

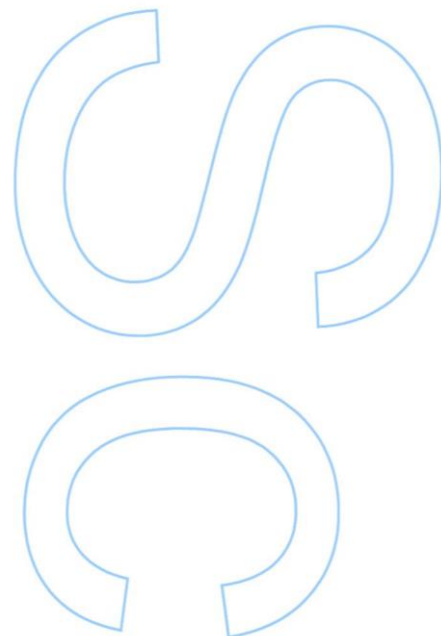
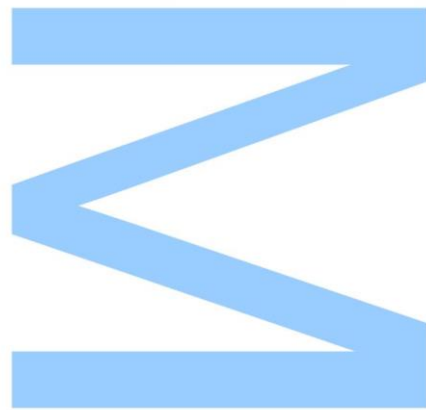
# Assessment of Programming Challenges using Gamification

José Carlos Costa Paiva

Mestrado em Ciência de Computadores  
Departamento de Ciência de Computadores  
2018

## **Orientador**

José Paulo de Vilhena Geraldes Leal, Professor Auxiliar,  
Faculdade de Ciências da Universidade do Porto



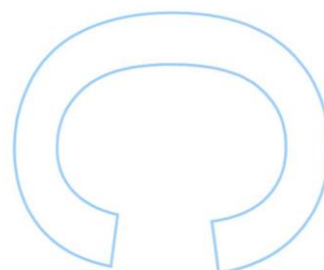
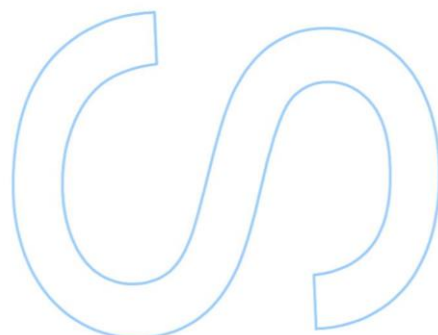
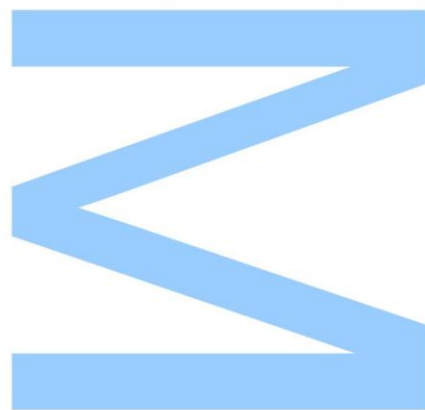




Todas as correções determinadas  
pelo júri, e só essas, foram efetuadas.

O Presidente do Júri,

Porto, \_\_\_\_/\_\_\_\_/\_\_\_\_







# Abstract

Undergraduate programming courses are facing serious challenges, as evidenced by the high dropout and failure rates of students in these courses. Furthermore, even students who achieve their degree tend to demonstrate severe difficulties in programming and designing software systems. Learning to program requires a high level of abstraction as well as the development of practical skills, such as logical thinking, problem-solving, trial and error, debugging, among others. However, the practical component of these subjects is scarce, making it a must to train outside of the classroom.

Automatic assessment tools, primarily developed for programming contests, have been used to enable students to have ready and immediate feedback in or out of the classroom. These tools typically focus on comparing the obtained output with the expected result, returning an accepted value if they are the same or rejected otherwise, which is not suitable for teaching. For this reason, much attention has been devoted to improving the quality and quantity of feedback provided by them. However, providing students with these means for learning does not eliminate the existing issues by itself, since only motivated learners will take the initiative to practice, apply the necessary effort, and persist after repetitive failures.

Lack of motivation is a recurring problem in most areas of education. Several methods to engage students in educational activities have already been proposed, such as problem- and project-based learning, pair programming, mentoring, competition-based learning, among many others. A widespread and encouraging approach to motivate people is gamification. Gamification involves the use of game-thinking and game mechanics to engage people in less interesting or more demanding activities. In the context of programming learning, an already tried approach is to adopt full-fledged games only replacing the usual input devices with code. This model, particularly when it promotes competition, has demonstrated to be quite effective in IBM CodeRally and Robocode. Nevertheless, the effort that the production of this type of games requires hinders their use by the instructors.

This thesis proposes a game-based assessment environment for programming challenges in which students develop a program to control the player, henceforth called Software Agent. In addition to allowing students to benefit from graphical feedback, this environment also promotes competition through competitive evaluation and tournaments. One of the key features of Asura is that it attempts to reduce the necessary effort of building Asura challenges to make it similar

to that of creating traditional programming exercises. In particular, it provides authors with a framework to facilitate the development of Asura games, which offers a higher level abstraction and a number of pre-implemented common functionalities, as well as a command-line interface to scaffold the project structure.

Asura validation is twofold. The first experiment aims to evaluate the effectiveness of the framework in reducing the effort required to build Asura challenges. As a measure of difficulty, the reference are the well-known ICPC problems. The second test attempts to estimate the success of Asura in increasing motivation and, consequently, the practice of programming. Both tests revealed relative success in fulfilling the formalized hypotheses.

**Keywords:** games, gamification, automatic assessment, learning, education, programming, competition, graphical feedback, tournament.

# Resumo

Os cursos universitários com componente de programação estão a enfrentar sérios desafios, como demonstram as altas taxas de abandono e de insucesso dos estudantes desses cursos. Além disso, até os estudantes que conseguem atingir a graduação tendem a demonstrar dificuldades acentuadas a programar e a projetar *software*. Aprender a programar exige um elevado nível de abstração assim como o desenvolvimento de habilidades práticas, tais como raciocínio lógico, resolução de problemas, tentativa e erro, depuração, entre outras. Contudo, a componente prática destes cursos é escassa, tornando-se imprescindível o treino fora da sala de aula.

As ferramentas de avaliação automática, inicialmente desenvolvidas para concursos de programação, foram o suporte encontrado para permitir que os alunos tenham um *feedback* acessível e imediato dentro ou fora da sala de aula. Estas ferramentas, tipicamente, centram-se em comparar o resultado obtido ao resultado esperado, imprimindo um valor aceite se estes forem iguais ou rejeitado caso contrário, o que não é adequado ao ensino. Por essa razão, muita atenção tem sido dedicada à melhoria da qualidade e quantidade de *feedback* fornecido por estas ferramentas. Contudo, dar aos estudantes estes meios de aprendizagem não elimina, por si só, os problemas existentes, pois, somente alunos motivados irão tomar a iniciativa de usar as ferramentas que têm ao dispor, aplicar o esforço necessário, e persistir após falhar repetidamente.

A falta de motivação é um problema recorrente na maioria das áreas de ensino. Vários métodos para cativar os alunos nas atividades educacionais já foram propostos, tais como a aprendizagem baseada em problemas ou projetos, programação em pares, mentoria, competição, entre muitos outros. Uma abordagem já bastante difundida e com alguns resultados encorajadores na motivação de pessoas é a gamificação. A gamificação recorre ao uso do raciocínio e mecânicas aplicados nos jogos para envolver as pessoas em atividades menos interessantes ou exigentes. No contexto do ensino da programação, os jogos já foram usados de forma completa, apenas substituindo os habituais dispositivos de entrada por código. Este modelo, particularmente quando promove a competição, tem-se revelado bastante efetivo, como prova o sucesso dos jogos IBM CodeRally e Robocode. Todavia, o esforço que a criação deste tipo de jogos exige impossibilita o seu uso por parte dos professores.

Esta tese propõe um ambiente de avaliação de desafios de programação baseados em jogos – Asura –, no qual os alunos desenvolvem um programa para controlar o jogador, daqui em diante denominado Agente de Software. Além de fornecer feedback visual, este ambiente promove a

competição através de avaliação competitiva e torneios. Uma das principais características do Asura é o foco na redução do esforço necessário para a construção de desafios, de modo a torná-lo semelhante ao da criação de exercícios de programação tradicionais. Em particular, fornece aos autores uma *framework* para facilitar o desenvolvimento de jogos, que providencia um nível de abstração mais elevado e bastantes funcionalidades comuns pré-implementadas, assim como uma interface de linha de comandos para gerar o projeto.

A validação do Asura é efetuada em duas fases. O primeiro experimento tem como objetivo avaliar a eficácia da *framework* em reduzir o esforço requerido na autoria de desafios do Asura. Como medida para a dificuldade, a referência foram os conhecidos problemas ICPC. O segundo teste tenta estimar o sucesso do Asura em aumentar a motivação e, consequentemente, a prática da programação. Ambos os testes revelaram sucesso relativo no cumprimento das hipóteses formalizadas.

**Palavras-chave:** jogos, gamificação, avaliação automática, aprendizagem, ensino, programação, competição, feedback visual, torneio.

# Acknowledgements

My Master’s studies where a wonderful experience in my life, which undoubtedly provided me with some scientific knowledge and strengthened my critical sense and creative thinking. Hence, I would like to thank everyone who contributed to the completion of this thesis.

Firstly, I would like to express my sincere gratitude to my advisor Prof. José Paulo Leal for the continuous support during this journey, for his clarifications, patience, and long discussions that helped me to sort out relevant details about my work. His guidance helped me all the time of research, development, and writing of this thesis.

Also, I would like to thank my colleague Helder Correia who helped me to publicize the course among students of the first and second year of undergraduate programming courses.

Finally, I would like to thank my family for everything.

This work is partially funded by the ERDF – European Regional Development Fund – through the COMPETE 2020 Programme within project POCI-01-0145-FEDER-006961, and by National Funds through the FCT – Fundação para a Ciência e a Tecnologia (Portuguese Foundation for Science and Technology) – as part of project UID/EEA/50014/2013, and by FourEyes. FourEyes is a Research Line within project “TEC4Growth – Pervasive Intelligence, Enhancers and Proofs of Concept with Industrial Impact/NORTE-01-0145-FEDER-000020” financed by the North Portugal Regional Operational Programme (NORTE 2020), under the PORTUGAL 2020 Partnership Agreement, and through the ERDF.

*José Carlos Paiva*  
Porto, 2018

To you as a reader.

# Contents

<b>Abstract</b>	<b>i</b>
<b>Resumo</b>	<b>iii</b>
<b>Acknowledgements</b>	<b>v</b>
<b>Contents</b>	<b>xi</b>
<b>List of Tables</b>	<b>xiii</b>
<b>List of Figures</b>	<b>xvii</b>
<b>Listings</b>	<b>xix</b>
<b>Acronyms</b>	<b>xxi</b>
<b>1 Introduction</b>	<b>1</b>
1.1 Approach . . . . .	4
1.2 Objectives . . . . .	5
1.3 Contributions . . . . .	5
1.4 Publications . . . . .	6
1.5 Outline . . . . .	7
<b>2 Background</b>	<b>9</b>
2.1 Games . . . . .	9
2.2 Types of Players . . . . .	12

2.3	Gamification . . . . .	13
2.4	Serious Games . . . . .	15
2.5	Game-ification Design . . . . .	17
2.5.1	MDA Framework . . . . .	18
2.5.2	GaML . . . . .	20
2.5.3	UML . . . . .	21
2.5.4	Petri Nets . . . . .	21
2.5.5	Machinations Framework . . . . .	22
2.6	Game-based Learning . . . . .	23
2.7	Competition-based Learning . . . . .	25
<b>3</b>	<b>State of the Art</b>	<b>27</b>
3.1	Platforms for Open Online Courses of Programming . . . . .	28
3.2	Competition-based Programming Learning . . . . .	29
3.2.1	Automatic Judge Systems . . . . .	29
3.2.2	HackerRank . . . . .	30
3.2.3	CodeFights . . . . .	30
3.3	Game-based Programming Learning . . . . .	31
3.3.1	CodinGame . . . . .	32
3.3.2	Greenfoot . . . . .	32
3.3.3	SoGaCo . . . . .	33
3.3.4	Games for Teaching Programming Concepts . . . . .	34
<b>4</b>	<b>Previous Work</b>	<b>49</b>
4.1	Mooshak . . . . .	49
4.1.1	Evaluation Engine . . . . .	50
4.1.2	Feedback . . . . .	52
4.1.3	Exercise Authoring . . . . .	53
4.1.4	Interoperability . . . . .	53



