp of both Intro.py and Game.py using the following statement:


```
ls -l Game.py Intro.py
```

Anything saved after Oct 16 16:00 is late and should be marked as such. If you find a late submission, please add a note of this at the end of your marking report (see below).

Now create a file called 's0233694.marks'. Run the 3 marking scripts and store their results in 's0233694.marks'. If the student has followed the instructions in the assignment this process should be smooth. If not, you may be prompted for some minor corrections or variable names. Please have the assignment handy, so you are able to answer these questions. This is how you run each of the scripts:

```
python Mark-Intro.py
python Mark-Game.py
python Mark-Theory.py <s0XXXXXX>
```

The first two scripts assume you have the student's Intro.py and Game.py in the current directory. The third one requires the data-subdirectory.

After you have run the scripts and saved the answers, please mark S1-S5 and N1-N5 manually and insert the marks where Mark-Game.py has created a space for them. Finally, add up all the marks (imagine that the total is 77) and insert a single line at the bottom of s0233694.marks:

```
=====> TOTAL MARKS: 77/100
```

If this particular student was late in submitting their file, please write the following instead:

```
=====> TOTAL MARKS: 77/100
=====> LATE: Oct 16 17:51
```

Assuming the timestamp on the student's files was Oct 16 17:51

Please keep track of frequent mistakes as we would like to compile a list of 'common mistakes' for the students to look over. Any questions please let me know. I'll be away to Furbush Wed-Fri this week, but I will answer any questions you may have as soon as I get back.

We have been given exactly 2 weeks to do the marking, so please return all your <studentnumber>.marks to be by October 30th. This particular assignment takes very little time per student, so I have assigned each of you half of the marking, that should take you about as much time as it took me to write the marking scripts :)

Based on the number of submissions per tutorial group, I suggest the following division:

Ingo	- t01, t02, t08, t09, t10, t11	= 51 submissions
Songfang	- t03, t04, t05, t06, t07	= 51 submissions

Thanks,
Laura (orange.laura@gmail.com).

```
=====  
=== Game Preliminaries (40 marks) ===  
=====
```

- 1a. WORTH: 4
NOTES: Auto Check.
- 1b. WORTH: 1
NOTES: Auto Check.
- 1c. WORTH: 0
NOTES: Starred Question (no marks).
- 2a. WORTH: 4
NOTES: Auto Check.
- 2b. WORTH: 0
NOTES: Starred Question (no marks).
- 2c. WORTH: 3
NOTES: Auto Check.
- 2d. WORTH: 0
NOTES: Starred Question (no marks).
- 3a. WORTH: 4
NOTES: Auto Check.
- 3b. WORTH: 2
NOTES: Auto Check.
Correct answers for dial2 and dial3 are each worth 1 mark.
- 3c. WORTH: 3
NOTES: Auto Check.
- 3d. WORTH: 0
NOTES: Starred Question (no marks).
- 3e. WORTH: 3
NOTES: Auto Check.
- 3f. WORTH: 0
NOTES: Starred Question (no marks).
- 4a. WORTH: 4
NOTES: Auto Check.
- 4b. WORTH: 0
NOTES: Starred Question (no marks).
- 4c. WORTH: 3
NOTES: Auto Check (interactive).
The marking program will prompt you for an answer to the question:
"Is *game1* composed using *composeLevels*?"
You can find the answer to this question in the file Into.py.
- 4d. WORTH: 3
NOTES: Auto Check (interactive).
The marking program will prompt you for an answer to the question:
"Is *game2* composed without the use of *composeLevels*?"
You can find the answer to this question in the file Into.py.
- 4e. WORTH: 0
NOTES: Starred Question (no marks).
- 4f. WORTH: 0
NOTES: Starred Question (no marks).
- 5a. WORTH: 2

NOTES: Auto Check.

5b. WORTH: 2

NOTES: Please Mark.

The marking program will prompt you for an answer to the question:

"Is the answer to question 5b correct?"

You should find the answer to question 5b as a comment at the top of the file Intro.py. A correct answer says something along these lines: The redundancy in the state information is created by the *minimize* function. It collapses states together and so the information of two separate states gets merged together.

5c. WORTH: 2

NOTES: Auto Check (interactive).

After printing the state-information, the marking program will prompt you for an answer to the question:

"Is there still redundancy in these states?"

* If at least one of the states contains statements that basically mean the same (like "You see a chest. It is closed" and "You see a closed chest"), answer 'y'.

* If none of the states contain redundant info, answer 'n'. Note that we are only looking for redundancy WITHIN states not BETWEEN states.

```
=====
=== The Game (30 marks) ===
=====
```

B1. WORTH: 3 marks

NOTES: Auto Check.

B2. WORTH: 3 marks

NOTES: Auto Check.

B3. WORTH: 3 marks

NOTES: Auto Check.

B4. WORTH: 3 marks

NOTES: Auto Check.

B5. WORTH: 3 marks

NOTES: Auto Check (interactive).

The marking program will prompt you for an answer to the question:

"Please give the names of all the level variables: "

You should enter the names of the arguments of the *composeLevels* statement as you find them in the file Game.py, separated by commas.

For example: 'lev1,lev2,lev3'

S1. WORTH: 3 marks

NOTES: Please check that the game consists of 3 levels or more by examining the file Game.py.

* If the game consists of only 1 level: 1 mark

* If the game consists of only 2 levels: 2 marks

* If the game consists of 3 levels or more: 3 marks

S2. WORTH: 3 marks

- NOTES: Please check that the game includes a union of size 5 or larger by examining the file `Game.py`.
- * If there is a union of size 2: 1 mark
 - * If there is a union of size 3 or 4: 2 marks
 - * If there is a union of size 5 or larger: 3 marks
- S3. WORTH: 3 marks
- NOTES: Please check that the game includes an interleave of size 5 or larger by examining the file `Game.py`.
- * If there is an interleave of size 2: 1 mark
 - * If there is an interleave of size 3 or 4: 2 marks
 - * If there is an interleave of size 5 or larger: 3 marks
- S4. WORTH: 3 marks
- NOTES: Please check that the game includes a precondition interleave by examining the file `Game.py`.
- * If there is a pre-condition interleave: 3 marks
- S5. WORTH: 3 marks
- NOTES: Please check that the game includes a nested precondition interleave by examining the file `Game.py`. Note that this cannot include the pre-condition interleave used for S4!
- * If there is a nested pre-condition interleave but it uses the pre-condition of S4: 1 mark
 - * If there is a nested pre-condition interleave and it does not use the pre-condition of S4: 3 marks
- N1. WORTH: -3 marks
- NOTES: Please look at the code in `Game.py` and determine if the code is well-documented.
- * If there is no documentation at all: -3 marks
 - * If there is some documentation, but not satisfactory: -1 or -2 marks depending on the usefulness of the code. Judge for yourself.
- N2. WORTH: -3 marks
- NOTES: Please find all `*singleton*` calls in the file `Game.py`.
- * If none of them have any info: -3 marks
 - * If many are missing: -2 marks
 - * If only a few are missing: -1 mark
- N3. WORTH: -3 marks
- NOTES: * If code compiles in less than 5 minutes (on DICE): no penalty.
- * If code compiles in more than 5 minutes, but less than 10 (on DICE): -3 marks
 - * If code does not compile in under 10 minutes: award 0/30 marks for the entire game section.
- N4. WORTH: -3 marks
- NOTES: Please have a quick look at the variable names the students use.
- Note 1: Names like `'x'`, `'i'`, or `'v4'` are only acceptable as counters in a loop.
- Note 2: Variables can be too long, as well as too short.
- Note 3: The variable name should somehow tell you what it contains.
- * If none of these conventions are followed: -3 marks
 - * If only one of these conventions is used: -2 marks
 - * If only two of these conventions are used: -1 mark
- N5. WORTH: -3 marks
- NOTES: Please have a look at the variable `*gameFSA*`.
- Is it composed using `*compseLevels*`?
- * If no: -3 marks

=====
=== Theory (30 marks) ===
=====

Q16. WORTH: 1 mark

NOTES: Auto Check (interactive).

The marking program will print the SOLUTION and the STUDENT ANSWER.

If you think the student answer is correct with respect to the SOLUTION,
please enter 'y', if not, please answer 'n'.

Q17. WORTH: 1 mark

NOTES: Auto Check (interactive).

The marking program will print the SOLUTION and the STUDENT ANSWER.

If you think the student answer is correct with respect to the SOLUTION,
please enter 'y', if not, please answer 'n'.

Q18. WORTH: 1 mark

NOTES: Auto Check.

Q19. WORTH: 1 mark

NOTES: Auto Check.

Q20. WORTH: 1 mark

NOTES: Auto Check.

Q21. WORTH: 1 mark

Q22. WORTH: 3 marks

NOTES: Auto Check.

Q23. WORTH: 1 mark

NOTES: Auto Check.

Q24. WORTH: 1 mark

NOTES: Auto Check.

Q25. WORTH: 2 marks

NOTES: Auto Check.

Q26. WORTH: 2 marks

NOTES: Auto Check.

Q27. WORTH: 2 marks

NOTES: Auto Check.

Q28. WORTH: 3 marks

NOTES: Auto Check.