4.2	Enki . . . . .	54
4.2.1	Graphical User Interface . . . . .	54
4.2.2	Gamification Service . . . . .	56
<b>5</b>	<b>Asura</b>	<b>57</b>
5.1	Architecture . . . . .	58
5.2	Asura Builder . . . . .	59
5.2.1	Game Movie Builder . . . . .	59
5.2.2	Game Manager . . . . .	61
5.2.3	Communication Manager-Players . . . . .	63
5.2.4	Wrappers . . . . .	64
5.2.5	CLI Tool . . . . .	65
5.2.6	Overview of Tic Tac Toe . . . . .	66
5.3	Asura Evaluator . . . . .	66
5.4	Asura Tournament Manager . . . . .	68
5.4.1	Round-robin . . . . .	70
5.4.2	Swiss-system . . . . .	71
5.4.3	Single Elimination . . . . .	72
5.4.4	Double Elimination . . . . .	73
5.5	Asura Viewer . . . . .	73
<b>6</b>	<b>Validation</b>	<b>79</b>
6.1	User Acceptance and Efficacy of Asura Builder . . . . .	79
6.2	Effectiveness and Usability of Asura . . . . .	81
6.2.1	War of Asteroids . . . . .	82
6.2.2	Data Analysis and Results . . . . .	84
<b>7</b>	<b>Conclusion</b>	<b>89</b>
7.1	Conclusions . . . . .	89

7.2	Future Work . . . . .	90
<b>A</b>	<b>Overview of Gamified Learning Systems</b>	<b>93</b>
<b>B</b>	<b>Overview of Platforms for Open Online Programming Courses</b>	<b>97</b>
<b>C</b>	<b>Asura Implementation Diagrams</b>	<b>99</b>
C.1	Java Interfaces of Asura Components . . . . .	99
C.2	Game Movie Builder . . . . .	99
C.3	Asura Tournament Manager . . . . .	100
C.4	Asura Viewer . . . . .	101
<b>D</b>	<b>Object Definitions</b>	<b>103</b>
D.1	Match JSON Schema . . . . .	103
D.2	Tournament JSON Schema . . . . .	108
<b>E</b>	<b>Developing a Tic Tac Toe game with Asura Builder</b>	<b>117</b>
E.1	Scaffolding Project . . . . .	117
E.2	Game Manager . . . . .	119
E.3	Game State . . . . .	121
E.4	Game Wrapper . . . . .	124
E.5	Control SA . . . . .	125
<b>F</b>	<b>Validation Exercises Statements</b>	<b>127</b>
F.1	Asura Builder . . . . .	127
F.2	ES6 Course . . . . .	137
F.2.1	ICPC Problems . . . . .	137
F.2.2	Asura Challenges . . . . .	148
F.2.3	Exam Problems . . . . .	157
<b>G</b>	<b>Validation Questionnaires</b>	<b>163</b>

G.1 User Acceptance and Efficacy of Asura Builder . . . . .	163
G.2 Usability of Asura . . . . .	168
G.3 Test of Gamer Personality . . . . .	175
<b>Bibliography</b>	<b>181</b>



# List of Tables

2.1	Mapping from a mundane task into a game . . . . .	14
3.1	Overview of the games to teach specific programming concepts . . . . .	47
4.1	Classification for execution errors thrown during dynamic analysis . . . . .	51
4.2	Mapping from special evaluator’s exit codes to classification . . . . .	52
A.1	Game elements used in selected research papers . . . . .	94
A.2	Details of the experiments conducted in selected research papers . . . . .	95
A.3	Reported outcomes in selected research papers . . . . .	96
B.1	Overview of the platforms for open online programming courses . . . . .	98



# List of Figures

2.1	Diagram of the types of players according to Bartle . . . . .	13
2.2	View of gamification separated from serious games . . . . .	16
2.3	View of gamification containing serious games . . . . .	16
2.4	MDA framework: player and game designer perspectives of the game . . . . .	18
2.5	The current model of the flow state . . . . .	24
3.1	CodeFights: second challenge of a fight against a company’s bot . . . . .	31
3.2	CodinGame: solving a puzzle which provides 2D game-like graphical feedback . .	32
3.3	Interface of Greenfoot . . . . .	33
3.4	SoGaCo: test view of PrimeGame . . . . .	34
3.5	Graphical feedback of a player casting a spell in Catacombs . . . . .	35
3.6	Example quest of the game Saving Sera . . . . .	36
3.7	Screenshot of level 2 of EleMental . . . . .	36
3.8	Screenshot of Kernel Panic . . . . .	37
3.9	Screenshot of Wu’s Castle . . . . .	38
3.10	Screenshot of a custom puzzle of RoboZZle built by another player . . . . .	39
3.11	Screenshot of a puzzle of LightBot version 1 . . . . .	40
3.12	Interface of TALENT programming tool . . . . .	40
3.13	Screenshot of a battle in RoboCode . . . . .	41
3.14	Screenshot of a race in Code Rally . . . . .	42
3.15	Fight between two MUPPETS objects . . . . .	43

3.16	Interface of the PlayLogo 3D game . . . . .	44
3.17	Interface of the Gidget game . . . . .	45
3.18	Leek Wars: fight between two leeks in a two-axis environment . . . . .	45
4.1	User interface of the Creator environment of Mooshak 2 . . . . .	53
4.2	Components diagram of the network of Enki . . . . .	54
4.3	Graphical user interface of Enki . . . . .	55
5.1	Diagram of components of the architecture of Asura . . . . .	58
5.2	Data models exchanged between the game manager and the players . . . . .	63
5.3	3-tier architecture of an Asura SA . . . . .	64
5.4	Sequence diagram of the interaction between Tic Tac Toe game components . . .	67
5.5	Binding of IO streams between the game manager and the SAs . . . . .	68
5.6	Data model of the Asura Tournament Manager . . . . .	69
5.7	Wizard embedded in the Administration GUI to organize tournaments . . . . .	70
5.8	Match mode of Asura Viewer . . . . .	74
5.9	Tournament mode of Asura Viewer – brackets view . . . . .	75
5.10	Tournament mode of Asura Viewer – ranking view . . . . .	76
5.11	Asura Viewer integrated in Enki . . . . .	77
6.1	Results of the user acceptability evaluation of Asura Builder . . . . .	80
6.2	Screenshot of the original Asteroids version from Atari . . . . .	82
6.3	Screenshot of the War of Asteroids . . . . .	83
6.4	Validation of an SA of the War of Asteroids . . . . .	85
6.5	Results of the usability questionnaire of Asura . . . . .	85
6.6	Comparison of quantitative metrics per group . . . . .	86
6.7	Correlation between the different player types and the quantitative metrics . . .	87
C.1	Diagram of the Java interfaces of Asura components . . . . .	99
C.2	UML class diagram of the game movie builder . . . . .	100



C.3	Simplified UML class diagram of the Asura Tournament Manager . . . . .	101
C.4	UML class diagram of the Asura Viewer . . . . .	102



# Listings

D.1	JSON Schema for matches . . . . .	103
D.2	JSON Schema for tournaments . . . . .	108
E.1	Scaffolding a new project with Asura Builder CLI . . . . .	117
E.2	General project structure generated by Asura Builder CLI . . . . .	118
E.3	Game Manager class of Tic Tac Toe . . . . .	119
E.4	Game State class of Tic Tac Toe . . . . .	121
E.5	Java game wrapper of Tic Tac Toe . . . . .	124
E.6	Control SA of Tic Tac Toe . . . . .	125



# Acronyms

<b>CLI</b>	Command-Line Interface	<b>IT</b>	Information Technologies
<b>CS</b>	Computer Science	<b>JAR</b>	Java ARchive
<b>DSL</b>	Domain Specific Language	<b>JSON</b>	JavaScript Object Notation
<b>ES</b>	EcmaScript	<b>LMS</b>	Learning Management System
<b>GaML</b>	Gamification Modeling Language	<b>LTI</b>	Learning Tools Interoperability
<b>GUI</b>	Graphical User Interface	<b>MDA</b>	Mechanics, Dynamics and Aesthetics
<b>GWT</b>	Google Web Toolkit	<b>MOOC</b>	Massive Open Online Course
<b>ICPC</b>	International Collegiate Programming Contest	<b>SA</b>	Software Agent
<b>IDE</b>	Integrated Development Environment	<b>UML</b>	Unified Modeling Language
<b>IMS</b>	Instructional Management System	<b>XML</b>	eXtensible Markup Language



# Chapter 1

## Introduction

In recent years, we have witnessed a rapid growth in the industry demand for computing expertise, driven by the drastic impact of Information Technologies (IT) in human interaction, enterprise productivity, and information gathering and dissemination [56]. This demand is twofold. On the one hand, the various fields of the industry require professionals able to use existing computational solutions targeted to their areas. On the other hand, the job market is insatiable in the quest for programmers (including software developers and computer scientists) who can develop these solutions, explore top technologies, and find answers to computational problems [7, 38].

On the track of these requirements, we are currently facing a long-expected increase in the number of enrollments in programming-related courses [37, 74]. Paradoxically, the number of graduates is not reflecting these advances and remains insufficient, particularly due to high failure and dropout rates that chase these courses during the initial years [81, 113, 114]. Furthermore, even students who achieve their degrees may still have many troubles in programming [135] and designing software systems [63, 129].

The difficulty of teaching and learning to program is widely accepted among educators [173]. Novice programmers are daunted by the complexity of the domain, which requires correct understanding of abstract concepts as well as the development of practical skills, such as logical thinking, problem-solving, trial and error, and debugging [141, 166]. Programming languages taught are also seen as obstacles to engagement in subjects [135], due to their poor design and often ambiguous syntax [158]. These issues hinder programming learning, as students tend to classify the subject as boring and lessons hard to follow, a sentiment which is further intensified by their perceptions that the teaching methods are not adequate, and the practice in classes is insufficient [101].

The traditional teaching methods as used in other disciplines, supported by textbooks, slide presentations, and lecture notes, are useful to introduce concepts and present readable solutions but inadequate to programming students. They present programs in a static way (i.e., the final product), typically, containing the ideal solution rather than the dynamic process of achieving one solution which is not the best but has a similar effect, like frequently happens during

programming [18]. This can give the wrong idea that the process is linear and straight, and cause frustration, self-punishment, and loss of motivation when students realize that they are unable to find a simple solution by themselves. When the expository material is accompanied with pen-and-paper exercises solved during classes, students are able improve logical thinking and problem-solving using more natural languages (e.g., pseudocode) and far less strict environments. Nevertheless, novices also need to be aware of the real-world development process, which involves the analysis of the execution flow and intermediate outcomes of the program, the use of real syntax and structural elements of the chosen language, the handling of errors, etc [152]. To this end, programming courses typically dedicate some time to hands-on exercises on computers during laboratory classes.

However, this amount of time is scarce considering the quantity of knowledge that students need to assimilate. For instance, in a regular sixteen-week semester, introductory programming courses often do not have more than twelve programming assignments [16], which is both too much for teachers to provide meaningful individualized feedback to students and too little for novices to master programming skills. Therefore, instructors typically rely on automated assessment systems to increase the number of exercises that students complete, by granting instant and effortless feedback on attempts to solve an exercise [43]. Yet, since these tools are primarily developed for programming contests, their feedback usually sums up to a single yes (i.e., accepted) or no (i.e., wrong answer), which can cause undesirable behaviors [117].

The use of automated assessment tools not only supports instructors during lab classes, but also enables students to have feedback easily and immediately outside the classroom, and thus promotes self-advancement through practice anytime and anywhere. Hence, much attention has been devoted to improve the quantity and quality of feedback [102, 197] as well as to evaluate other program features, such as code quality [188] and comments [133]. Nevertheless, providing students with these tools does not mitigate the existing issues by itself [175], since only motivated learners will take the initiative to practice, put the necessary amount of effort, and persist after repetitive failures [120, 165].

Lack of motivation is one of the most outstanding problems in all areas of education. For this reason, researchers have endeavored to find and experiment methods that could engage students in learning activities. Consequently, several approaches have been purposed such as problem-based learning [131, 151], mentoring [45], storytelling [108], pair programming [88], competition-based learning [54, 125], and project-based learning [17]. A widespread encouraging approach to motivate people is gamification, regularly re-emerging under terms edutainment, ludic design, serious games, game-based learning, or playful interaction. Gamification involves the utilization of game-thinking and game mechanics to engage people [204], and has already been applied in diverse areas such as finance [98], education [32, 100, 181], health [60, 139], human resources [70, 137], news [47, 182], productivity [147, 153], sustainability [71, 126], and tourism [199].

The use of gamification in educational contexts is gradually being accepted as a promising and effective strategy [106] because it is well established that it fosters students' motivation and has



a positive impact on knowledge acquisition and content understanding [48, 159, 194]. Games are also linked to beneficial behaviors, such as becoming more active and socially engaged [69], focused and attentive to details [82], creative [103], receptive to new experiences [195], and persistent [195]. Moreover, games are used by persons of any gender, ethnicity, age, and socioeconomic level [67].

Even though gamification might seem like the elixir for healing all ills, it neither fixes badly designed activities [89] nor works like an homogeneous multipurpose self-adhesive [84, 116] nor has positive effects in every situation [20, 44]. In 2012, Gartner has predicted that, by 2014, 80 percent of the implementations of gamification would fail to meet their goals, primarily due to poor design [26]. Gamification methods tend to prioritize the train or educate elements over the fun, which is generally a bad idea since people notice they are being pushed into doing something. For instance, most gamification methods are just another kind of reward-based learning, which varies on the extrinsic motivators they add (e.g., badges, points, status, progress bars, or leaderboards), to raise interest [10, 84, 85, 91, 104, 156, 160, 191]. Rewards neither foster correct changes in attitudes and behaviors or longtime commitment [136]. On the contrary, they undermine the intrinsic interest that one might have to perform a task [136]. Completing the activity becomes a means to obtain the reward rather than to develop skills, and the best way to solve the task is the first to come to mind, taking risks or exploring are not options because it can result in a loss [111].

Students are motivated when they feel capable of succeeding in a given task if enough effort is applied, and they understand and value the task's outcome [31]. In addition to that, effort and persistence are higher while students assign the responsibility for their failures to insufficient effort, misconception about what to do, or the use of a bad strategy [30]. Games fulfill these needs by having clearly defined rules, increasingly difficult goals, and immediate graphical feedback or rewards for achieved goals, and giving to the players a sense of control over their own actions [76]. Other game elements such as make-believe, storytelling, and competition provide an extra stimulus and enrich the experience.

An already tried approach to foster programming practice is to use a full-fledged game, in which the player does not control its actions with common input devices but rather using a program. These approaches are generally enhanced with competition to increase the challenge and retain the attention of participants for a greater amount of time. Some examples that have demonstrated a great acceptance among the community are the IBM CodeRally [140] – a Java game-based car rally competition presented at the 2003 ACM International Collegiate Programming Contest (ICPC) World Finals – and Robocode [128] – a Java-based virtual robot game. Although, developing this kind of challenges involves a complex and time-consuming process that often teachers are not ready to perform. Moreover, learning programming is an incremental process in which the introduction of a new concept should be accompanied by exercises to assimilate information, suggesting that a single game may be insufficient.

## 1.1 Approach

This thesis focuses on alleviating the problem of lack of programming practice in undergraduate programming courses. To this end, it proposes a game-based assessment environment for programming challenges in which students code a program that controls the game player, henceforth called Software Agent (**SA**). The feedback provided by the evaluation environment consists of a movie displaying the behavior of the **SA** during the game. These challenges are boosted with competition, among all submitted **SAs**, in the form of tournaments such as those found on traditional games and sports, including knockout and group formats. The outcome of the tournament is presented on an interactive viewer, which allows to see each match of the tournament and the rankings. Due to the emphasis on competition, the environment was named Asura. The name comes from the Hindu mythology, in which Asura is a class of deities who are obsessed with power, wealth, and ego. When an Asura does not win, it questions, challenges and attacks the others to get a share of what they have won.

One of the key features of Asura is that it attempts to reduce the necessary effort of building Asura challenges. For that, it provides authors with a framework, named Asura Builder, which includes a game movie builder with an interface to build the JavaScript Object Notation (**JSON**) movie object seamlessly as well as an abstract game manager, a common interface for game state, and facilities to communicate with **SAs** seamlessly. A Command-Line Interface (**CLI**) tool for scaffolding Asura challenges with ease is also part of Asura Builder. Besides generating the required infrastructure for building challenges, this tool also allows to replace the game manager with pre-built game managers, add support for coding **SAs** in a different programming language, among others. As a benchmark for the difficulty of developing Asura challenges, the reference are the well-known **ICPC** problems whose creation requires coding both a solution program and a test case generator.

The architecture of Asura is composed of four new components, namely Asura Builder, Asura Viewer, Asura Evaluator, and Asura Tournament Manager. These components integrate with Mooshak 2 – the new version of Mooshak [124] – and its pedagogical environment, named Enki [156]. Each of these components plays a role in the architecture, as described below.

**Enki** is the pedagogical environment of Mooshak 2, which incorporates Asura. Enki already integrates with several services, such as a gamification service to engage students through badges, leaderboards, and other game mechanics, and an educational resources sequencing service to provide different learning rhythms according to students' capabilities. Enki can present resources of two types: evaluative (e.g., code, diagram, quiz, and game exercises) and expositive (e.g., PDF and videos).

**Asura Builder** is an independent component developed to facilitate the creation of Asura challenges, consisting of a framework and a **CLI**. It produces a Java ARchive (**JAR**) file containing the game binaries, which can be uploaded into Mooshak via its administration interface.

**Asura Viewer** is the component responsible for transforming the **JSON** objects, produced during the game execution, into graphical game-like feedback, that is displayed to the learner.

**Asura Evaluator** extends the evaluator component of Mooshak 2, which is responsible for grading a submission based on a set of tests, to support game-based assessment. In particular, it manages multiple submissions concurrently as part of the same evaluation.

**Asura Tournament Manager** is the component which manages the tournaments. It has a wizard embedded into Mooshak's administration Graphical User Interface (**GUI**) to set up tournament parameters, that are then sent to the class implementing the selected tournament format.

## 1.2 Objectives

The approach detailed in Section 1.1 has two main objectives. The first is to provide instructors with a way to build game-based programming challenges which has a comparable cost to that of creating traditional programming exercises. The second is to use those games to foster students' engagement in learning activities and, thus, increase time dedicated to programming. These goals can be formally described by the following hypotheses:

**Hypothesis H1** It is possible to build game-based programming challenges with a similar amount of effort to that of creating an **ICPC**-like problem.

**Hypothesis H2** The proposed game-based programming challenges are capable of motivating and increase practice time of students.

These hypotheses were validated in separated experiments, as described in Chapter 6.

Beyond these objectives, it is also a purpose of this thesis to survey the state of the art on gamification in education, games that teach specific units of programming, and platforms that provide game-based educational programming challenges.

## 1.3 Contributions

The major contributions of this thesis include (i) the development of a game-based programming assessment environment – Asura – that aims at increasing programming practice time of undergraduate students by motivating them to code outside classroom, (ii) the creation of a framework to build game-based programming challenges with a similar difficulty to that of creating **ICPC**-like problems, and (iii) the review of the state of the art on gamification and games for teaching programming. Other minor contributions are (iv) the definition of a compact

JSON format to represent game-like graphical feedback and (v) another to describe a tournament, (vi) the creation of an interactive viewer for tournament and game movies, and (vii) a Java library for organizing tournaments.

## 1.4 Publications

This thesis consolidates a number of research works already published and presented in international journals, conferences, and symposiums. Hence, the content of this thesis does not report novel results by itself, but rather gives a comprehensive description and further details of the published work. In particular, this concerns the following presentations and publications:

**Presentation I.** José Paulo Leal and José Carlos Paiva, [Asura: An Environment for Assessment of Programming Challenges using Gamification](#). In *ACM-ICPC 11th Competitive Learning Institute Symposium (CLIS)*, Beijing, China, April 2018. An Abstract<sup>1</sup> submitted for acceptance.

**Publication I.** José Carlos Paiva and José Paulo Leal, [Asura: A Game-based Assessment Environment for Mooshak \(Short Paper\)](#). In *7th Symposium on Languages, Applications and Technologies*, SLATE 2018, ISBN: 978-3-95977-072-9, OASICS Vol. 62, ISSN: 2190-6807, DOI: 10.4230/OASICS.SLATE.2018.9

**Publication II.** José Carlos Paiva, José Paulo Leal, and Ricardo Queirós, Authoring Game-based Programming Challenges to Improve Students' Motivation. In *21st International Conference on Interactive Collaborative Learning*, ICL 2018, (to be published)

Presentation I briefly introduces the concept of Asura, how it works, and the initial design of the experiment that would be conducted to validate Asura. This presentation has been preceded by a reviewed one-page abstract.

Publication I describes Asura, its concept, architecture, and components. It does not contain the validation of the tool, but rather a detailed description about how the experiment is going to be conducted. This is a short paper (8-pages) submitted to a peer-reviewed conference and presented at Guimarães, Portugal.

Publication II presents the authoring component of Asura – the Asura Builder –, giving an extensive overview of the framework and the CLI tool. It also describes the validation of the component (Hypothesis H1) and reports its results. This is a full paper (12-pages) submitted to a double-blind peer-reviewed conference and presented at Kos Island, Greece. The paper will be published in the conference proceedings, which will be added to the Springer series “Advances in Intelligent Systems and Computing” as ICL2018 Proceedings.

<sup>1</sup>[https://ciiwiki.ecs.baylor.edu/index.php/Asura:\\_An\\_Environment\\_for\\_Assessment\\_of\\_Programming\\_Challenges\\_using\\_Gamification](https://ciiwiki.ecs.baylor.edu/index.php/Asura:_An_Environment_for_Assessment_of_Programming_Challenges_using_Gamification)

## 1.5 Outline

This thesis is composed of seven chapters. The contents of each of the next chapters are described below.

**Chapter 2 – Background** presents the necessary definitions and theoretical knowledge for the better understanding of the contents of this thesis. In particular, it gives a formal definition for games, gamification, and serious games. Then, it presents some of the existing methods and languages to formalize gamification and game elements and studies how different persons interact with games. Finally, it explores how games and competition are being used in education.

**Chapter 3 – State of the Art** presents systems that provide gamified open online programming courses, programming learning platforms which stand out for their competitive side, games that teach a specific programming unit, and platforms in which learners' code controls an element of a game.

**Chapter 4 – Previous Work** describes systems and tools that were already described in the literature and are crucial parts of the current work.

**Chapter 5 – Asura** provides an in-depth overview of Asura, including details on its architecture, design, and implementation, as well as the core functionalities.

**Chapter 6 – Validation** describes the experiments conducted to evaluate Asura from the perspective of both, authors and learners, as well as the analysis performed over the collected data and its outcome.

**Chapter 7 – Conclusion** summarizes the main contributions of this thesis, goals achieved, and failures. Then, it outlines the future work.



## Chapter 2

# Background

Motivation has been found to be a useful indicator of students' achievement [9]. When students are engaged, they participate in discussions, interact with colleagues and teachers, and do additional work about the subjects [92]. Thus, much work has been devoted to finding ways to foster students' engagement.

From the different methods, game-based approaches are amongst the most successful. Both students and non-students already spend a significant part of their leisure time playing games [150], so it was expected that games have something that engages people. Hence, the idea is to use the elements that provoke engagement to create a learning environment in which students can enjoy the time they spend while learning.

The purpose of this chapter is, firstly, to give a proper definition of games, gamification, and serious games. Then, it aims to understand how different players react to games. Finally, it describes how different game elements are being used in education, detailing the competition element.

### 2.1 Games

Defining the term “game” is complex, subjective and arguable, because there is a multitude of games each possessing many different characteristics. In its simplest form, a “game is something we can play”. This is a generally accepted definition, but it does not set any boundaries to what can be classified as a game or not. For example, it is possible to play with a toy, even though it is not a game.

Therefore, there is a need to set conditions to identify games accurately. Many definitions were already given in the literature, and none of them was accepted to its greatest extent. Each author focuses on a subset of crucial components that defines a game in its personal opinion or for the study he conducts.

In one of the first studies [95] on this subject, Johann Huizinga tried to relate the play element

with culture, by defining “play” as

a free activity standing quite consciously outside “ordinary” life as being “not serious”, but at the same time absorbing the player intensely and utterly. It is an activity connected with no material interest, and no profit can be gained by it. It proceeds within its own proper boundaries of time and space according to fixed rules and in an orderly manner. It promotes the formation of social groupings which tend to surround themselves with secrecy and to stress their difference from the common world by disguise or other means.

In this definition, playing is an utterly absorbing activity that happens outside of ordinary life, which is not serious neither associated with profit. This activity has rules and takes place within its boundaries of space and time. Another interesting factor is that it tends to create social groups that separate the players from the rest of society.

Albeit this definition points out some common aspects of play, such as the idea that it is not serious but still absorbs the attention of players, it does not help to define a game. For example, a game may fail to get the attention of the player, but it is still a game. Moreover, watching movies also fits this definition.

Elaborating on the previous work, Caillois [36] highlights some qualities of games, namely: it is not obligatory; it is limited in space and time; its result is unpredictable; it does not produce any tangible product; it has rules; it is separated from reality. Some of these ideas are borrowed from the previous definition, but it also introduces some new, particularly the understanding that it is a voluntary, uncertain and make-believe activity. Although these ideas seem to make sense, there are still cases where they do not apply (e.g., there is not a make-believe element in a Tic-Tac-Toe game). Also, this definition applies to other activities, such as theater, which are not games.

In a more scientific approach, games were defined as “an exercise of voluntary control systems, in which there is an opposition between forces, confined by a procedure and rules to produce a disequibrial outcome” [11]. This description can be broke down into four parts:

- “an exercise of voluntary control systems” – players are free to join or not the game;
- “there is an opposition between forces” – games have one or more goals and at least two sides trying to win;
- “confined by a procedure and rules” – games must have rules and a series of steps;
- “produce a disequibrial outcome” – the initial state of the game was balanced, but there must be a result, and it should define a winner.

Even if this definition underlines some critical concepts of games, it does not encompass most



single-player games since there is only one player struggling to win. Also, most games can end in a draw, which is not included in the definition above.

As an attempt to extend the work above, Salen & Zimmerman proposed that game “is a system in which players engage in an artificial conflict, defined by rules, that results in a quantifiable outcome”. This definition brings up the engagement which is essential to the success of a game, but not essential to the definition of a game. It also revives the idea of “artificial”, which was borrowed from the earlier definitions. However, this word is now classifying the conflict, not the game itself, which may seem more reasonable.

A problem with the previous definition is that it lacks some key aspects of games, such as goals and the influence of players in the outcome. In fact, every definition fails at some point, and it is not the purpose of this work to bring in a consensual definition, but preferably one that highlights the reason for exploring games. Thus, in the context of this dissertation, we shall define a game as **an activity, with one or more decision-makers, which blends a set of rules, a challenge (goal-based or competitive), and a quantifiable outcome, to become a source of engagement for players.**

This definition contains concepts of each of the earlier approaches, but also from other authors. The primary ideas are:

**Activity:** An activity is something that is done by applying intellectual or physical effort.

**Decision-makers:** The game must have at least one participant, who is responsible for making the needed decisions during the game.