Q29. WORTH: 3 marks

NOTES: Auto Check.

Q30. WORTH: 3 marks

NOTES: Auto Check.

The marking program will print the SOLUTION and the STUDENT ANSWER.

If you think the student answer is correct with respect to the SOLUTION,
please enter 'y', if not, please answer 'n'.

Q31. WORTH: 3 marks

NOTES: Auto Check.

The marking program will print the SOLUTION and the STUDENT ANSWER.

If you think the student answer is correct with respect to the SOLUTION,
please enter 'y', if not, please answer 'n'.

Q32. WORTH: 1 mark

NOTES: Auto Check.

Appendix H

Turing Machine Assignments

Inf2A: Assignment 3

Adventure Games through TMs

– Version A –

ANDERSON, KORTE

Week 7 (30 October 2006)

If you have any questions regarding this assignment, please post them to the newsgroup `eduni.inf.course.inf2a`, or ask a demonstrator during one of the lab sessions. The deadline for this assignment is **Friday 10 November, 4pm**. You should spend a maximum of 25 hours on this assignment; remember that you have work for other courses as well. Happy programming!

In this assignment you will be writing an adventure game using Turing Machines (TMs). Section 1 of the assignment consists of two parts. Part 1 introduces you to the Python module `tm` and the task of programming adventure games. In Part 2 you will be programming an adventure game. Section 2 of the assignment contains some theory questions and has to be completed online, via the Inf2a Assignment 3 website.

You should put your answers in three files (`Main.py`, `Intro.tm` and `Game.tm`). Only files with those exact names will be accepted by the `submit` program. You should put all your answers to the questions in Section 1.1. into `Intro.tm` and `Main.py`. Similarly, you should put your game, according to the specification in Section 1.2. into `Game.py`. Put all three files into a single directory and from that directory, submit them using the following command:

```
submit inf2 inf2a A3 Main.py Intro.tm Game.tm
```

Laura Korte, whom many of you will know from Inf1B lab-sessions and Inf1B and Inf2A tutorials, is doing research on the topic of knowledge modeling through game building. She would like to use your submissions to this assignment for research purposes. Please note that the identity of participants will remain anonymous. Furthermore, any document containing results of this research project will be presented in such a way that no individual can be identified. If you wish for your assignment to be excluded from the research, please mention this as a comment at the top of your `Main.py` file. Also, if you have any further questions, or would like to receive a copy of the results of this research, please email Laura at `orange.laura@gmail.com`.

1 Programming

For this assignment, you will be using the Python module `tm`. You can find it and its API in the main menu of the Assignment 3 website, which is linked from the Inf2A website. Download the module `tm.pyc` and the files `Main.py` and `Intro.tm` from the website.

In file `Intro.tm` you will find the description of the Turing machine and in file `Main.py` you will find the game wrapper. Let's have a look at `Intro.tm` first. The uncommented part looks like this:

```

## initial      final  blank  variable  input  prompt
0              100    .      $         ?      Action?

## read instruction from tape5
0  ('%', '@', '$', '$', 'p') 2  ('%', '@', '$', '$', 'p') ('T', 'T', 'R', 'T', 'T') pick some berries
0  ('%', '@', '$', '$', 'I') 5  ('%', '@', '$', '$', 'I') ('T', 'T', 'R', 'T', 'T') print info
0  ('%', '@', '$', '$', '$') 7  ('%', '@', '$', '$', 'x') ('T', 'T', 'R', 'T', 'T') invalid action
0  ('%', '@', '$', '$', '.') 666 ('%', '@', '$', '$', '.') ('R', 'R', 'R', 'T', 'T') reached end of input without succeeding

## invalid action
7  ('%', '@', '$', '$', 'x') 8  ('%', '@', 'X', '$', 'x') ('T', 'T', 'T', 'T', 'T') print 'X' (invalid action)
8  ('%', '@', 'X', '$', 'x') 0  ('%', '@', '.', '$', '=') ('T', 'T', 'L', 'T', 'R') remove 'X' and goto next instr.

## pick some berries
2  ('%', 'B', '$', '$', '$') 3  ('%', 'B', '$', '$', '$') ('T', 'R', 'T', 'T', 'T') go left until you find 'B'
2  ('%', '$', '$', '$', '$') 2  ('%', '$', '$', '$', '$') ('T', 'L', 'T', 'T', 'T') :
3  ('%', '-', '$', '$', '$') 4  ('%', '+', '0', '$', '$') ('T', 'R', 'R', 'T', 'T') change '-' to '+', print '1'
3  ('%', '+', '$', '$', '$') 4  ('%', '+', '1', '$', '$') ('T', 'R', 'R', 'T', 'T') change '+' to '+', print '2'

## print info
5  ('%', '@', '.', '$', 'I') 6  ('%', '@', 'I', '$', 'I') ('T', 'T', 'T', 'T', 'T') print 'I' (info)
6  ('%', '@', 'I', '$', 'I') 0  ('%', '@', '.', '$', '=') ('T', 'T', 'L', 'T', 'R') remove 'I' and goto next instr.

## return head2 to its original position and go to the next instruction
4  ('%', '@', '$', '$', '$') 0  ('%', '@', '$', '$', '$') ('T', 'T', 'L', 'T', 'R') goto to next instr.
4  ('%', '$', '$', '$', '$') 4  ('%', '$', '$', '$', '$') ('T', 'R', 'T', 'T', 'T') find '@' on tape2

```

The `tm.py` package allows us to define k -tape Turing machines. The machine is equipped with k tapes and k tape heads each of which can move independently. The machines also have a simple mechanism for interacting with a user (formally this could just be seen as another tape but the mechanism chosen here is intuitive and makes defining interaction easier). The format for defining Turing machines in the package follows the following conventions:

- Any line beginning with `##` is read as a comment line and makes no contribution to the definition of the Turing machine. Of course comments may be very useful to improve the intelligibility of the TM definition.
- The first non-comment line in the TM definition defines most aspects of the machine except for the transitions. The alphabet of the machine is always implicitly the character set of the computer you are using. The first line comprises *six* elements in this order:
 1. The number of the initial state.
 2. The number of the final state.
 3. The symbol to be used to stand for the blank tape symbol (this is usually `'.'`).
 4. The so-called variable symbol (usually `'$'`). This is used to abbreviate transitions which do not change all of the tapes of the k -tape machine.
 5. The input symbol (usually `'?'`), if this symbol appears in a tape configuration the user is prompted to supply a symbol that will be substituted for `?`.
 6. The prompt, this will be displayed when the user is asked to supply input when the input symbol is encountered. Your prompt should *not* contain spaces.
- The remaining non-comment lines in the definition of a Turing machine comprise the transitions. Each line corresponds to a transition and each transition is a 5-tuple: $s\ t\ s'\ t'\ m$, where s is the current state, t is what is being read on the tapes, s' is the next state, t' is what is written on the tapes, and m is how the heads move on each tape. If we are describing a k -tape Turing machine, t and t' are both k -tuples of tape symbols and m is a k tuple of letters drawn from `'L'`, `'R'`, and `'T'` respectively standing for the head movements (*L*)eft, (*R*)ight, and *s*(*T*)ay. Anything on the line after the five elements of a transition is treated as a comment.

Let's have a look at the other file `Main.py` now. As you can see, it imports the `tm` module. You will also notice two functions called `loadIntroTapes()` and `loadIntroTM()`. They are reproduced below:

```
def loadIntroTapes(inputT=None,randomT=None):
    import random
    tape4_size = 50
    tape5_size = 50
    tape1 = "%#0,0@#0,0:a"
    tape2 = "%A-B-C-D-E-&5@"
    tape3 = "a"
    if randomT:
        tape4 = randomT
    else:
        tape4 = ""
        for x in range(tape4_size):
            c = random.choice("0123456789")
            tape4 = tape4 + c
    if inputT:
        tape5 = inputT
    else:
        tape5 = tape5_size*'?
    return (tape1,tape2,tape3,tape4,tape5)

def loadIntroTM(inputT=None,randomT=None):
    t = tm.read_tm_program("Intro.tm")
    (tape1,tape2,tape3,tape4,tape5) = loadIntroTapes(inputT=inputT,randomT=randomT)
    t.load_tape(tape1)
    t.load_tape(tape2,pos=13)
    t.load_tape(tape3)
    t.load_tape(tape4)
    t.load_tape(tape5)
    return t

t = loadIntroTM()
```

As you can see, `loadIntroTapes()` creates 5 strings representing the 5 initial tapes of our TM. The first string is not important at the moment, but will later contain information about the game-world. The second tape is the working tape, which is where we keep track of the player's possessions, and which we will later also use for simple computations. The third tape is our output tape. If the game-loop (`playIntro(tx)`) notices that the head of this tape is currently over a non-empty location on the tape, it will print it to the player. The fourth tape is the random tape: not important right now, but we will later use this tape (containing random numbers) to spice up our game. The last tape is the input tape; it merely contains a number of '?'s which – as mentioned before – prompt the user for an input symbol. The last line in this fragment of `Main.py` creates a Turing machine using `loadIntroTM` and `loadIntroTapes`. It will have the following initial tapes:

```
Tape 1: [%]#0,0@#0,0:a
Tape 2: %A-B-C-D-E-&5[0]
Tape 3: [a]
Tape 4: [8]7273906678323790385800648997259412080268578315652
Tape 5: [?]????????????????????????????????????????????????????????????
```

Note that tape 4 will never be exactly the same, because it is composed of 50 *random* numbers. The format of the 5 tapes is also discussed in this week's tutorial. Moving on to `loadIntroTM()` now, we see that this simply reads the TM defined in `Intro.tm` before loading each of the 5 tapes generated by `loadIntroTapes()`. The two remaining functions in the file `Main.py` which we will discuss right now are `showStep(tx,n=i)` and `playIntro(tx)`:

```
def showStep(tx,n=1):
    tx.stepn(n)
    print "STATE = " + str(tx.state)
    print tx.show()
```

```

def playIntro(tx):
    print introIntro
    loop = True
    while loop:
        o = tx.tapes[outputTape].read()
        if o == 'I':
            s1 = tx.tapes[stateTape].tape2string()
            fields = s1.split('@')
            fields = fields[0].split('#')
            displ = "Location          : " + fields[len(fields)-1]
            s2 = tx.tapes[workTape].tape2string()
            displ = displ + "\nResource A (bear hide): " + s2[2]
            displ = displ + "\nResource B (berries) : " + s2[4]
            displ = displ + "\nResource C (rock)   : " + s2[6]
            displ = displ + "\nResource D (fish)    : " + s2[8]
            displ = displ + "\nResource E (water)  : " + s2[10]
            displ = displ + "\nHealth          : " + s2[12] + "/5"
            print displ
        elif o <> tx.blank:
            print introDicti[o]
        try: tx.step()
        except tm.success: loop = False
        except tm.fail: loop = None
    if loop == False:
        print introCongrats

```

The first of these will come in handy when you are debugging your TM. The function takes a TM (*tx*) and an optional natural number (*i*). It advances the TM *i* steps and then prints the state followed by all (in our case 5) tapes. The second function allows you to play your game as defined by the TM *tx*. As you can see, it starts by printing out the introduction after which it goes into a loop which it does not exit until the final state of the TM has been reached. Basically, this function makes the TM advance step after step, only to be interrupted by either input (it will ask the player for a symbol) or output (it will print this to the player). Note that there is a special case for when the head of the output tape is currently over an 'I'. Can you tell what it does?

Coming back to the actual TM, let's try to run it. Type the following command in a terminal window from the directory where we have stored `Main.py`, `tm.pyc`, and `Intro.tm`:

```
python -i Main.py
```

From the Python prompt, there are several commands you can try. Type in the following commands to find out what they do:

- `t`
- `t.state`
- `t.show()`
- `showStep(t,n=1)`
- `showStep(t,n=10)`
- `playIntro(t)`

And once you are within `playIntro(t)`:

- `w`
- `I`
- `p`
- `I`

- p

As you can see, `playIntro(t)` allows you to play the game defined in `Intro.tm`. Right now it is a very boring one where the only thing you can do is pick some berries, but that is about to change. Before you start the assignment though, try to figure out how exactly ‘\$’ and ‘?’ work. This week’s tutorial sheet may help you with the first one.

1.1 Game Preliminaries (40 marks)

- (12 marks) In `Intro.tm` picking berries is implemented using the command ‘p’ to pick the berries and `introDicti['0']` and `introDicti['1']` to send messages to the user. After the berries are picked, the value of the variable ‘B’ is updated from ‘-’ to ‘+’. Extend `Intro.tm` to update the other variables (A,C,D, and E) using `introDicti['2']` – `introDicti['9']` and the following commands: ‘b’ to fight the bear, ‘r’ to pick up the rocks, ‘f’ to catch some fish, and ‘a’ to fill you water flask.
- (10 marks) It is not credible that you should always defeat the bear. Change `Intro.tm` such that instead of just updating variable ‘A’, you read a digit off the random tape (= tape 4) based on which you will decide whether or not the player wins this fight: if the digit is 0,1,2,3, or 4, the player loses; if the digit is 5,6,7,8, or 9, the player wins. If the player wins the fight, you should proceed as normal: update the variable ‘A’ and print the appropriate message (`introDicti['2']` or `introDicti['3']`) to the player. If the player loses the fight, their health should deteriorate by one unit and an appropriate message (uncomment `introDicti['L']` and `introDicti['W']`) should be printed to the player. If the player only had 1 health left before (s)he started the fight and lost, (s)he should die and the TM fail. The current health of the player is stored in the digit immediately to the left of the ‘@’ sign on the second tape.
- (3 marks) Change `Intro.tm` to deal with a new command ‘h’ which represents picking and eating some healing herbs. Eating healing herbs should result in an additional health unit, but only if the player has less then 5 health units at the time of consumption. Uncomment `introDicti['H']` and `introDicti['F']` and use them as the messages to print to the player after (s)he has eaten the herbs.
- (10 marks) Uncomment the rest of `introDicti` in `Main.py` and the coordinate code¹ in `Intro.tm` and make sure it fits with your own code such that you can give the additional commands ‘n’, ‘e’, ‘s’, and ‘w’. Now change your rules in `Intro.tm` which recognise the commands ‘b’, ‘p’, ‘r’, ‘f’, ‘a’, and ‘h’ so that they are only available in certain locations. the dictionary `introDicti` in `Main.py` contains the exact details. For example, take the following entry from `introDicti`:

```
"m" : "Mountains: pick up some (r)ocks, or move on (n,e,s, or w)."
```

This tells us that ‘r’ – picking up a rock – should only be available in the mountains ‘m’.

- (3 marks) Have a look at `walkThroughIntro()` in `Intro.tm` and try it out if you like. It is very similar to `playIntro(tx)`, except that instead of the ‘?’s on tape 5 asking the player for input, it has the actions already specified in the first items of the tuples in the `info`-list. Similarly, the random tape is pre-defined, rather than created at compile-time to allow for identical walk-throughs. You can find the items that make up the random tape in the third position of the tuples in the `intro`-list. The `walkThroughIntro()` function also prints out additional information allowing you to comment on a run of your game. These comments are stored in the second position of the tuples in the `intro`-list. So for each input-symbol, you can provide a comment string explaining what is happening in your game.

Change `walkThroughIntro()` so that it proves that you have implemented question 4 correctly. To do this, you will have to (a) extend `introInfo` to prove all aspects of question 4, and (b) make sure the output of `walkThroughIntro()` looks something like this:

¹The coordinate code is discussed in the week 7 tutorial (T3).

```

You are in your home village.
>>> PLAYER INPUT: I
>>> COMMENTS   : Initial info.

Location          : 0,0
Resource A (bear hide): -
Resource B (berries) : -
Resource C (rock)   : -
Resource D (fish)   : -
Resource E (water)  : -
Health            : 5/5
You are in your home village.
>>> PLAYER INPUT: p
>>> COMMENTS   : Proof that 'p' does not work in the home village.

Invalid Action!
You are in your home village.
>>> PLAYER INPUT: n
>>> COMMENTS   : Going North.

Cave: there is a bear on the path. If you run (n,e,s, or w) you will not gain anything,
but you won't die either. If you fight the (b)ear, you may gain a valuable resource,
but you can also get hurt.
>>> PLAYER INPUT: I
>>> COMMENTS   : Proof that going North works.

Location          : 0,1
Resource A (bear hide): -
Resource B (berries) : -
Resource C (rock)   : -
Resource D (fish)   : -
Resource E (water)  : -
Health            : 5/5
Cave: there is a bear on the path. If you run (n,e,s, or w) you will not gain anything,
but you won't die either. If you fight the (b)ear, you may gain a valuable resource,
but you can also get hurt.
>>> PLAYER INPUT: b
>>> COMMENTS   : Proof that 'b' is available in a cave.

You beat the bear and have gained a new resource: bear hide!
>>> PLAYER INPUT: I
:
:
etc.

```

There is no need for your output to be identical to the one above, but it should follow the same general idea of proving that 'n', 'e', 's', and 'w' are available and that you can only use 'b' in a cave, and 'h' in the hills, etc. To prove that an action is only available in location *X*, it is enough to show that it works in *X* and that it does not work in *one* other kind of location.

6. (2 marks) You may have noticed that right now, a *spring* 's' never occurs in the game. This is because we do not want spring to be generated like all the other location. We want exactly one spring in our game. Put the spring at location (5,5).

Hint. You should not have to change `Intro.tm` for this question.

1.2 The Game - Version A (30 marks)

In this section you will extend the game we have just considered in order to ensure all the requirements below are maintained. The extension is specified after the table of requirements. You should read the specification carefully and ensure your extended game implements the specified extension while ensuring all the requirements are met.

Exactly 3 marks will be awarded for each of the following requirements that you satisfy:

- (Requirement 1) *Extending state/world*: at least one of your tapes should be (infinitely!) extendible through playing the game.
- (Requirement 2) *Updating current state/world*: you should have a notion of current state, which gets updated (not extended!) throughout the game.
- (Requirement 3) *Randomness*: your TM should make some of its decisions based on the contents of the random tape.
- (Requirement 4) *Output*: your game should produce output to the player (i.e. write to tape 3).
- (Requirement 5) *Input*: your game should allow the player to control the game using input.
- (Requirement 6) *Victory dependence*: your victory should be dependent on at least two different pieces of information. For example: (1) the location of the player and (2) their current inventory/health.
- (Requirement 7) *10 Steps*: it should take the player at least 10 steps to win the game.
- (Requirement 8) *Choice*: there are several different ways for the game to unfold (e.g. different player-input results in different game events).
- (Requirement 9) *Winning*: it is possible to win the game by fulfilling the victory condition (i.e. the Turing machine reached its final state).
- (Requirement 10) *Losing*: it is possible to lose the game (i.e. the Turing machine fails).