**Rules:** Rules are necessary to a game. They provide a structure to play it, by delimiting the possible actions of the decision-makers (players).

**Challenges:** Games do not make sense without a challenge. A challenge is a task or a group of tasks that determines the purpose of the game (i.e. what a player should do during the game).

**Goals:** Even if the goal is very simple, it must exist. A goal is an in-game state that a player should reach through its decisions.

**Competitive:** Although competition is not required in a game, most of the time it distinguishes an engaging game from the others. A competition requires at least two decision-makers playing the game, possibly at different times.

**Quantifiable outcome:** At the end of the game, the player has either won, tied, lost or received an amount of points. This is what usually separates a game from other play activities (e.g., theater and guitar).

**Source of engagement:** The engagement of the players should be the most significant concern of every game designer. A game which is not engaging is meant to fail. However,

engagement is subjective and cannot be measured. Thus, games must blend the previous game concepts in such a way that it lets the door open for engagement.

## 2.2 Types of Players

Games are generally a rich source of motivation. Their features, such as rewards, goal achievement, and competition, promote the release of dopamine – a chemical responsible for motivation – from the brain. When players are motivated enough (i.e., they enter a state called flow [145]), their level of concentration highly increases, and they lose the awareness of the outside world, making them capable of taking actions that they never thought they would or could take.

Although games are motivators for people in general, players may be engaged by different reasons. So, it is essential to know the player before designing a game or a gamified system. According to a study [122], there are four different kinds of fun that players seek in a game:

**Hard Fun** happens when a player desires to test his skills and beat the game.

**Easy Fun** is the enjoyment provided by experiencing new things.

**Altered Emotional State Fun** occurs whenever the player likes the emotion that a game passes to him.

**Social Fun** is when the player is not interested in the game itself, but rather in socializing through the game features.

These kinds of fun are roughly translated into the four types of players identified by Bartle [14], which can be described as follows:

**Achievers** These players are always seeking to win or achieve something. They are reward-driven, and once they stop achieving, they are likely to give up.

**Explorers** An explorer likes to investigate the world and discover new things about it. They are, particularly, attracted by easter eggs<sup>1</sup>.

**Killers** Players of this type have a similar desire to win as achievers. Although, they need someone else to lose, and to watch their win. Furthermore, they want the others to venerate and respect them.

**Socializers** A socializer plays to interact with the others, not directly because of the game. They use the game as a backdrop to create social relationships.

---

<sup>1</sup>A hidden message, a secret feature, or an intentional inside joke left by the designers or developers.

Bartle has distributed players in two axes according to the interests of the players. The x-axis measures if the player is more interested in the underlying world that the game provides, or the players that play against/with him/her. The y-axis assesses the doing-to (acting) against the doing-with (interaction). The distribution of the types of players is presented in Figure 2.1.

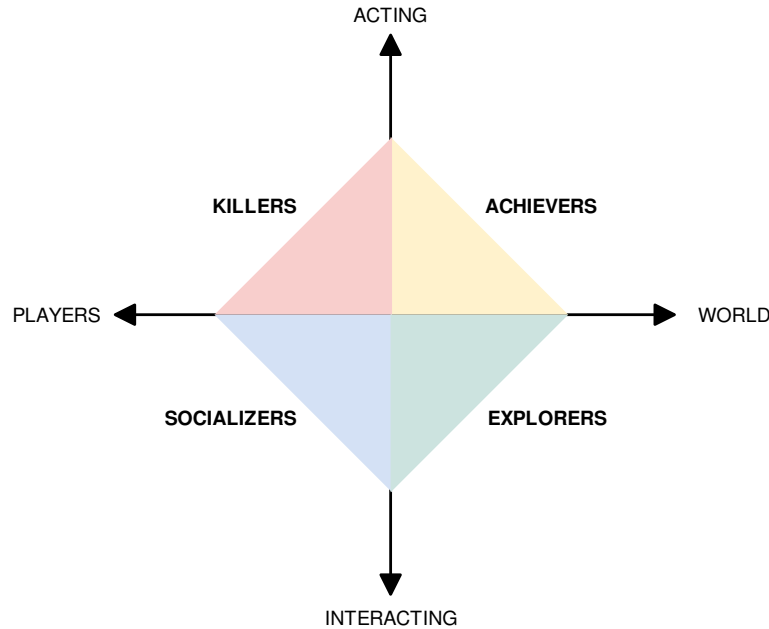


Figure 2.1: Diagram of the types of players according to Bartle

Although the diagram of Figure 2.1 roughly separates the types, players are not exclusively of one kind or the other. Typically, they have characteristics of each type, that may vary throughout their life and from game to game. The average player has 80% socializer, 50% explorer, 40% achiever, and 20% killer [204].

## 2.3 Gamification

As kids, we usually made mental challenges with tedious or repetitive tasks. We did slalom with imaginary obstacles while walking or riding a bike, limited ourselves to take just one step per floor tile, tried to start and stop a chronometer as fast as we could, or balanced our body while walking on a thin surface. If we failed these insignificant challenges, we have felt bad inside for a while or punished ourselves. But when we succeeded, we felt like heroes and, possibly, rewarded ourselves.

We were unconsciously turning our mundane tasks into a game, to motivate ourselves to complete them or just to enjoy that time. Each of these games matches the definition of a game presented in Section 2.1, i.e., they are activities which have rules, challenge(s), and a quantifiable outcome, and they are used as a source of engagement. Table 2.1 picks up the activity of walking over the floor tiles with at most one step per tile to demonstrate how these tasks can be transformed into a game.

Game Concept	Instance
Activity	Walking over the floor tiles with at most one step per tile
Decision-maker(s)	Kid
Rules	{ Do not take more than one step per tile; Walk only on floor tiles; }
Challenge	Goal-based. Goals: { Walk from a location A to a location B; }
Quantifiable outcome	Success or Fail
Source of engagement	Overcome the challenge

Table 2.1: Mapping from a mundane task into a game

Converting a task into a game is a significant step to push us to complete it because we are highly stimulated by problem-solving (i.e., overcome the challenge in this case). However, just beating the game is usually not a high motivation. As members of a society in which we are worth our achievements, players need to show off their successes, to have a sense of progress, to collaborate or to beat other players, and to receive something for their effort, as studied in Section 2.2. Thus, additional elements typically found in video games, such as indicators of progress, rewards, long-term and short-term quests, frequent and precise feedback of their evolution and other people, are also essential.

When game-thinking and game elements come together, our brain releases dopamine which stimulates us. Therefore, even though the primary role of games is entertainment, the potential of their components for motivating people can be used for other purposes. This strategy of engaging people has been clustered under the term “gamification”. To the best of the author’s knowledge, this term was introduced in the early 21st century by Nick Pelling, who started wondering whether applying a game-like user interface design would make electronic transactions more enjoyable. Still, the term was not widely adopted until 2010 [57], under broader contexts. Since then, many efforts towards a definition of gamification have been made, such as

At its root, gamification applies the mechanics of gaming to non-game activities to change people’s behavior. When used in a business context, gamification is the process of integrating game dynamics (and game mechanics) into a website, business service, online community, content portal, or marketing campaign in order to drive participation and engagement. [33]

The use of game mechanics and experience design to digitally engage and motivate people to achieve their goals. [27]

The use of game design elements in non-game contexts. [58]

The process of game-thinking and game mechanics to engage users and solve problems. [204]

The most widely used definitions are the last two, which have distinct ideas about the extent of gamification. While the former [58] excludes serious games (described in Section 2.4) from gamification (i.e., only non-game activities which include game design elements are gamified), the latter is broader. Although there is no definition globally accepted, for the purpose of this thesis, the latter interpretation will be adopted.

Gamification has been largely explored in several fields, ranging from finance [98] to education [32, 100, 181], health [60, 139], human resources [70, 137], news [47, 182], productivity [147, 153], sustainability [71, 126], and tourism [199].

At the enterprise level, two of the cases of higher success are Nike+ and Foursquare. Nike+ [115] is an application that attempts to get people active by tracking distance, speed, and time of their runs. This data is stored to calculate the progress of the user in the next runs, and to compete with others. Competition comes in the form of leaderboards for different time frames, namely weekly, monthly, all-time, among others. Foursquare [73] is a service to search and discover places for leisure activities. Apart from rating and leaving feedback of sites, users can check-in whenever they visit a place and earn badges or statuses for their activities. The status of the user in a particular site can be lost if one of his friends visits it more frequently.

Even though gamification might seem a panacea, it does not automatically produce motivation. More than that, it can have the opposite effect. In 2012 Gartner said that by 2014, 80% of the gamified applications would fail to meet their objectives [26] because their focus was in rewarding users with points and badges, rather than in other game design aspects, such as promoting competition and collaboration, or defining a meaningful game economy. In this way, the goal of the gamification becomes misaligned either with the goal of the player and the goal of the activity. If the player was already interested in doing the activity and just needed a little push to do it, the badges given meaninglessly will just deviate him from the objective of the activity. Also, if people notice that gamification is trying to manipulate their behavior, they are likely to disengage from it.

The fact that gamification has downsides when it is not well applied should not discourage its use, but rather encourage to a preliminary study of the game elements adequate for the activity and that do not affect its desired outcome. Like the kid does to his tasks, the goal stays intact (e.g., go from a location to another), what is added is the challenge and the feeling of achievement.

## 2.4 Serious Games

Serious games are games with another purpose than entertainment [183]. This definition may cause some confusion on to which extent serious games are different from gamification. The

literature is also not clear about this issue since different ideas are under consideration, of which two stand out.

On the one hand, gamification is “the use of game design elements in non-game contexts” [58]. This definition separates gamification from serious games, as depicted in Figure 2.2. While gamification consists of using elements of games that do not constitute complete games, serious games are complete games not primarily for entertainment.

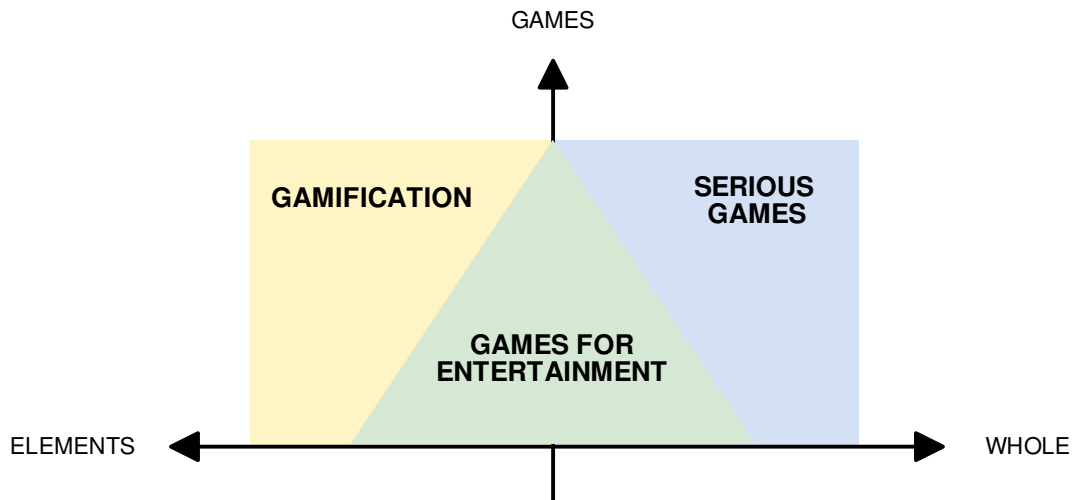


Figure 2.2: View of gamification separated from serious games [58]

On the other hand, gamification is “the process of game-thinking and game mechanics to engage users and solve problems” [204]. This definition unites several concepts such as serious games into gamification, as presented in Figure 2.3.

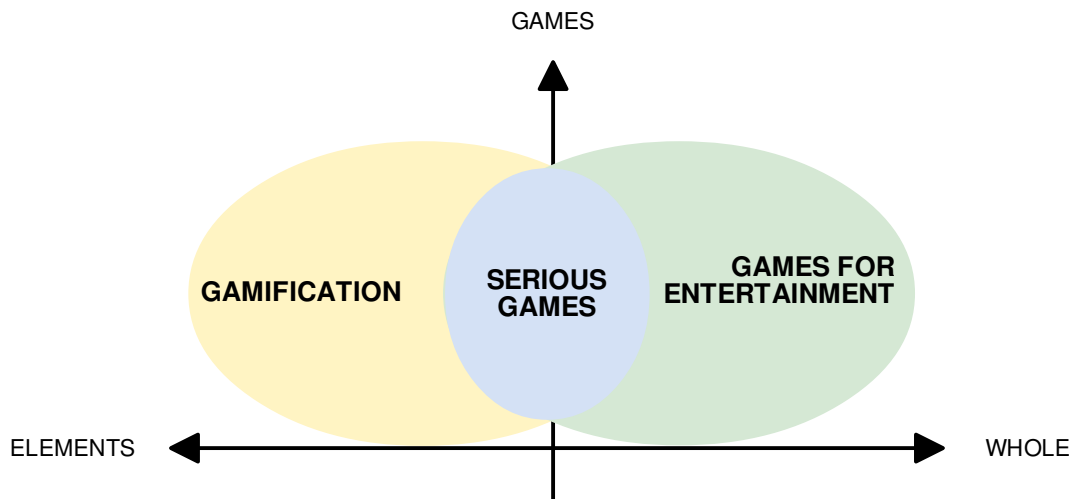


Figure 2.3: View of gamification containing serious games [204]

As stated in Section 2.3, this dissertation follows the latter view. Serious games and other gamified applications can both be seen as games (from the definition of games provided in Section 2.1) with a primary purpose that is not entertainment. What distinguishes them is that

serious games add a purpose on top of games for entertainment, instead of adding game-thinking and mechanics to an activity which is not a game. Also, the additional purpose of serious games is typically education (in its various forms). However, both fit the definition of gamification in that they use game-thinking and game mechanics to engage users while solving problems.