Furthermore, 3 marks will be deducted for each of the following requirements that you do not satisfy:

- (Negative 1) Your code should be documented per state.
- (Negative 2) Your code should be documented per line.
- (Negative 3) Your game should include a dictionary `gameDicti` which makes for a playable game.

To satisfy N1 and N2, your code should look something like the first block of code in `Intro.tm` (displayed below). As you can see, there is a description for the state '0', i.e. *read instruction from tape5* and a description for each of the individual lines:

```
## read instruction from tape5
0      ('%', '@', '$', '$', 'p') 2      ('%', '@', '$', '$', 'p') ('T', 'T', 'R', 'T', 'T') pick some berries
0      ('%', '@', '$', '$', 'I') 5      ('%', '@', '$', '$', 'I') ('T', 'T', 'R', 'T', 'T') print info
0      ('%', '@', '$', '$', '$') 7      ('%', '@', '$', '$', 'x') ('T', 'T', 'R', 'T', 'T') invalid action
0      ('%', '@', '$', '$', '.') 666    ('%', '@', '$', '$', '.') ('R', 'R', 'R', 'T', 'T') reached end of input without succeeding
```

You probably noticed that many of these requirements are already satisfied by the code you wrote in the introduction questions. Since you are allowed to use this code for your game, you are getting quite a few marks *almost* for free. Almost, because marks for requirements will only be awarded if you show in your demo(s) that you have satisfied them. There are also a few requirements which are not satisfied by `Intro.tm`. In this section you will extend the game in such a way that it does satisfy them. Here is a description of the extension:

Take the code of `Intro.tm` and paste all of it in a new file called `Game.tm`. The game as it is, does not have a victory condition. We will add one by having the TM succeed when the player returns to his home village *with all five resources*. The TM should *not* succeed before the player returns to his home village and it should *not* succeed if they player does not have all five resources.

Please remember that you should not spend more than 25 hours on this assignment!

7. (a) Uncomment the game functions/variables in `Main.py` and provide bodies for them.
- (b) Implement the extension described above in `Game.tm`.
- (c) Uncomment the walk-through function `walkThroughGame()` and the variable `gameInfo` in `Main.py`. Provide bodies for them such that running the function proves that you have satisfied all the requirements except 8.
- (d) Prove that you have satisfied requirement 8, by specifying an alternative sequence of instructions and comments in `gameInfoAlt` in `Main.py`.

2 Submitting your Assignment

When you have finished sections 1 and 2, you are ready to submit this part of your assignment. Please remember to check that you have three files called `Main.py`, `Intro.tm`, and `Game.tm`. Only files with those *exact* names will be accepted by the `submit` program. Put these three files into a single directory and submit them using the following command from a terminal window:

```
submit inf2 inf2a A3 Main.py Intro.tm Game.tm
```

Remember that the questions on this handout are only worth 70/100 marks and that you still need to answer some theory questions on the Assignment 3 website for the remaining 30 marks!

THE END

Plagiarism

There are some general principles in preparing any work:

- You should complete coursework yourself, using your own words, code, figures, etc.
- Acknowledge your sources for text, code, figures etc that are not your own.
- Take reasonable precautions to ensure that others do not copy your work and present it as their own.

If you produce your own unassisted work then you are in no danger of plagiarising. However, if your work requires some help or collaboration then you should follow the guidelines below.

It is acceptable to use general ideas picked up, for example, in discussion with others, or in textbooks, without acknowledgement. In general, if you write the work you submit, in its entirety, yourself then you do not need to include an acknowledgement. An exception is when a really pivotal idea comes from an external source, in which case you should acknowledge that source.

There are occasions on which you will wish to include work written by somebody else within your own work. For example, you may be quoting a passage from a book within an essay, or you may be unable to finish a program yourself and require a piece of code written by someone else. On such occasions, you should copy the original work verbatim together with a full and explicit acknowledgement of the original source. The beginning and end of the copied material must be clearly indicated:

- In the case of copied text (in an essay or written answer to a question), you should enclose the text within quotation marks and explicitly indicate the source of the work close to where the copied material appears.
- In the case of code, you should insert a comment right before the start of the code you have copied or collaborated on, indicating who you have gotten it from or who you have collaborated with in developing it, and then another comment right after the end of the relevant code.

These rules about acknowledging the source of work that is not fully your own, apply irrespective of the type of work it is (prose, code, etc.).

If you collaborate on the preparation of all or part of a piece of work then the collaboration must be explicitly acknowledged, and the extent of the collaboration should be stated as fully as possible by all parties involved.

It is always unacceptable to attempt to disguise someone else's work in order to present it as your own, for example by making cosmetic changes to text, or by changing variable names or altering comments in code.

When in doubt, state explicitly what you have done and the sources of any help (including fellow students, tutors or textbooks) you receive. This will enable your work to be marked appropriately, and will ensure that you avoid suspicion of plagiarism.

In order to prevent others from copying your work and presenting it as their own, set the protection on your practical work directories appropriately. It is also sensible to ensure that any practical submissions that you print are collected promptly from the printers. By all means discuss your general ideas with other students, but do not distribute or lend your solutions to them.

This text was adapted from "Guidelines on Plagiarism" at <http://www.inf.ed.ac.uk/admin/ITO/DivisionalGuidelinesPlagiarism.html>.

Inf2A: Assignment 3

Adventure Games through TMs

– Version B –

ANDERSON, KORTE

Week 7 (30 October 2006)

If you have any questions regarding this assignment, please post them to the newsgroup `eduni.inf.course.inf2a`, or ask a demonstrator during one of the lab sessions. The deadline for this assignment is **Friday 10 November, 4pm**. You should spend a maximum of 25 hours on this assignment; remember that you have work for other courses as well. Happy programming!

In this assignment you will be writing an adventure game using Turing Machines (TMs). Section 1 of the assignment consists of two parts. Part 1 introduces you to the Python module `tm` and the task of programming adventure games. In Part 2 you will be programming an adventure game. Section 2 of the assignment contains some theory questions and has to be completed online, via the Inf2a Assignment 3 website.

You should put your answers in three files (`Main.py`, `Intro.tm` and `Game.tm`). Only files with those exact names will be accepted by the `submit` program. You should put all your answers to the questions in Section 1.1. into `Intro.tm` and `Main.py`. Similarly, you should put your game, according to the specification in Section 1.2. into `Game.py`. Put all three files into a single directory and from that directory, submit them using the following command:

```
submit inf2 inf2a A3 Main.py Intro.tm Game.tm
```

Laura Korte, whom many of you will know from Inf1B lab-sessions and Inf1B and Inf2A tutorials, is doing research on the topic of knowledge modeling through game building. She would like to use your submissions to this assignment for research purposes. Please note that the identity of participants will remain anonymous. Furthermore, any document containing results of this research project will be presented in such a way that no individual can be identified. If you wish for your assignment to be excluded from the research, please mention this as a comment at the top of your `Main.py` file. Also, if you have any further questions, or would like to receive a copy of the results of this research, please email Laura at orange.laura@gmail.com.

1 Programming

For this assignment, you will be using the Python module `tm`. You can find it and its API in the main menu of the Assignment 3 website, which is linked from the Inf2A website. Download the module `tm.pyc` and the files `Main.py` and `Intro.tm` from the website.

In file `Intro.tm` you will find the description of the Turing machine and in file `Main.py` you will find the game wrapper. Let's have a look at `Intro.tm` first. The uncommented part looks like this:


```

## initial      final  blank  variable  input  prompt
0              100    .      $         ?      Action?

## read instruction from tape5
0  ('%', '@', '$', '$', 'p') 2  ('%', '@', '$', '$', 'p') ('T', 'T', 'R', 'T', 'T') pick some berries
0  ('%', '@', '$', '$', 'I') 5  ('%', '@', '$', '$', 'I') ('T', 'T', 'R', 'T', 'T') print info
0  ('%', '@', '$', '$', '$') 7  ('%', '@', '$', '$', 'x') ('T', 'T', 'R', 'T', 'T') invalid action
0  ('%', '@', '$', '$', '.') 666 ('%', '@', '$', '$', '.') ('R', 'R', 'R', 'T', 'T') reached end of input without succeeding

## invalid action
7  ('%', '@', '$', '$', 'x') 8  ('%', '@', 'X', '$', 'x') ('T', 'T', 'T', 'T', 'T') print 'X' (invalid action)
8  ('%', '@', 'X', '$', 'x') 0  ('%', '@', '.', '$', '=') ('T', 'T', 'L', 'T', 'R') remove 'X' and goto next instr.

## pick some berries
2  ('%', 'B', '$', '$', '$') 3  ('%', 'B', '$', '$', '$') ('T', 'R', 'T', 'T', 'T') go left until you find 'B'
2  ('%', '$', '$', '$', '$') 2  ('%', '$', '$', '$', '$') ('T', 'L', 'T', 'T', 'T') :
3  ('%', '-', '$', '$', '$') 4  ('%', '+', '0', '$', '$') ('T', 'R', 'R', 'T', 'T') change '-' to '+', print '1'
3  ('%', '+', '$', '$', '$') 4  ('%', '+', '1', '$', '$') ('T', 'R', 'R', 'T', 'T') change '+' to '+', print '2'

## print info
5  ('%', '@', '.', '$', 'I') 6  ('%', '@', 'I', '$', 'I') ('T', 'T', 'T', 'T', 'T') print 'I' (info)
6  ('%', '@', 'I', '$', 'I') 0  ('%', '@', '.', '$', '=') ('T', 'T', 'L', 'T', 'R') remove 'I' and goto next instr.

## return head2 to its original position and go to the next instruction
4  ('%', '@', '$', '$', '$') 0  ('%', '@', '$', '$', '$') ('T', 'T', 'L', 'T', 'R') goto to next instr.
4  ('%', '$', '$', '$', '$') 4  ('%', '$', '$', '$', '$') ('T', 'R', 'T', 'T', 'T') find '@' on tape2

```

The `tm.py` package allows us to define k -tape Turing machines. The machine is equipped with k tapes and k tape heads each of which can move independently. The machines also have a simple mechanism for interacting with a user (formally this could just be seen as another tape but the mechanism chosen here is intuitive and makes defining interaction easier). The format for defining Turing machines in the package follows the following conventions:

- Any line beginning with `##` is read as a comment line and makes no contribution to the definition of the Turing machine. Of course comments may be very useful to improve the intelligibility of the TM definition.
- The first non-comment line in the TM definition defines most aspects of the machine except for the transitions. The alphabet of the machine is always implicitly the character set of the computer you are using. The first line comprises *six* elements in this order:
 1. The number of the initial state.
 2. The number of the final state.
 3. The symbol to be used to stand for the blank tape symbol (this is usually `'.'`).
 4. The so-called variable symbol (usually `'$'`). This is used to abbreviate transitions which do not change all of the tapes of the k -tape machine.
 5. The input symbol (usually `'?'`), if this symbol appears in a tape configuration the user is prompted to supply a symbol that will be substituted for `?`.
 6. The prompt, this will be displayed when the user is asked to supply input when the input symbol is encountered. Your prompt should *not* contain spaces.
- The remaining non-comment lines in the definition of a Turing machine comprise the transitions. Each line corresponds to a transition and each transition is a 5-tuple: $s\ t\ s'\ t'\ m$, where s is the current state, t is what is being read on the tapes, s' is the next state, t' is what is written on the tapes, and m is how the heads move on each tape. If we are describing a k -tape Turing machine, t and t' are both k -tuples of tape symbols and m is a k tuple of letters drawn from `'L'`, `'R'`, and `'T'` respectively standing for the head movements (*L*)eft, (*R*)ight, and *s*(*T*)ay. Anything on the line after the five elements of a transition is treated as a comment.

Let's have a look at the other file `Main.py` now. As you can see, it imports the `tm` module. You will also notice two functions called `loadIntroTapes()` and `loadIntroTM()`. They are reproduced below:

```
def loadIntroTapes(inputT=None,randomT=None):
    import random
    tape4_size = 50
    tape5_size = 50
    tape1 = "%#0,0@#0,0:a"
    tape2 = "%A-B-C-D-E-&5@"
    tape3 = "a"
    if randomT:
        tape4 = randomT
    else:
        tape4 = ""
        for x in range(tape4_size):
            c = random.choice("0123456789")
            tape4 = tape4 + c
    if inputT:
        tape5 = inputT
    else:
        tape5 = tape5_size*'?
    return (tape1,tape2,tape3,tape4,tape5)

def loadIntroTM(inputT=None,randomT=None):
    t = tm.read_tm_program("Intro.tm")
    (tape1,tape2,tape3,tape4,tape5) = loadIntroTapes(inputT=inputT,randomT=randomT)
    t.load_tape(tape1)
    t.load_tape(tape2,pos=13)
    t.load_tape(tape3)
    t.load_tape(tape4)
    t.load_tape(tape5)
    return t

t = loadIntroTM()
```

As you can see, `loadIntroTapes()` creates 5 strings representing the 5 initial tapes of our TM. The first string is not important at the moment, but will later contain information about the game-world. The second tape is the working tape, which is where we keep track of the player's possessions, and which we will later also use for simple computations. The third tape is our output tape. If the game-loop (`playIntro(tx)`) notices that the head of this tape is currently over a non-empty location on the tape, it will print it to the player. The fourth tape is the random tape: not important right now, but we will later use this tape (containing random numbers) to spice up our game. The last tape is the input tape; it merely contains a number of '?'s which – as mentioned before – prompt the user for an input symbol. The last line in this fragment of `Main.py` creates a Turing machine using `loadIntroTM` and `loadIntroTapes`. It will have the following initial tapes:

```
Tape 1: [%]#0,0@#0,0:a
Tape 2: %A-B-C-D-E-&5[0]
Tape 3: [a]
Tape 4: [8]7273906678323790385800648997259412080268578315652
Tape 5: [?]????????????????????????????????????????????????????????????
```

Note that tape 4 will never be exactly the same, because it is composed of 50 *random* numbers. The format of the 5 tapes is also discussed in this week's tutorial. Moving on to `loadIntroTM()` now, we see that this simply reads the TM defined in `Intro.tm` before loading each of the 5 tapes generated by `loadIntroTapes()`. The two remaining functions in the file `Main.py` which we will discuss right now are `showStep(tx,n=i)` and `playIntro(tx)`:

```
def showStep(tx,n=1):
    tx.stepn(n)
    print "STATE = " + str(tx.state)
    print tx.show()
```

```

def playIntro(tx):
    print introIntro
    loop = True
    while loop:
        o = tx.tapes[outputTape].read()
        if o == 'I':
            s1 = tx.tapes[stateTape].tape2string()
            fields = s1.split('@')
            fields = fields[0].split('#')
            displ = "Location          : " + fields[len(fields)-1]
            s2 = tx.tapes[workTape].tape2string()
            displ = displ + "\nResource A (bear hide): " + s2[2]
            displ = displ + "\nResource B (berries) : " + s2[4]
            displ = displ + "\nResource C (rock)   : " + s2[6]
            displ = displ + "\nResource D (fish)    : " + s2[8]
            displ = displ + "\nResource E (water)  : " + s2[10]
            displ = displ + "\nHealth           : " + s2[12] + "/5"
            print displ
        elif o <> tx.blank:
            print introDicti[o]
        try: tx.step()
        except tm.success: loop = False
        except tm.fail: loop = None
    if loop == False:
        print introCongrats

```

The first of these will come in handy when you are debugging your TM. The function takes a TM (*tx*) and an optional natural number (*i*). It advances the TM *i* steps and then prints the state followed by all (in our case 5) tapes. The second function allows you to play your game as defined by the TM *tx*. As you can see, it starts by printing out the introduction after which it goes into a loop which it does not exit until the final state of the TM has been reached. Basically, this function makes the TM advance step after step, only to be interrupted by either input (it will ask the player for a symbol) or output (it will print this to the player). Note that there is a special case for when the head of the output tape is currently over an 'I'. Can you tell what it does?

Coming back to the actual TM, let's try to run it. Type the following command in a terminal window from the directory where we have stored `Main.py`, `tm.pyc`, and `Intro.tm`:

```
python -i Main.py
```

From the Python prompt, there are several commands you can try. Type in the following commands to find out what they do:

- `t`
- `t.state`
- `t.show()`
- `showStep(t,n=1)`
- `showStep(t,n=10)`
- `playIntro(t)`

And once you are within `playIntro(t)`:

- `w`
- `I`
- `p`
- `I`

- p

As you can see, `playIntro(t)` allows you to play the game defined in `Intro.tm`. Right now it is a very boring one where the only thing you can do is pick some berries, but that is about to change. Before you start the assignment though, try to figure out how exactly ‘\$’ and ‘?’ work. This week’s tutorial sheet may help you with the first one.