One of the most known serious game is America's Army [149]. The United States military developed this game as a recruiting tool. It is a tactical squad first-person shooter multi-player game (with single-player training sessions), which can be played both by players who seek for entertainment (playing as a soldier) and within the military as a training environment for skills such as intelligence, first aid, survival, cooperation, among others. To enlisted military members who are preparing a mission, enjoyment is not essential, but a supplement.

Beyond the military, serious games have been used in several areas of the industry with relative success, such as education [72, 178], health [78], and sustainability [107].

## 2.5 Game-ification Design

The goal of gamification is quite different from that of games since the former aims to enhance motivation in diverse activities whereas the latter is intended for pure entertainment. Hence, there is also a clear distinction in the way they are designed [132], starting from the base which is either a non-game activity (gamification) or an idea of fun (games) to the phase where game metrics are defined. Nevertheless, the process of gamifying activities hugely depends on principles of game design theory.

Salen and Zimmerman [176] defined a set of game design fundamental principles to run in an iterative process comprising the (1) understanding of graphical design, system, interactivity, and players' choices, actions, and outcomes, (2) the definition of rules, game experience, game representation, and social interaction, and (3) the connection of the rules to the play, pleasures, meanings, ideologies, and stories that they create. Brathwaite and Schreiber [25] claim that the posterior integration of the identified game elements must be considered, introducing the concept game atoms, i.e., the "smallest parts of a game that can be isolated and studied individually". Therefore, design games would be a composition of atoms. Based on this idea, a study [169] has introduced the 10 atoms of a successful massive multiplayer online game.

Once game elements are compiled into game design fundamental principles, it is necessary to follow a set of standards, guidelines, and criteria to assemble them in a deliberate manner. Even though some authors argue that game design is too complex to be described by a formal procedure [49], its need led to the development of multiple frameworks [61, 66, 97, 176] and formal modeling languages/methods [93]. The next subsections present some of these frameworks and formal languages, starting with the Mechanics, Dynamics and Aesthetics (MDA) framework to establish a typical separation of game elements.

### 2.5.1 MDA Framework

A widely used framework in game design (and gamification) is the **MDA** framework [97], which is a conceptual model of game elements that separates them in three categories: mechanics, dynamics, and aesthetics. This framework breaks down the game design into three abstraction levels:

**Mechanics** are the functional components of the game, such as rules, challenge, environment, type of interactions, among others.

**Dynamics** consist of how the player interacts with the mechanics. They determine the actions of the player in response to the mechanics of the system.

**Aesthetics or Emotions** are the desired emotional outcome provoked in the player, when he interacts to the mechanics and dynamics of the system.

When building a game, the designer will follow a feature-driven approach (i.e., he defines the mechanics and then determines the dynamics, which lead to particular aesthetics). However, it is also important to consider that the player will see the game upside-down (experience-driven approach), in which case the emotions are key. Figure 2.4 presents the abstraction levels of the **MDA** framework from the point of view of both, the game designer and the player.

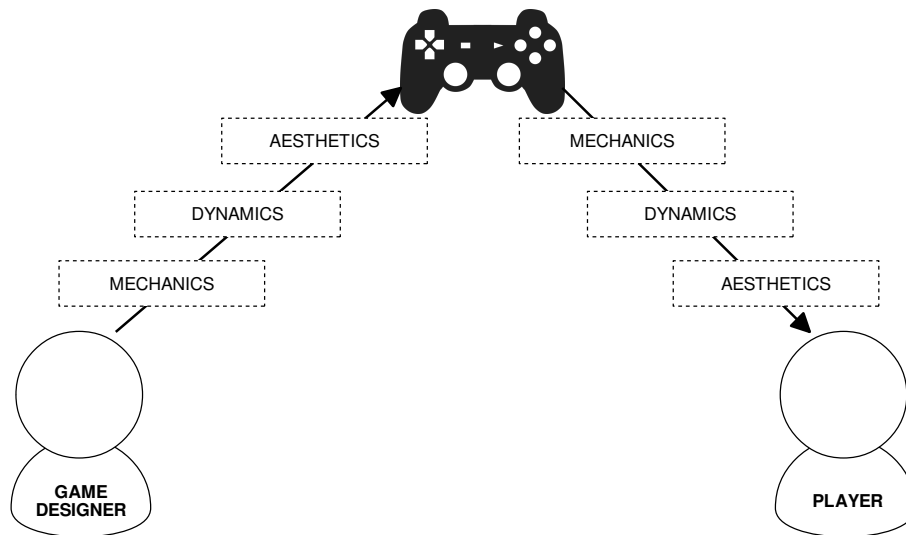


Figure 2.4: MDA framework: player and game designer perspectives of the game

The rest of this subsection provides more detail of each level, particularly focusing on the gamification of non-game systems.



## Mechanics

Mechanics consist of the decisions made by designers regarding the challenge, rules, environment, and types of interaction. These decisions are related to the game. Thus, all players will have the same mechanics during the entire game. In a game like checkers, these include the number of pieces, their initial distribution, how they move, how one player can “eat” pieces of the other, how to create queens, the configuration of the board, and how the outcome is decided.

The mechanics of a game can be divided into three categories [174]:

**Setup Mechanics** These mechanics are responsible for configuring the environment of the game, which objects will be used and how they are distributed by each player, how the game unfolds (e.g., turn-based), and what is the players’ structure (e.g., single or multiple players).

**Rule Mechanics** As the name implies, rule mechanics define the allowed actions (e.g., it is possible to move the piece diagonally) and the constraints that limit them (e.g., each play must take at most one minute), but also the challenge of the game (e.g., beat the opponent by “eating” all his pieces).

**Progression Mechanics** These mechanics are the extra motivators that push the player to complete the goal, such as rewards, points, badges, leaderboards, and quests. They are responsible for increasing the likelihood of repeating certain behaviors (e.g., when a goal is achieved and the player is rewarded, he will likely try to complete more goals). However, rewards should be significant within the context of a community, and distributed meaningfully (having too many rewards will reduce their impact).

From the gamification perspective, progression mechanics are the most widely used because they are the little push that many people need to get on the track of the task. Yet, to solve deeper problems in the activity to gamify, it may be necessary to immerse the player in a more game-like environment, thus recurring to setup (e.g., use a team-based player structure) and rule (e.g., timing constraints) mechanics.

## Dynamics

The dynamics consist of the player behavior, conditioned by the mechanics. They include strategic actions, in-game behavior, and interactions that happen during the game. A great example of a dynamic is bluffing at Poker. Bluffing is not a mechanic, but a behavior that emerges as the player participates in the game.

Decisions regarding mechanics can directly influence many other dynamics. For example, the players’ structure can favor dynamics like cooperation if it is team-based, or competitiveness if it is individual. However, some dynamics can be hard to predict. The friendly-fire, introduced

in some first-person shooter games to mimic reality, is one of such mechanics that can lead to unexpected behaviors. While it can promote a stronger cooperation between team players not to shoot each other, when points or badges are involved (e.g., the element which defuses a bomb gets a savior badge), competition may emerge, and players might “kill” team members to grab the reward themselves. Hence, game designers study all the possible outcomes and introduce methods to prevent negative behaviors (e.g., apply score penalties for shooting teammates).

## Aesthetics

The aesthetics are the desired emotional responses of a player while he interacts with the game. There are positive and negative aesthetics that can result from how an individual player follows the mechanics and then produces the dynamics. The next list describes eight positive aesthetics that game designers typically lookup, and provides an example of how they can be promoted in a gamified system:

**Sensation** – play to stimulate senses (e.g., inserting audio or visual elements).

**Fantasy** – play to enter into a make-believe world (e.g., consider a list of tasks as an interplanetary ride).

**Narrative** – play to experience drama (e.g., introducing schedules for tasks and hints).

**Challenge** – play to overcome difficulties (e.g., using leaderboards).

**Fellowship** – play to socialize (e.g., allowing discussion in forums).

**Discovery** – play to discover new things (e.g., promoting research inside/outside of the system).

**Expression** – play to discover personal strengths (e.g., awarding badges).

**Submission** – play as a hobby (e.g., timed tasks)

Sometimes it is easier to see aesthetics as a player (i.e., emotions). The most important goal of gamification is to create player enjoyment [184]. Fun (or enjoyment) comes from positive emotions, such as surprise, wonder, self-realization, excitement, and fascination. These type of feelings must emerge from the player, while negative emotions (e.g., disappointment, sadness or anger) should be reduced, otherwise, he will likely stop playing.

### 2.5.2 GaML

Gamification Modeling Language (**GaML**) [93] is a formal language that adheres to a context-free grammar for defining gamification concepts. It is based on the current opinions of gamification literature on game design elements and is consistent with earlier models and

taxonomies, particularly the five levels of gamification design [58]: Game Interface Patterns, Game Design Patterns, Game Design Principles, Game Models, and Game Design Methods. A key feature of **GaML** is its abstract syntax which can be understood and modified by non-domain experts. Furthermore, **GaML** has two concrete representations, namely a textual and a graphical representation, being both equally understandable and modifiable.

This language maintains the portability between different system architectures provided by the **MDA** framework but also enables to describe technical details of the gamified application by using text, allows the complete separation of the gamification design from the development, and ensures consistency in the development phase due to task separation. In addition to that, **GaML** guarantees that all **GaML**-based models conform to the language grammar and can then be automatically compiled to a specific technology without involving developers.

### 2.5.3 UML

The Unified Modeling Language (**UML**) is a general-purpose modeling and development language in the field of software engineering intended to standardize the design of systems. The language has two categories of diagrams: structural and behavioral diagrams. Structural diagrams emphasize the structure of a system, including the components, packages, and classes that it contains. This category encompasses a number of diagrams such as class diagrams, component diagrams, package diagrams, deployment diagrams, among others. Behavioral diagrams illustrate what must happen in the system being modeled (i.e., the functionality of the system). This includes activity diagrams, use case diagrams, sequence diagrams, among others.

Since object orientation provides a clear and formal technique for describing games and a formalistic approach to the actions taken in the game [179], it makes sense to consider an abstract modeling language, like **UML**, as a valid and beneficial way to describe games. In fact, **UML** has already been used to illustrate several aspects of games [23, 185], particularly activity diagrams for game execution flowcharts, sequence diagrams for the interaction of game objects, or use-case diagrams to show the set of actions triggered by a player action.

### 2.5.4 Petri Nets

Petri Nets are both a mathematical and a graphical modeling language for analyzing and designing complex, distributed, and concurrent systems initially invented for the purpose of describing chemical processes [142]. They have already been used in diverse areas such as the modeling of production lines, distributed database systems, communication networks, and business processes. Due to the possibility of representing Petri Nets as algebraic equations, they are also considered as a robust analysis tool which allows to check for the existence of starvation or deadlocks in multithreaded programs, measure systems' performance, and determine the precedence relation among events.

The graphical representation of Petri Nets consists of a diagram with circles (i.e., places) and squares (i.e., transitions) connected by arrows (i.e., arcs). The meaning of each symbol depends on what the designer is modeling. For instance, places can represent conditions, resources, or data, and transitions can mean tasks or events. There can be multiple arcs from a transition to a place and vice-versa, and also arcs going back to its input places (double arrow). A place may contain zero or more dots (i.e., tokens), which depending on the interpretation given to places may symbolize resources or the value of a condition. Typically, a transition can be activated when all its input places have at least one token, and this would consume one token from each input place and produce a token in all output places.

Much effort has been driven towards the improvement of the Petri Nets' capabilities and the simplification of its design and readability [203]. Meanwhile, they have started being applied in game design, particularly to plot descriptions and manage stories [28], but their capabilities are also being tested in modeling games fully [6]. Petri Nets can be advantageous due to their simple syntax (when comparing to UML) and ability to represent complex systems while preserving or reinforcing semantics of the diagram by using a hierarchy of diagrams. However, even with hierarchies, the growing complexity of the system can hinder readability.

### 2.5.5 Machinations Framework

Game mechanics are commonly obfuscated by the rest of the game's system and only revealed by detailed studies of the game structure. However, most of the models used to represent game mechanics (e.g., code representations, finite state diagrams or Petri Nets) are complex and inadequate for game designers. Furthermore, they do not provide a sufficient level of abstraction that immediately uncovers structural features such as feedback loops. The Machinations framework [61] is the formalization of the view of games as rule-based dynamic systems, focusing on the visualization of the structures in game mechanics that create emergent gameplay. To this end, it includes Machinations diagrams that are designed to illustrate the structural characteristics and dynamic behavior of the games in an understandable way to designers. Machinations are also a powerful analytical tool to compare and analyze games, particularly to detect recurrent patterns in the design of games.

The diagrams are designed to be simple enough to be drawn with any computer drawing tool or even on paper while keeping the description of the system accurate, showing clearly how different elements interact. This enabled the creation of a tool<sup>2</sup> to simulate, run, and interact directly with the Machinations diagrams (i.e., “play” the game through a Machinations diagram). The tool also allows to set up artificial agents to play the game rather than real players, allowing the simulation with a great number of sessions.

The Machinations framework defines a Domain Specific Language (DSL) to describe rules and mechanics, excluding level design elements (which can be represented by a similar DSL). Hence, it is best suited for most board, strategy, and economy simulation games that do not heavily rely

---

<sup>2</sup><https://www.machinations.io/>

on level design. In addition to that, the framework does not consider the characteristics of the player nor their cultural context, but rather sees games as complex state machines which can be in different states and whose transition to the next state is determined by the current state.

## 2.6 Game-based Learning

One of the major concerns amongst researchers in education is the disengagement of the students in the learning activities, which frequently leads students to dropout of learning institutions. Nevertheless, even students who do not dropout tend to have high rates of boredom and loss of interest with learning [119], thus reducing their attainment [190].

On the other side, when people are highly motivated in a task, they can reach a (flow) state of intense concentration and complete absorption on it [51]. To achieve this state, it is necessary an optimal experience (see Figure 2.5), i.e., the task must provide:

- Clear sense of the rules and next goals;
- Immediate feedback or rewards for achieved goals;
- A challenge adjusted over the average skills of the person, which requires effort and allows him to perfect his skills, while gives him a chance of completing it;
- A sense of control over own actions.

If a person experiences these elements, he is likely to feel a sense of deep enjoyment that worths expending energy. So, this happens very often with games since most of them incorporate all those elements, as described in previous sections. Although, whether or not an individual achieves flow state depends on the skills of the player and the skills required by the game he is playing. That is one of the reasons why most games try to adjust the challenge to the averaged skill players while providing means to adjust difficulty.

In respect to education, the flow has been linked to foster learning and academic achievement [52]. These facts have led some researchers to question themselves whether games could support learning, rather than shutting the door of the education against them. On this branch, studies predict that gamification is increasingly becoming a common method to invoke engagement and flow in students [50].

Furthermore, many studies have focused on using gamification to motivate students [32, 72, 189]. Meanwhile, some claim that succeeded to induce flow in students [86, 87, 180], and to demonstrate its relationship with learning outcomes [15, 41, 96, 109]. Although, some research found contradictory results, i.e., no significant connection could be established between the level of motivation and better learning results [1, 29].

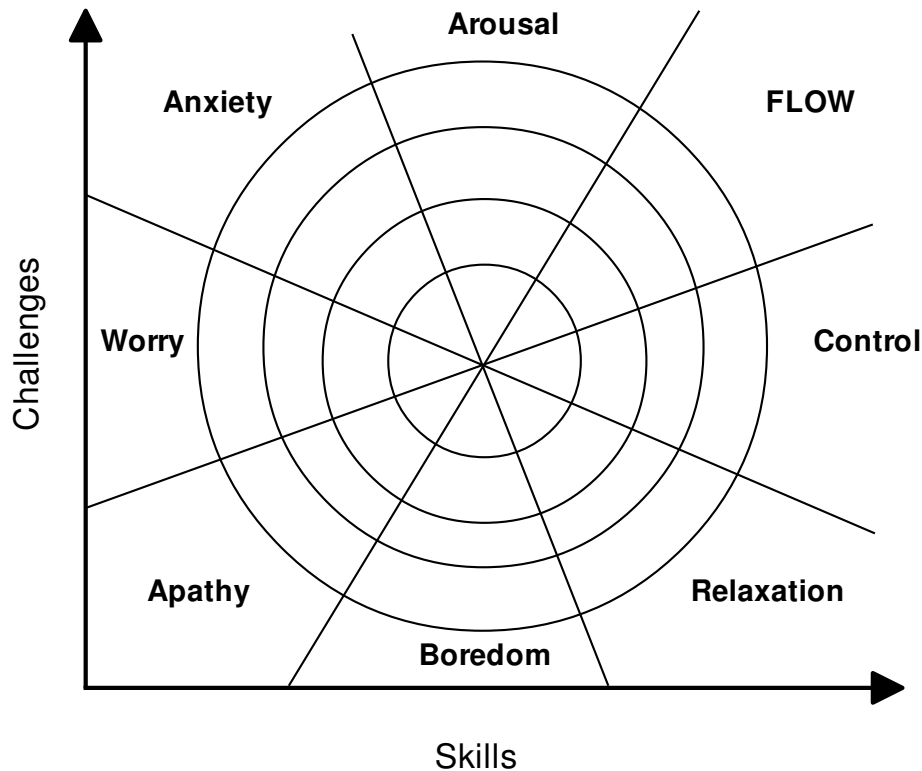


Figure 2.5: The current model of the flow state [146]

As with entertainment games, gamified learning activities (or game-based learning) need to propose challenges to the students in a way that they need to apply a great amount of effort to overcome them, and still have a chance to do so. In this way, if clear goals and rules, immediate feedback, and freedom of actions are also provided to the student, then all requirements to achieve flow are filled. Thus, ideally, students will try to overcome the challenges, not because they expect some reward, but because they are worthwhile [109].

However, creating challenges adapted to the students' skills is often tricky, especially in heterogeneous environments like schools. So, most gamified systems rely on extrinsic rewards, such as points, badges, and progress bars, which can constitute the little push needed by many students to perform their activities [138], particularly those who seek status, recognition, and prestige [172]. Although, as stated in Section 2.5.1, these rewards should be distributed meaningfully to have some positive effect and to not harm motivation that is intrinsic to the student.

Nevertheless, gamification of education has been a top trending in the last years due to its accredited positive boost in motivation. There are several case studies in the literature, particularly performed in high school and universities, distributed among different learning subjects, ranging from Computer Science/Information Technology [5, 10, 21, 85, 156, 186, 191] to Mathematics [44, 68, 160, 200], Foreign Languages [91, 162], Communication and Multimedia [12, 89, 94, 104], Psychology [118], and Medicine/Biology [24, 148, 163]. Appendix A presents a comparative study of the aforementioned research papers, which aims to explore what has been

done and to understand the respective impact in learning.

## 2.7 Competition-based Learning

Games can be challenging in different ways. Some games challenge the player through successive sets of goals; others promote competition amongst all participants. The latter subset of games generally attempts to artificially create a scenario in which there is equality of winning chances to all players, in order to give precise and incontestable value to winner [36]. Therefore, the qualities of the participants (e.g., speed, memory, strength, or cleverness) would decide who is better. In the field of education, competition is a very important and contested game element, since there is no agreement on whether it produces beneficial [34, 35, 171], insignificant [192, 201], or harmful [198] effects in students' learning.

The main reason why several authors consider that competition produces adverse effects is based on the negative impact of losing in students' motivation [193]. Although, the impact of competition in learning may depend both, on the type of competition and on how its results are used.

Competition comes in various forms. Some games go straight to the core of competition, i.e., one player (or team) plays against the other, if he wins, then the other loses. In other games, competition is based on leaderboards. Most of the game-based learning environments focus on the latter [35, 44, 144, 192], but there is also some work on the former [34, 171], mainly in computer programming learning.

Some teachers use the result of these competitions to assign grades to their students (competitive-based learning). In this case, students need to outperform the other students to achieve an higher grade. Even though this approach may foster a strong competition between students, there is a punishment to those who lose which, typically, harms their motivation. To diminish this issue, competition is generally only used to support learning (competition-based learning). Thus, the learning result is almost or completely independent (may count as bonus point) of the competition, allowing the students to enjoy the learning process [34, 177].





## Chapter 3

# State of the Art

Learning and teaching to program is a hard task [105, 173]. If students keep facing difficulties for too long they are likely to give up, which generally results in high dropout and failure rates in programming courses [19]. This is a great concern for educators in this area and has led to a large body of research, not only to seek new forms of introducing students to programming, but also to boost the interest and motivation of people in coding.

Among the most successful approaches are the use of competition and other game-like features in web learning environments to stimulate learners in solving more programming activities. This dissertation also aims to apply characteristics of games, particularly the assessment of programming exercises following competition formats and game-like graphical feedback of the behavior of the code, to create a more engaging programming learning environment.

To the best of author's knowledge, there are already learning environments that provide some of these features, but none of them includes automatic and competition-based assessment, game-like graphical feedback, and gamified educational content presentation, while supporting educators themselves to create their own courses with activities leveraging all these features. Thus, this chapter reviews platforms and systems which contain some characteristics common to the current work. Unlike the study conducted in Section 2.6 and described in Appendix A which focuses on the literature of empirical game-based learning systems tested in general academic contexts, this chapter gives emphasis on programming learning environments available on the web, that may or may not be described in the literature.

This chapter is divided as follows. Section 3.1 describes some systems that support open online programming courses either without gamification or using only game elements that foster extrinsic motivation. Section 3.2 presents programming learning platforms which stand out for their competitive side. Section 3.3 addresses platforms and games in which learners' code takes action in a game world.

### 3.1 Platforms for Open Online Courses of Programming

Open online courses of programming are easily found on the web, since there are dozens of platforms that support them. Some platforms, such as Coursera<sup>1</sup>, Udacity<sup>2</sup> and edX<sup>3</sup>, offer a variety of learning material from top universities' courses, with the possibility, typically at a monetary cost, of getting a verified certificate. Other platforms have less formal courses with a wide range of activities to provide hands-on experience on a specific language or framework (e.g., Codecademy<sup>4</sup>, Khan Academy<sup>5</sup>, Free Code Camp<sup>6</sup>).

The former platforms are designed for people who already have some previous background in the subjects. They aim to improve the knowledge of people in areas that they are already interested in (intrinsic motivation). These platforms are only a means for people to have access to materials that were otherwise difficult to get. So, there is a tiny or no gamification factor in them. Also, creating courses for platforms like Coursera is an arduous task, requiring the agreement and involvement of third parties (e.g., universities).

In less formal open online courses' providers, the use of gamification is a common technique to attract learners. The elements of gamification that are generally used are progress indicators, badges, levels, leaderboards, and experience points. For example, Khan Academy uses badges and progress tracking to engage students to enlist and complete courses. Even that it also allows educators to easily create courses for their learners, it is limited to JavaScript-based programming activities (HTML and CSS are also supported).

Codecademy also uses progress indicators and badges. It supports programming languages such as Python, JavaScript, Java, SQL, Bash/Shell, and Ruby. However, creating courses on Codecademy is not yet possible.

In the context of educational institutions, the most widely used approach to create gamified open online courses is to rely on an Learning Management System (**LMS**) with game mechanics. **LMSs** were created to deliver course contents, and collect assignments of the students. However, many of them evolved to provide more engaging environments resorting to gamification. Some of the most notable examples are Academy LMS, Moodle, and Matrix which include badges, achievements, points, and leaderboards. Moodle also has several plugins which offer a variety of other gamification elements.

Another tool that serves this purpose is Peer 2 Peer University (P2PU) [3]. P2PU is a social computing platform that fosters peer-created and peer-led online learning environments. It allows students to join, complete, and quit challenges, and to enroll in courses. By completing learning tasks, courses, or challenges, they can earn badges, which are based on Mozilla Open Badges

---

<sup>1</sup>[www.coursera.org](http://www.coursera.org)

<sup>2</sup>[www.udacity.com](http://www.udacity.com)

<sup>3</sup>[www.edx.org](http://www.edx.org)

<sup>4</sup>[www.codecademy.com](http://www.codecademy.com)

<sup>5</sup>[www.khanacademy.org](http://www.khanacademy.org)

<sup>6</sup>[www.freecodecamp.org](http://www.freecodecamp.org)

framework<sup>7</sup>.

However, LMSs and P2PU do not have out-of-the-box support for automatic assessment of programming exercises, which is a somewhat crucial feature for any programming learning environment. Thus, any of the three types of open online courses' providers described above have its strengths, but lack the ability to easily create courses, a strategy to engage learners, the possibility to present different forms of learning materials, or the support for a multitude of programming languages. The same happens with other tools described in the literature [79, 99].

From the research conducted, Enki [156] is the only tool that blends gamification, assessment and learning, presenting content in various forms as well as delivering programming assignments with automatic feedback, while allowing any stakeholder to create courses freely. A brief summary of the reviewed platforms and systems for open online programming courses is presented in Chapter B.

## 3.2 Competition-based Programming Learning

Humans are competitive by nature, and while learning activities focusing too much on competition can produce adverse effects [110], competition has proven to be an effective technique to stimulate students to go beyond their capabilities [42].

The use of competition for learning programming is also not new [34, 121, 125]. For instance, programming contests are becoming more and more popular among learners [187]. This is good for students, who are increasingly facing programming contests after leaving universities, as a part of the recruitment process for top technology companies.

This section reviews some platforms and tools that use competition as a way to engage students in learning programming.

### 3.2.1 Automatic Judge Systems

Automatic judge systems are responsible for marking (i.e., assigning a grade to) programming assignments, by comparing the obtained output with the expected output [77]. These systems made it possible to realize programming competitions in a really competitive and engaging way. They have already been used, for more than fifteen years now, to manage programming contests all over the world, ensuring impartial and accurate timely feedback on program evaluation.

Programming contests are, by their definition, competitive activities where there are winners and losers. They are voluntary activities where individuals or teams strive to find solutions to previously unseen problems. Participants generally have a preparation phase where they learn several interesting strategies to understand and solve problems faster. Also, as contests

---

<sup>7</sup>[www.openbadges.org](http://www.openbadges.org)

are managed by automatic judges, participants can make as many submissions as they want during competition, and learn from their mistakes. So, losers actually win knowledge which may compensate the negative effects typically associated, by many people, with competitive learning activities [170].