1.1 Game Preliminaries (40 marks)

- (12 marks) In `Intro.tm` picking berries is implemented using the command ‘p’ to pick the berries and `introDicti['0']` and `introDicti['1']` to send messages to the user. After the berries are picked, the value of the variable ‘B’ is updated from ‘-’ to ‘+’. Extend `Intro.tm` to update the other variables (A,C,D, and E) using `introDicti['2']` – `introDicti['9']` and the following commands: ‘b’ to fight the bear, ‘r’ to pick up the rocks, ‘f’ to catch some fish, and ‘a’ to fill you water flask.
- (10 marks) It is not credible that you should always defeat the bear. Change `Intro.tm` such that instead of just updating variable ‘A’, you read a digit off the random tape (= tape 4) based on which you will decide whether or not the player wins this fight: if the digit is 0,1,2,3, or 4, the player loses; if the digit is 5,6,7,8, or 9, the player wins. If the player wins the fight, you should proceed as normal: update the variable ‘A’ and print the appropriate message (`introDicti['2']` or `introDicti['3']`) to the player. If the player loses the fight, their health should deteriorate by one unit and an appropriate message (uncomment `introDicti['L']` and `introDicti['W']`) should be printed to the player. If the player only had 1 health left before (s)he started the fight and lost, (s)he should die and the TM fail. The current health of the player is stored in the digit immediately to the left of the ‘@’ sign on the second tape.
- (3 marks) Change `Intro.tm` to deal with a new command ‘h’ which represents picking and eating some healing herbs. Eating healing herbs should result in an additional health unit, but only if the player has less then 5 health units at the time of consumption. Uncomment `introDicti['H']` and `introDicti['F']` and use them as the messages to print to the player after (s)he has eaten the herbs.
- (10 marks) Uncomment the rest of `introDicti` in `Main.py` and the coordinate code¹ in `Intro.tm` and make sure it fits with your own code such that you can give the additional commands ‘n’, ‘e’, ‘s’, and ‘w’. Now change your rules in `Intro.tm` which recognise the commands ‘b’, ‘p’, ‘r’, ‘f’, ‘a’, and ‘h’ so that they are only available in certain locations. the dictionary `introDicti` in `Main.py` contains the exact details. For example, take the following entry from `introDicti`:

```
"m" : "Mountains: pick up some (r)ocks, or move on (n,e,s, or w)."
```

This tells us that ‘r’ – picking up a rock – should only be available in the mountains ‘m’.

- (3 marks) Have a look at `walkThroughIntro()` in `Intro.tm` and try it out if you like. It is very similar to `playIntro(tx)`, except that instead of the ‘?’s on tape 5 asking the player for input, it has the actions already specified in the first items of the tuples in the `info`-list. Similarly, the random tape is pre-defined, rather than created at compile-time to allow for identical walk-throughs. You can find the items that make up the random tape in the third position of the tuples in the `intro`-list. The `walkThroughIntro()` function also prints out additional information allowing you to comment on a run of your game. These comments are stored in the second position of the tuples in the `intro`-list. So for each input-symbol, you can provide a comment string explaining what is happening in your game.

Change `walkThroughIntro()` so that it proves that you have implemented question 4 correctly. To do this, you will have to (a) extend `introInfo` to prove all aspects of question 4, and (b) make sure the output of `walkThroughIntro()` looks something like this:

¹The coordinate code is discussed in the week 7 tutorial (T3).

```

You are in your home village.
>>> PLAYER INPUT: I
>>> COMMENTS : Initial info.

Location : 0,0
Resource A (bear hide): -
Resource B (berries) : -
Resource C (rock) : -
Resource D (fish) : -
Resource E (water) : -
Health : 5/5
You are in your home village.
>>> PLAYER INPUT: p
>>> COMMENTS : Proof that 'p' does not work in the home village.

Invalid Action!
You are in your home village.
>>> PLAYER INPUT: n
>>> COMMENTS : Going North.

Cave: there is a bear on the path. If you run (n,e,s, or w) you will not gain anything,
but you won't die either. If you fight the (b)ear, you may gain a valuable resource,
but you can also get hurt.
>>> PLAYER INPUT: I
>>> COMMENTS : Proof that going North works.

Location : 0,1
Resource A (bear hide): -
Resource B (berries) : -
Resource C (rock) : -
Resource D (fish) : -
Resource E (water) : -
Health : 5/5
Cave: there is a bear on the path. If you run (n,e,s, or w) you will not gain anything,
but you won't die either. If you fight the (b)ear, you may gain a valuable resource,
but you can also get hurt.
>>> PLAYER INPUT: b
>>> COMMENTS : Proof that 'b' is available in a cave.

You beat the bear and have gained a new resource: bear hide!
>>> PLAYER INPUT: I
:
:
etc.

```

There is no need for your output to be identical to the one above, but it should follow the same general idea of proving that 'n', 'e', 's', and 'w' are available and that you can only use 'b' in a cave, and 'h' in the hills, etc. To prove that an action is only available in location *X*, it is enough to show that it works in *X* and that it does not work in *one* other kind of location.

6. (2 marks) You may have noticed that right now, a *spring* 's' never occurs in the game. This is because we do not want spring to be generated like all the other location. We want exactly one spring in our game. Put the spring at location (5,5).

Hint. You should not have to change `Intro.tm` for this question.

1.2 The Game - Version B (30 marks)

In this section you are required to extend the game we have considered in the first part of the practical. You are free to extend the game in any way you choose. The only constraints is that the game must satisfy the (positive and negative) requirements given below. It is not even essential that you extend the current game. If you choose, you could develop a game from scratch that satisfies the requirements, but – depending on the game you choose to implement – that could take much more than 25 hours of work.

Exactly 3 marks will be awarded for each of the following requirements that you satisfy:

- (Requirement 1) *Extending state/world*: at least one of your tapes should be (infinitely!) extendible through playing the game.
- (Requirement 2) *Updating current state/world*: you should have a notion of current state, which gets updated (not extended!) throughout the game.
- (Requirement 3) *Randomness*: your TM should make some of its decisions based on the contents of the random tape.
- (Requirement 4) *Output*: your game should produce output to the player (i.e. write to tape 3).
- (Requirement 5) *Input*: your game should allow the player to control the game using input.
- (Requirement 6) *Victory dependence*: your victory should be dependent on at least two different pieces of information. For example: (1) the location of the player and (2) their current inventory/health.
- (Requirement 7) *10 Steps*: it should take the player at least 10 steps to win the game.
- (Requirement 8) *Choice*: there are several different ways for the game to unfold (e.g. different player-input results in different game events).
- (Requirement 9) *Winning*: it is possible to win the game by fulfilling the victory condition (i.e. the Turing machine reached its final state).
- (Requirement 10) *Losing*: it is possible to lose the game (i.e. the Turing machine fails).

Furthermore, 3 marks will be deducted for each of the following requirements that you do not satisfy:

- (Negative 1) Your code should be documented per state.
- (Negative 2) Your code should be documented per line.
- (Negative 3) Your game should include a dictionary `gameDicti` which makes for a playable game.

To satisfy N1 and N2, your code should look something like the first block of code in `Intro.tm` (displayed below). As you can see, there is a description for the state '0', i.e. *read instruction from tape5* and a description for each of the individual lines:

```
## read instruction from tape5
0      ('%', '@', '$', '$', 'p')  2      ('%', '@', '$', '$', 'p')  ('T', 'T', 'R', 'T', 'T')  pick some berries
0      ('%', '@', '$', '$', 'I')  5      ('%', '@', '$', '$', 'I')  ('T', 'T', 'R', 'T', 'T')  print info
0      ('%', '@', '$', '$', '$')  7      ('%', '@', '$', '$', 'x')  ('T', 'T', 'R', 'T', 'T')  invalid action
0      ('%', '@', '$', '$', '.')  666    ('%', '@', '$', '$', '.')  ('R', 'R', 'R', 'T', 'T')  reached end of input without succeeding
```

You probably noticed that many of these requirements are already satisfied by the code you wrote in the introduction questions. Since you are allowed to use this code for your game, you are getting quite a few marks *almost* for free. Almost, because marks for requirements will only be awarded if you show in your demo(s) that you have satisfied them. There are also a few requirements which are not satisfied by `Intro.tm`. You will have to extend the game in such a way that it does satisfy them, and prove this in your demo(s). Of course, you are also free to design your own game from scratch if you like. Just make sure that it satisfies all requirements and that you have shown this in your demo(s).

Please remember that you should not spend more than 25 hours on this assignment!

7. (a) Uncomment the game functions/variables in `Main.py` and provide bodies for them.
- (b) Write a TM game in `Game.tm` (and `Main.py`) satisfying all requirements specified above.
- (c) Uncomment the walk-through function `walkThroughGame()` and the variable `gameInfo` in `Main.py`. Provide bodies for them such that running the function proves that you have satisfied all the requirements except 8.
- (d) Prove that you have satisfied requirement 8, by specifying an alternative sequence of instructions and comments in `gameInfoAlt` in `Main.py`.

2 Submitting your Assignment

When you have finished sections 1 and 2, you are ready to submit this part of your assignment. Please remember to check that you have three files called `Main.py`, `Intro.tm`, and `Game.tm`. Only files with those *exact* names will be accepted by the `submit` program. Put these three files into a single directory and submit them using the following command from a terminal window:

```
submit inf2 inf2a A3 Main.py Intro.tm Game.tm
```

Remember that the questions on this handout are only worth 70/100 marks and that you still need to answer some theory questions on the Assignment 3 website for the remaining 30 marks!

THE END

Plagiarism

There are some general principles in preparing any work:

- You should complete coursework yourself, using your own words, code, figures, etc.
- Acknowledge your sources for text, code, figures etc that are not your own.
- Take reasonable precautions to ensure that others do not copy your work and present it as their own.

If you produce your own unassisted work then you are in no danger of plagiarising. However, if your work requires some help or collaboration then you should follow the guidelines below.

It is acceptable to use general ideas picked up, for example, in discussion with others, or in textbooks, without acknowledgement. In general, if you write the work you submit, in its entirety, yourself then you do not need to include an acknowledgement. An exception is when a really pivotal idea comes from an external source, in which case you should acknowledge that source.