There are many automatic judges available in these days, such as DOMJudge [65], PC<sup>2</sup> [8], PKU JudgeOnline [161], and Mooshak [124]. Most of them born to support programming contests. However, the ability of these tools to support learning in programming classes was explored soon [125]. They are capable of providing immediate feedback to students inside and outside the classroom as well as stimulating, through timed challenges and leaderboards, their effort and persistence in solving activities [83].

### 3.2.2 HackerRank

HackerRank<sup>8</sup> is a platform dedicated to hosting competitive programming challenges. This platform allows the user to code their solutions in more than 30 programming languages, test their code with custom input, and discuss problems with other peers. It has a practice mode where challenges are grouped by domains (e.g., Java, Javascript, Artificial Intelligence, among others), and a contest mode where challenges can be created and sponsored by any organization.

In the practice mode, there are leaderboards, badges, and points. Each domain has a leaderboard and a badge. A badge carries either one, two, three or four stars depending on the total points earned by a user. Typically, challenges proposed in contests are added to a practice domain, once the contest ends, according to the subjects that they cover.

The contest mode has various forms. There are contests that are opened indefinitely, in which challenges are proposed every week/day/month, and there are contests that run for a limited amount of time (e.g., 48 hours). Contests have their own leaderboards and badges, that take into account the time spent by the users, since the beginning of the contest, to solve the problem. There is also a particular type of challenge where Software Agents (SAs) developed by users run against each others, and the score is assigned based on the result (win, draw, or loss).

Contests also enable users to increase or decrease their rating, earn medals (which are given to top solutions), and possibly win cash prizes or jobs at top technology companies.

### 3.2.3 CodeFights

CodeFights is a platform where users can “fight” against other players or bots developed by companies. A fight consists of two players trying to solve three coding challenges before each other. Typically, in the first challenge the player must fix a bug in a piece of code. The second challenge requires the player to fill a gap in a function. Finally, the last challenge consists in solving a complete puzzle.

---

<sup>8</sup><https://www.hackerrank.com/>

Winning a fight enables the players to collect badges, to earn experience points, to increase its level, and to receive coins. Coins can buy clues to solve puzzles, while the other elements serve mainly to gain status.

Behind fights, players can also practice their programming skills with puzzles of technical interviews from several top companies, solve daily and weekly coding challenges, play the arcade mode (i.e., solve groups of similar puzzles), or participate in tournaments created by themselves or other players.

The platform supports more than 35 programming languages. The users can use the built-in editor to write their solutions, and test them using public tests or user-defined tests (textual input and output). The interface to develop a solution is presented in Figure 3.1.

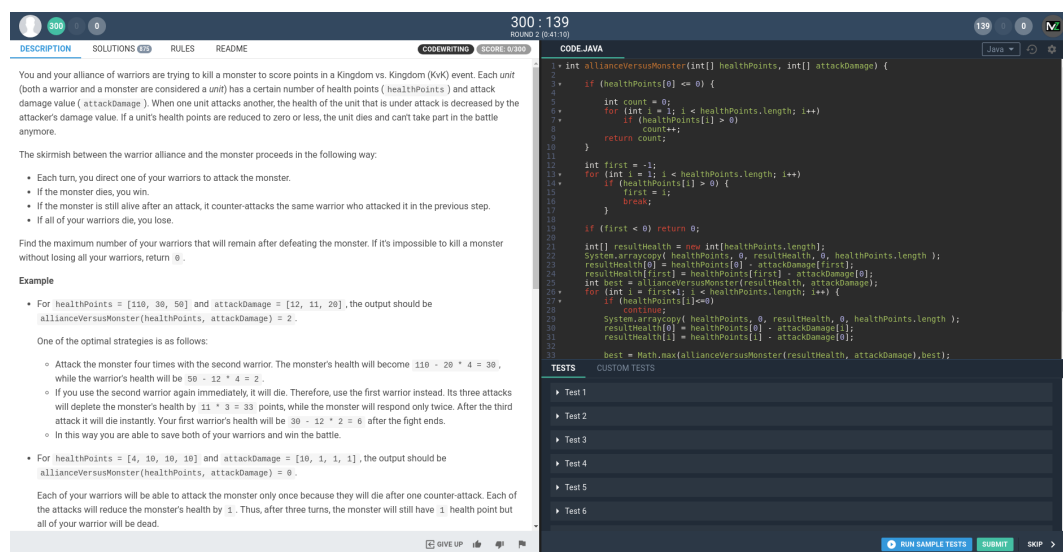


Figure 3.1: CodeFights: second challenge of a fight against a company's bot

### 3.3 Game-based Programming Learning

Games are usually effective motivators, as explained in Chapter 2. However, gamified environments are generally very different from games, and fail to use elements that are more prone to motivate users intrinsically such as the graphical feedback and in-game challenges.

There are some exceptions in gamification where the learning is an outcome of playing a real game (formally known as serious games). For instance, in the context of programming learning, it is common to have games where the typical input controls have been replaced by code.

This section reviews both platforms where the learning activities are game-based programming challenges with graphical feedback, focusing on their support for authoring new challenges, and games that teach programming concepts.

### 3.3.1 CodinGame

CodinGame<sup>9</sup> is a web-based platform that proposes several puzzles for learners to practice their coding skills. Most of these puzzles require the user to develop an **SA** to control the behavior of a character in a game environment, and provide a 2D game-like graphical feedback. Figure 3.2 shows one of these puzzles, where the user has the statement of the exercise on the left as well as the movie player, and the code editor and test cases on the right.

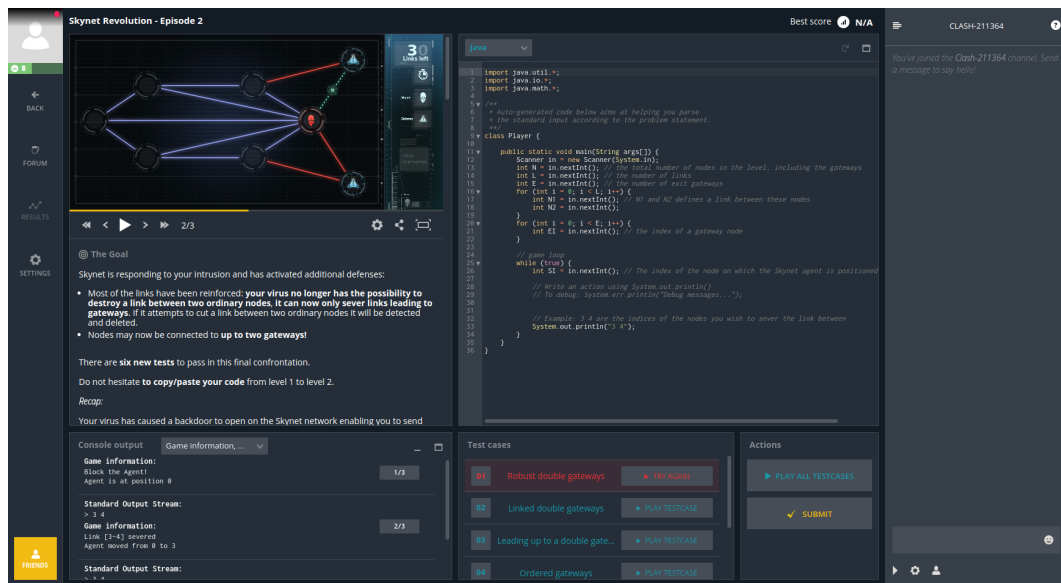


Figure 3.2: CodinGame: solving a puzzle which provides 2D game-like graphical feedback

The **SA** programmed by the player must pass all test cases (public and hidden) to solve the puzzle. Players can choose one of the more than twenty programming languages available to write their **SA**, or even solve in more than one language. Once the exercise is solved, players can access, rate, and vote on the best solutions.

Solving these game puzzles and awarding votes on solutions contributes to the leaderboards, level, and badges of the user. There are also contests from time to time in which the player can receive real-world rewards, by defeating other player bots or being the first to solve all problems.

Another mode that CodingGame provides is the *Clash of Code*, where a player competes against other players to be the first to submit the best solution to an exercise. These exercises typically last for five minutes and can be authored by the community, although they do not have a game-like graphical feedback in these cases.

### 3.3.2 Greenfoot

Greenfoot [112] is an educational Integrated Development Environment (IDE) aimed at teaching object-oriented programming in Java or Stride to 14+ years old students. It supports the

<sup>9</sup>[www.codingame.com](http://www.codingame.com)

development of applications that use 2D animated graphics such as games and simulations by encapsulating the actual graphics code so that programmers can concentrate entirely on programming object behavior.

The purpose of Greenfoot is twofold. On the one hand, it aims to be a pedagogical integrated development environment for novice programmers to create applications with immediate 2D visual feedback, integrating the common tools of an IDE (e.g., editor, compiler, and virtual machine) and educational tools (e.g., interactive object invocation, inspection, and class visualization). On the other hand, it intends to provide a simulation and micro-world framework that offers an higher degree of interaction than existing micro-worlds such as turtle graphics or Gridworld.

To this end, the Graphical User Interface (GUI) of Greenfoot, presented in Figure 3.3, is composed of two windows: the interactive viewer (on the left) and the code editor (on the right). The interactive viewer window contains the Greenfoot world (i.e., the area where the program executes), which is variable in size and holds the scenario's objects, and a class diagram to visualize the classes used in this scenario and their inheritance relations.

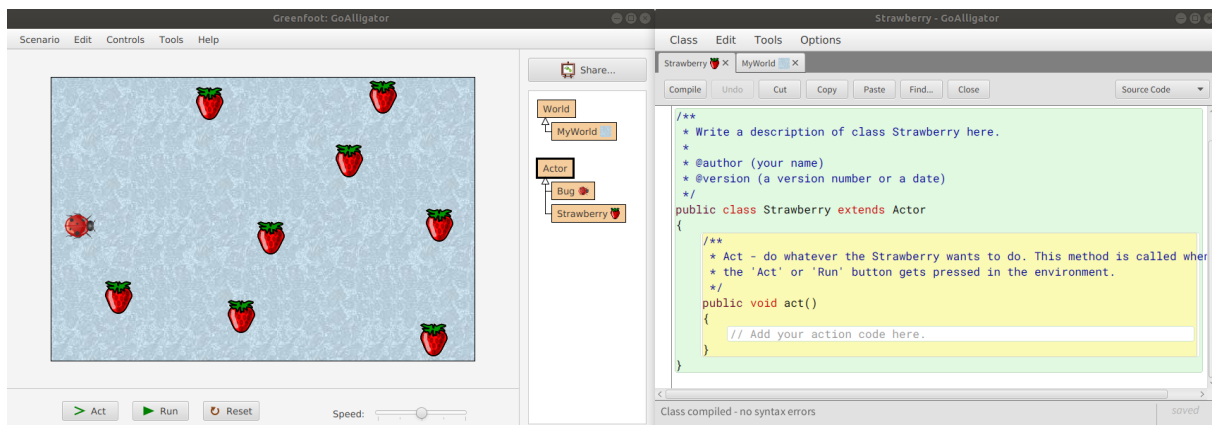


Figure 3.3: Interface of Greenfoot: the interactive viewer on the left and the code editor on the right

### 3.3.3 SoGaCo

SoGaCo (Social Gaming and Coding) [59] is a scalable cloud-based web environment that evaluates competitive SAs, developed either in Java or Python. The games of SoGaCo are mainly board games for two players, which offer an interactive GUI that allows learners to see step-by-step how their programs play the game.

The interface of SoGaCo has two distinct views, one for editing code and another to test the bots. The editor view consists of a simplified IDE with undo/redo operations, buttons to add and remove bots, a button to save and compile the code, and some styling configurations for the editor. On the left side, there is a list with all bots developed by the current user, from which he/she can select one to use or edit. The test view, presented in Figure 3.4, has three panels on the left containing the user bots, built-in bots, and shared bots from other users, one panel on



the center which displays the graphical feedback, and three panels on the right to control the display of graphical feedback, display the score and show captions of the game.

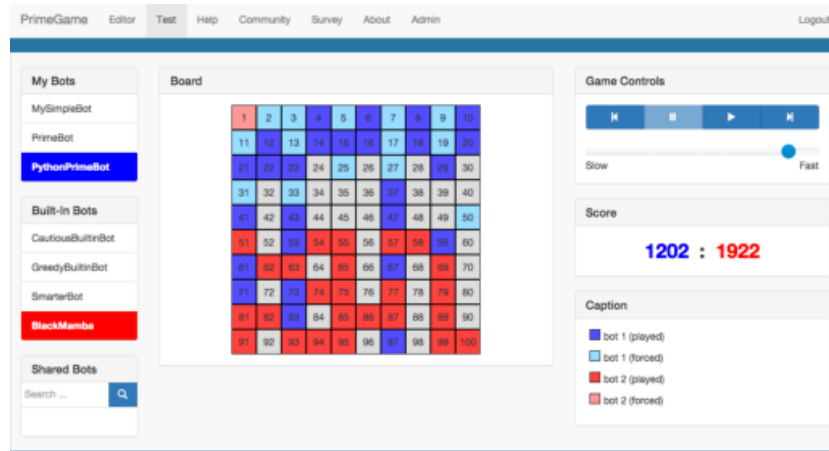


Figure 3.4: SoGaCo: test view of PrimeGame

The user starts coding the SA from a skeleton provided by the game author. During the development, he/she can test the bot against any bot present in one of the three bots' panels. After completing it, the single bot address (URL) can be shared with other peers to either compete against it or see its code.