There are occasions on which you will wish to include work written by somebody else within your own work. For example, you may be quoting a passage from a book within an essay, or you may be unable to finish a program yourself and require a piece of code written by someone else. On such occasions, you should copy the original work verbatim together with a full and explicit acknowledgement of the original source. The beginning and end of the copied material must be clearly indicated:

- In the case of copied text (in an essay or written answer to a question), you should enclose the text within quotation marks and explicitly indicate the source of the work close to where the copied material appears.
- In the case of code, you should insert a comment right before the start of the code you have copied or collaborated on, indicating who you have gotten it from or who you have collaborated with in developing it, and then another comment right after the end of the relevant code.

These rules about acknowledging the source of work that is not fully your own, apply irrespective of the type of work it is (prose, code, etc.).

If you collaborate on the preparation of all or part of a piece of work then the collaboration must be explicitly acknowledged, and the extent of the collaboration should be stated as fully as possible by all parties involved.

It is always unacceptable to attempt to disguise someone else's work in order to present it as your own, for example by making cosmetic changes to text, or by changing variable names or altering comments in code.

When in doubt, state explicitly what you have done and the sources of any help (including fellow students, tutors or textbooks) you receive. This will enable your work to be marked appropriately, and will ensure that you avoid suspicion of plagiarism.

In order to prevent others from copying your work and presenting it as their own, set the protection on your practical work directories appropriately. It is also sensible to ensure that any practical submissions that you print are collected promptly from the printers. By all means discuss your general ideas with other students, but do not distribute or lend your solutions to them.

This text was adapted from "Guidelines on Plagiarism" at <http://www.inf.ed.ac.uk/admin/ITO/DivisionalGuidelinesPlagiarism.html>.

Appendix I

API for `tm.py`

tm

\$Id: tm.py,v 100.21 2006/10/30 08:00:00 rvr Exp \$

Copyright (C) 2003-2004 Rogério Reis & Nelma Moreira {rvr,nam}@ncc.up.pt
Copyright (C) 2006 Laura Korte orange.laura@gmail.com

This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version.

This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with this program; if not, write to the Free Software Foundation, Inc., 675 Mass Ave, Cambridge, MA 02139, USA.

@author: Rogério Reis & Nelma Moreira {rvr,nam}@ncc.up.pt & Laura Korte orange.laura@gmail.com

The original is part of FAdo

Modules

[exceptions](#)

[string](#)

Classes

[builtin .object](#)

[TM](#)

[Tape](#)

[exceptions.Exception](#)

[TMexception](#)

[TMerror](#)

[fail](#)

[success](#)

class **TM**([builtin .object](#))

Class for Turing machines

Methods defined here:

__init__(self, tapes, poss, program={}, initstate='@void', fstates=[], blank='@b', star='_', input='?', prompt='Yes?')

 @param tapes: self evident

 @type tapes: tape list

 @param poss: idem

 @type poss: int list

 @param program: a dictionary with the name of the state as key and a list of (char_in,state,char_out,Movement) intructions.

 @param initstate: the initial state if the **TM**

@param fstates: a list of state names considered final, if in any moment the TM passes through one of those states it is equivalent of SUCCESS
 @param blank: the symbol for blank in the tape
 @param star: is the variable in the instructions
 @param input: is the input in the instructions
 @param prompt: is the prompt used when input is required from the user

howmany(self)

@returns: current number of steps executed

load_tape(self, tape, pos=0)

Loads a tape
 @param tape: the tape object
 @param pos: the position to start with
 @type tape: tape
 @type pos: int

show(self)

@returns: a list of strings representing the tapes

step(self)

Executes one step of the TM.

All result control is done via exceptions raised!

stepn(self, n)

Executes n steps

Data and other attributes defined here:

__dict__ = <dictproxy object>

dictionary for instance variables (if defined)

__weakref__ = <attribute '__weakref__' of 'TM' objects>

list of weak references to the object (if defined)

class TMerror(TMexception)

Method resolution order:

TMerror

TMexception

exceptions.Exception

Methods inherited from exceptions.Exception:

__getitem__(...)

__init__(...)

__str__(...)

```
class TMexception(exceptions.Exception)
```

Methods inherited from exceptions.Exception:

```
    __getitem__(...)
```

```
    __init__(...)
```

```
    __str__(...)
```

```
class Tape(__builtin___.object)
```

A Tape class for Turing Machines (infinite in both directions)

To be used only inside TMs

Methods defined here:

```
    __init__(self, initape=[], inipos=0, blank='@b')
```

An initial tape as well as an initial position can be given.

@param blank: the blank symbol to be used by the tape

@type blank: string

```
    __str__(self)
```

```
    bounds(self)
```

@returns: tape bounds

```
    move_left(self)
```

Moves tape one position to the left

```
    move_right(self)
```

Moves tape one position to the right

```
    pprt(self)
```

@returns: a string with a representation of the tape including the position of the head using []

```
    put(self, symbs)
```

Writes list on the tape, starting at the current position

```
    read(self)
```

@returns: current position tape contents

```
    tape2string(self)
```

@returns: a string containing all characters on the tape

```
    write(self, symbol)
```

Writes symbol on the tape, at the current position

Data and other attributes defined here:

```
__dict__ = <dictproxy object>
    dictionary for instance variables (if defined)

__weakref__ = <attribute '__weakref__' of 'Tape' objects>
    list of weak references to the object (if defined)
```

class **fail**(TMexception)

Method resolution order:

```
fail
TMexception
exceptions.Exception
```

Methods inherited from exceptions.Exception:

```
__getitem__(...)
__init__(...)
__str__(...)
```

class **success**(TMexception)

Method resolution order:

```
success
TMexception
exceptions.Exception
```

Methods inherited from exceptions.Exception:

```
__getitem__(...)
__init__(...)
__str__(...)
```

Functions

read_tm_program(file)

```
Reads a TM description
@arg file: file name
@type file: string
@returns: a TM already set with the loaded info.
```

read_tm_program1(lines)

```
Reads a TM description
@arg lines: description of a TM program
@type lines: string list
```

@returns: a TM already set with the loaded info.

read_tm_tape(file)

Reads a tape description

@arg file: file name

@type file: string

@returns: the tape containing the characters of @arg string

@rtype: list

Appendix J

Turing Machine Assignment (Marking Guide)

Dear Marker,

The assignment consists of 3 parts. Therefore there are 3 marking scripts:

```
Mark-Intro.py
Mark-Game.py
Mark-Theory.py
```

Besides these 3 scripts, the directory attached also contains the module `tm.pyc`, the 'data' directory, the two versions of the assignment, a pdf with all the errata, and example solutions of `Intro.tm`, `Game.tm`, and `Main.py`. The students were assigned one of the two versions of the assignment. This should not make any difference for the marking process, but in case you are interested, you find the version the student was assigned to on the first line of their `.txt` file in the `data`-directory. For example, the following means that the student was assigned to `v2`, which is version B:

```
GsyWZmxT    v2
:
:
```

To mark the submission of student `s0233694` in tutorial group `T03`, copy the following files to the marking directory:

```
/home/submit/submissions/inf2/inf2a/T03/s0233694/A3/Intro.tm
/home/submit/submissions/inf2/inf2a/T03/s0233694/A3/Game.tm
/home/submit/submissions/inf2/inf2a/T03/s0233694/A3/Main.py
```

The first thing you should do is make sure the submission was made in time. Check the timestamp of both `Intro.py` and `Game.py` using the following statement:

```
ls -l Game.py Intro.py
```

Anything saved after Nov 10 16:00 is late and should be marked as such. If you

find a late submission, please add a note of this at the end of your marking report (see below). Note that you should also check the timestamp of the theory answers this time. They are stored in the data directory under the name: <student-number>-posttest.dat

Now create a file called 's0233694.marks'. Run the 3 marking scripts and store their results in 's0233694.marks'. If the student has followed the instructions in the assignment this process should be smooth. If not, you may be prompted for some minor corrections or variable names. Please have the assignment handy, so you are able to answer these questions. This is how you run each of the scripts:

```
python Mark-Intro.py
python Mark-Game.py
python Mark-Theory.py <s0XXXXXX>
```

The first two scripts assume you have the student's Intro.py and Game.py in the current directory. The third one requires the data-subdirectory.

After you have run the scripts and saved the answers, please mark R1-R10 and N1-N3 manually and insert the marks where Mark-Game.py has created a space for them. Finally, add up all the marks (imagine that the total is 77) and insert a single line at the bottom of s0233694.marks:

```
=====> TOTAL MARKS: 77/100
```

If this particular student was late in submitting their files, please write the following instead:

```
=====> TOTAL MARKS: 77/100
=====> LATE: Nov 14 17:51 (Theory)
=====> LATE: Nov 14 17:55 (Game.py, Intro.tm, Main.py)
```

Assuming the timestamps on the student's files were Nov 14 17:51 and Nov 14 17:55.

Please note that there are two students with extentions until 4pm, Nov 17th: s0562009 and s0344918. There is also one student with an extention till 4pm, Nov 14th: s0569247. Please do not count their submissions as late if they are submitted before their new respective deadlines. I will send you the theory files of s0562009 and s0344918 on Friday.

We have been given exactly 2 weeks to do the marking, so please return all your <studentnumber>.marks to me by 9am, October 28th. This assignment, like the previous one, takes very little time per student, so I have assigned each of you half of the marking, that should take you about as much time as it took me to write the marking scripts :)

Based on the number of submissions per tutorial group, I suggest the following division of 46 submissions each:

```
(S) default: 1
(S) t01: 9
(S) t02: 8
```


(S) t03: 7
 (I) t04: 12
 (I) t05: 10
 (I) t06: 6
 (I) t07: 9
 (I) t08: 9
 (S) t09: 7
 (S) t10: 6
 (S) t11: 8

Please keep track of frequent mistakes as we would like to compile a list of 'common mistakes' for the students to look over. Any questions or bugs in the marking scripts please let me know ...

Thanks,
 Laura (orange.laura@gmail.com).

```
=====
=== Game Preliminaries (40 marks) ===
=====
```

NB. make sure entire dictionary (introDicti) is uncommented
 in student-file Main.py

Q1. WORTH: 12 marks
 NOTES: Auto Check.
 Q2. WORTH: 10 marks
 NOTES: Auto Check.
 Q3. WORTH: 3 marks
 NOTES: Auto Check.
 Q4. WORTH: 10 marks
 NOTES: Auto Check.
 Q5. WORTH: 3 marks
 NOTES: Auto Check (interactive).
 The marking program will print a demo of the student's intro game
 including comments. There are 14 items they need to prove:

- (1) 'b' is unavailable in a square other than '(c)ave'
- (2) 'b' is available in '(c)ave'
- (3) 'p' is unavailable in a square other than '(f)orest'
- (4) 'p' is available in '(f)orest'
- (5) 'r' is unavailable in a square other than '(m)ountain'
- (6) 'r' is available in '(m)ountain'
- (7) 'f' is unavailable in a square other than '(l)ake'
- (8) 'f' is available in '(l)ake'
- (9) 'h' is unavailable in a square other than '(h)ills'
- (10) 'h' is available in '(h)ills'

- (11) 'n' is available
- (12) 'e' is available
- (13) 'w' is available
- (14) 's' is available

- * Student proves (at least) 14 items: 3 marks
- * Student proves between 10-13 items: 2
- * Student proves between 6-9 items: 1
- * Student proves less than 5 items: 0

Q6. WORTH: 2 marks

NOTES: Auto Check.

=====
 === The Game (30 marks) ===
 =====

NB. Most students will have chosen to reuse the coordinate code provided to them for the introduction part. Because the coordinate code satisfies most of the requirements already, most proofs will relate to this coordinate code. If you have a submission which does not use the coordinate code and you are unsure how to mark (some of) the requirements, please let me know and I will have a look at it.

Also, some of these descriptions are a little abstract. Before you start marking, it is probably a good idea to run to marking script(s) on the answers provided, to give you an idea of what a correct answer could look like.

R1. WORTH: 3 marks

- NOTES: * Proof includes 'state is expanded' or 'new entry is added' or something similar: +1 mark
- * Proof describes how this happens not just this time, but a possibly infinite number of times throughout the game: +2 marks

R2. WORTH: 3 marks

- NOTES: Any example of current state will give 3 marks.
- In the intro-code examples are: player inventory, player health, and current location.

R3. WORTH: 3 marks

- NOTES: If proof includes 'decision made using the contents of the random tape' or something similar: 3 marks

R4. WORTH: 3 marks

- NOTES: Any example of output will give 3 marks.

R5. WORTH: 3 marks

- NOTES: Any example of input will give 3 marks.

R6. WORTH: 3 marks

- NOTES: Need 3 proof items:
- (1) proof that the first condition alone is not enough for victory
 - (2) proof that the second condition alone is not enough for victory
 - (3) proof that both conditions together result in victory
- Assign 1 mark for each of these items that are present in the demo.

R7. WORTH: 3 marks

NOTES: * if the student shows that THERE IS a route to victory which takes at least 10 steps: 1 mark
* if the student also explains why it is NOT POSSIBLE to win in less than than 10 steps: 3 marks

R8. WORTH: 3 marks

NOTES: Semi-Auto Check.
If R9 and R10 are correct, the marking program will assign 3 marks for R8 as well. If either R9 or R10 are not correct, you need to mark this item by hand.
* if the 2 demos differs and this stems from the instructions given by the player: 3 marks

R9. WORTH: 3 marks

NOTES: Auto Check.

R10. WORTH: 3 marks

NOTES: Auto Check.

N1. WORTH: -3 marks

NOTES: * if none of the states have documentation: -3
* if some of the states have documentation, but not all : -1 or -2 depending on the percentage of documented states and the usefulness of the comments provided (marker's discretion)

Please note that many students will reuse the code provided for the intro. Please ignore these blocks when marking this item.

N2. WORTH: -3 marks

NOTES: * if none of the lines have documentation: -3
* if some of the lines have documentation, but not all : -1 or -2 depending on the percentage of documented states and the usefulness of the comments provided (marker's discretion).
Note that the use of colons (:) is allowed to denote repetition.
Here is an example:

```
## lost the game, decrease health
113    ('%', '5', '$', '$', '$') 4    ('%', '4', 'L', '$', '$') ('T', 'R', 'R', 'T', 'T') health - 1
113    ('%', '4', '$', '$', '$') 4    ('%', '3', 'L', '$', '$') ('T', 'R', 'R', 'T', 'T') :
113    ('%', '3', '$', '$', '$') 4    ('%', '2', 'L', '$', '$') ('T', 'R', 'R', 'T', 'T') :
113    ('%', '2', '$', '$', '$') 4    ('%', '1', 'L', '$', '$') ('T', 'R', 'R', 'T', 'T') :
113    ('%', '1', '$', '$', '$') 666    ('%', '0', 'D', '$', '$') ('T', 'R', 'T', 'T', 'T') health - 1 => die
```

This means that the first 4 items all have the same comment.

Please note that many students will reuse the code provided for the intro. Please ignore these blocks when marking this item.

N3. WORTH: -3 marks

NOTES: * if the gameDicti does not contain any game-related info: -3
* if the gameDicti contains some game-related info, but not a lot: -2
* if the gameDicti contains mostly game-related info, but not all: -1
For example, the following dictionary would get -3:

```
gameDicti = { 'a' : "blah1"
              , 'b' : "blah2"
              , etc ...
```

```
=====
=== Theory (30 marks) ===
=====
```

Q1. WORTH: 2 marks

NOTES: Auto Check.

Q2. WORTH: 2 marks

NOTES: Auto Check.

Q3. WORTH: 2 marks

NOTES: Auto Check.

Q4. WORTH: 2 marks

NOTES: Auto Check.

Q5. WORTH: 5 marks

NOTES: Auto Check.

Q5. WORTH: 2 marks

NOTES: Auto Check.

Q6. WORTH: 5 marks

NOTES: Auto Check (interactive).

The marking program will print the QUESTION, SOLUTION, and STUDENT ANSWER.

Please mark according to the number of correct items:

* 1 correct item : 2 marks

* 2 correct items : 4 marks

* 3+ correct items : 5 marks

Q7. WORTH: 4 marks

NOTES: Auto Check (interactive).

The marking program will print the QUESTION, SOLUTION, and STUDENT ANSWER.

* If the student mentions the TM *tape* being infinite: +2 marks

* If the student mentions FSAs having a finite number of states: +2 marks

Q8. WORTH: 3 marks

NOTES: Auto Check.

Appendix K

Study 3: Parameter Estimates for Models 4a–c.

Model	Dependent Variable (Performance on ...)	Parameter	B	SE	95% CI for B	
					Lower	Upper
4a	Pre-Defined Task	Constant	-1.580*	0.631	-2.835	-0.326
		Approach-Avoidance	0.003	0.015	-0.026	0.032
		Mastery-Performance	-0.028	0.016	-0.060	0.005
		Motivation Total	0.019	1.965	0.000	0.038
		Pre-Def: Time Spent	0.468**	0.138	0.194	0.742
		Own-Game: Time Spent	-0.204	0.151	-0.504	0.096
		Gender = Female	-0.267	0.293	-0.858	0.306
		Gender = Male	n/a ^a	.	.	.
	Own Game Task	Constant	-0.667	0.611	-1.882	0.547
		Approach-Avoidance	-0.027	0.014	-0.055	0.001
		Mastery-Performance	-0.007	0.016	-0.039	0.024
		Motivation Total	-0.006	0.009	-0.025	0.012
		Pre-Def: Time Spent	0.205	0.133	-0.061	0.470
		Own-Game: Time Spent	0.109	0.146	-0.181	0.400
		Gender = Female	-0.107	0.283	-0.670	0.457
		Gender = Male	n/a ^a	.	.	.
4b	Pre-Defined Task	Constant	-1.580*	0.631	-2.835	-0.326
	Own Game Task	Constant	-0.667	0.611	-1.882	0.547
4c	Pre-Defined Task	Constant	-1.580*	0.631	-2.835	-0.326
	Own Game Task	Constant	-0.667	0.611	-1.882	0.547

* $p < .05$, ** $p < .01$.

^a This parameter is set to zero, because it is redundant.