The modular architecture of SoGaCo supports different games, but there is no known framework or standard form to develop games for SoGaCo. Nevertheless, it already contains several board games, such as PrimeGame, Mancala, Othello, and 5 in a Row.

### 3.3.4 Games for Teaching Programming Concepts

This subsection presents a series of games that have been developed specifically for teaching some concepts of programming. A large set of games was collected, of which those games that address both the cognitive and emotional side of the students were selected. In these games, the player starts by processing small pieces of information, and climbs the Bloom taxonomy hierarchy [22], until they are capable of evaluating their progress and create solutions to similar problems on their own. The motivation to keep evolving is based on the graphical feedback, well-defined challenges, competition, and other game elements.

A total of 14 games have been chosen and described. Each of the next sub-subsections describes a game, including which concepts it aims to teach, how, and its key features. A summary table can be found in sub-subsection 3.3.4.15.

#### 3.3.4.1 Catacombs

The Catacombs [13] is a three-dimensional role-playing game, in which the player is an apprentice wizard who has the task of casting increasingly complex spells to save two children who are



trapped within the catacombs. Spells are programs in a micro-language which the player must construct in the game, using an interface based either in dialog trees (Spellbook version) or matching (Gemstone version). In the Spellbook version, the player must construct the spells interactively by correctly answering questions posed by the spellbook. In the Gemstone version, the player has either to choose the correct spell among many wrong ones or fill-in blanks in spells that have been partially cut off. Figure 3.5 presents an example of the graphical feedback displayed when the wizard casts a spell.



Figure 3.5: Graphical feedback of a player casting a spell in Catacombs

The aim of Catacombs is to teach players how to declare variables, use simple and nested conditional statements, and loops. The answers to the questions automatically create lines of code in a micro-language, which are displayed to the user. If the answers are correct, the wizard progresses to the next level; otherwise he/she receives feedback about what is wrong in the answer and is prompted to try again.

#### 3.3.4.2 Saving Sera

Saving Sera [13] is a 2D exploratory role-playing game, where the player must rescue Princess Sera who was kidnapped by a monster. The player must complete a series of quests in order to progress in the game. The quests include fixing a “machina” (which is a real program in a micro-language) by interpreting and finding the bugs in a while loop that helps a fisherman count the fish he has caught and decide when to stop; debugging a nested for loop to place eggs in crates (see Figure 3.6); filling in the blanks of a “machina” with variables to make a small flyer using print statements; and organizing a flowchart through a drag-and-drop interface to create a quick-sort implementation. When the player makes a mistake, he/she needs to fight a script bug by answering a number of multiple-choice computer science questions.

Advanced players are able to create their own “machinas” in the game. Furthermore, new quests can also be developed using RPGMaker, which allows to develop games using a relatively simple drag-and-drop interface to create a tile-based map and dialog trees for characters, only with arrow key and space bar interactions.

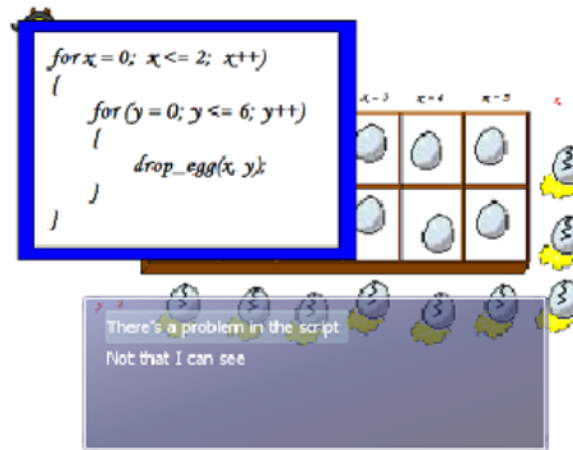


Figure 3.6: Example quest of the game Saving Sera

### 3.3.4.3 EleMental

EleMental [40] is a 3D game that aims to teach players how to use recursion and perform depth-first search (DFS) traversal using C# programming language. The goal of the game is to complete three programming puzzles with the support of a programmable avatar – Ele –, an avatar for visualizing data to collect – Thought –, and an in-game mentor – Cera –, who provides instructions as the player progresses.

The game starts with a small pretest to assess the knowledge of the player about recursion. Then, the player is asked to create a simple “Hello World” program to get used to the interface of the compiler. After these initial steps, players must walk Ele using a DFS traversal through the binary tree to collect Thoughts from the leaves. The second level, presented in Figure 3.7, requires learners to code the traversal for the left side of DFS, which Ele will execute while Cera explains what the code is doing. In level 3, they must code both sides of DFS and drive Ele through the binary tree using the keyboard and mouse, while visualizing the stack calls made by the recursive algorithm. Once the player finishes the game, they take a final survey to complete the game.

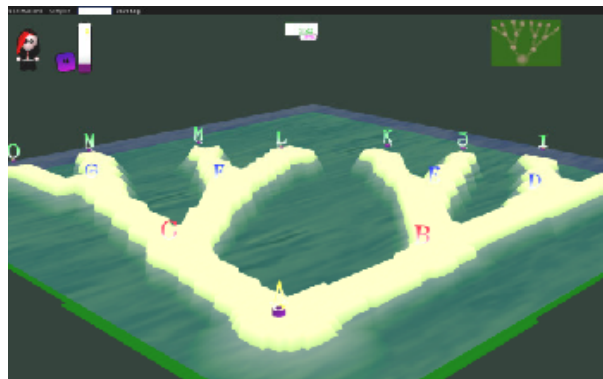


Figure 3.7: Screenshot of level 2 of EleMental

#### 3.3.4.4 Prog & Play

Prog & Play [143] is an API built on top of Kernel Panic, a 3D open source web-based real-time strategy game, to allow the player to give instructions through a program, rather than clicking on a map with the mouse. Kernel Panic takes place inside a computer, where the player controls one of three parties: systems, hackers, and networks. Each of these parties can use three different kinds of objects. For instance, systems have bits, bytes, and assemblers, while the hackers can use virus, bugs, and worms, and networks have ports, firewalls, and packets. These objects can generate a few other kinds of units. Figure 3.8 presents a screenshot from the game.

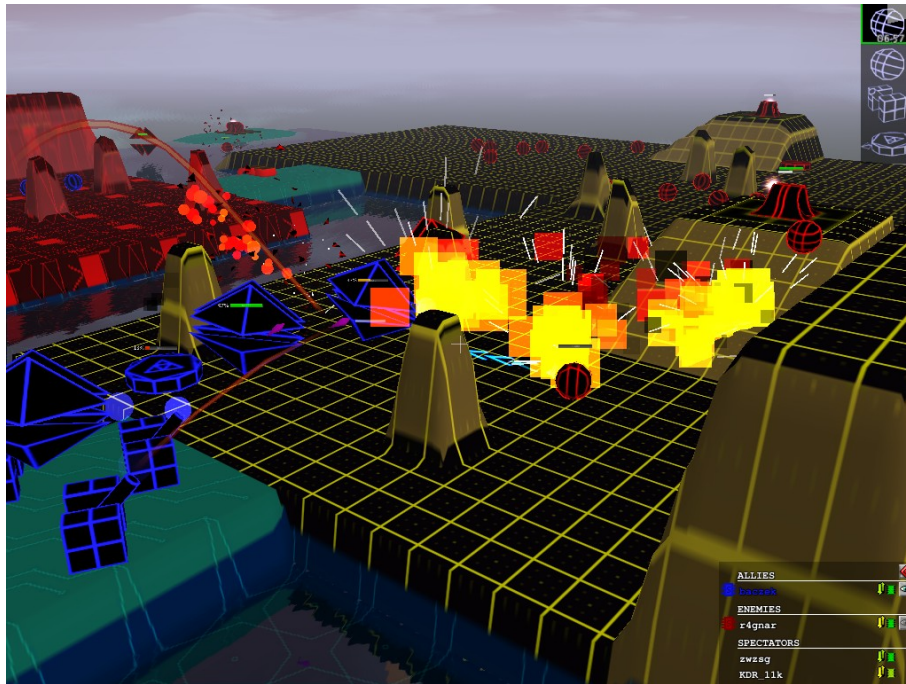


Figure 3.8: Screenshot of Kernel Panic

Kernel Panic is a simplified real-time strategy game in that the only resources which are managed are the time and space and, thus, all objects are free to create. Due to these characteristics, the differences between players are only based on the strategies and tactics used. For example, players are able to form alliances with each other to prevail in the game.

The Prog & Play API is available in several programming languages such as Ada, C, Java, OCaml, Scratch and Compalgo. Hence, it can be used in subjects requiring imperative, object-oriented, functional, or graphical teaching approaches.

#### 3.3.4.5 Wu's Castle

Wu's Castle [62] is a two-dimensional role-playing game, which aims to teach loops and arrays in an interactive and visual way. The player solves in-game challenges by interactively constructing C++ code, while watching the graphical feedback to check how the program behaves and identify

logic issues. Figure 3.9 shows a for loop to build snow men partially executed (i.e., half of the snow men are already built).



Figure 3.9: Screenshot of Wu's Castle

The game supports two types of interaction. The first consists of manipulating arrays by tuning the parameters of a for loop. The second is by navigating the avatar through the execution of nested loops. After starting the game, the player is firstly provided with a brief introduction to the background story and interface of the game. In the first level, he/she has to explore the manipulation of one-dimensional arrays using for loops. The second level requires the player to control the character through a nested loop, represented graphically as maze. While walking the character, the player is asked multiple-choice questions about the loop's code. Finally, the third level requires the modification of two-dimensional arrays using nested loops.

#### 3.3.4.6 RoboZZle

RoboZZle [154] is a 2D web-based puzzle game, in which the player has to navigate a robot through a colored tile-based map to catch all stars. The game provides a series of pre-defined commands accessible through a panel on the left side of the screen, which the user can move to the execution stack of the robot. These commands include instructions to move forward, turn left, and turn right, calls to another function, and color-based conditionals that can be combined with the previous commands.

The interface of the game, presented in Figure 3.10, contains four different panels. The top-left (and bigger) panel displays how the robot moves across the virtual world, according to the given commands. In this way, the player can detect where the robot does mistakes and correct accordingly. The bottom panel presents the stack of commands to execute, where the left-most command is the one being executed. The center panel contains a number of functions, initially empty, each with a variable length, which the player can fill with the commands to execute. The quantity of functions and their length is set by the author of the puzzle. This panel also contains inputs to control the visual feedback (e.g., speed, step-by-step, etc). Finally, the right panel

contains the commands that can be used.

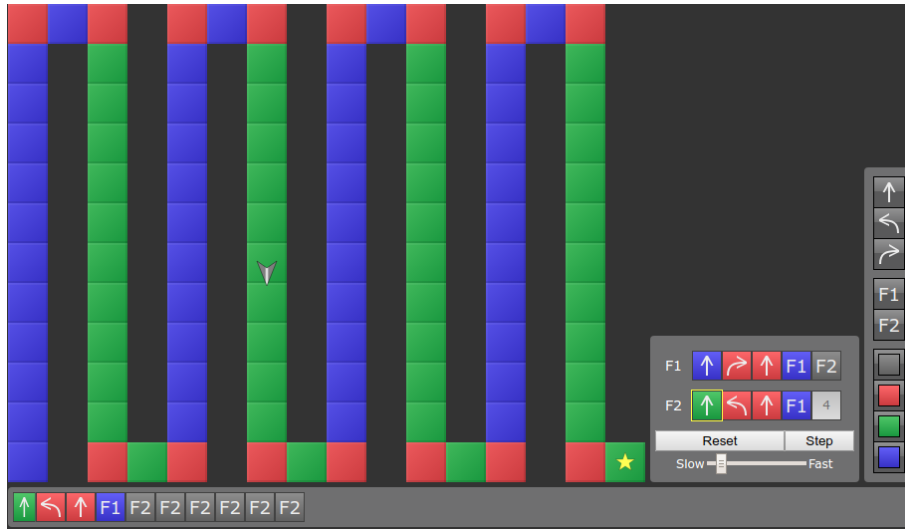


Figure 3.10: Screenshot of a custom puzzle of RoboZZle built by another player

The pedagogical aim of RoboZZle is to teach how to use conditionals, function calls, and recursion without showing any actual code. Furthermore, it allows any user to add new puzzles and share them with the community.

#### 3.3.4.7 LightBot

LightBot [2] is a 3D web-based (also available on mobile) puzzle game inspired by RoboZZle, in which the player navigates a robot through a tile-based virtual world to light up certain tiles. The way of controlling the robot is very similar to that of RoboZZle, but with additional commands to jump and light up tiles.

The interface of LightBot, presented in Figure 3.11, also mimics that of RoboZZle, but does not contain a panel with the current execution stack. In addition to this, the number of functions available to use is always fixed (i.e., a main method and two auxiliary functions) as well as their length (12 instructions for the main method and 8 for each auxiliary function).

The first version of LightBot is composed of 12 levels, whereas version 2 comes with 20 levels including eight levels of basic instructions, 6 of procedure calls, and 6 about loops/recursion. Moreover, it has a level editor where players can create new puzzles and share them with the community.

#### 3.3.4.8 TALENT

TALENT (Teaching ALgorithms EnvironmeNT) is a 2D web-based multiplayer adventure game, in which the user plays the role of an archaeologist, who collects objects that are available at

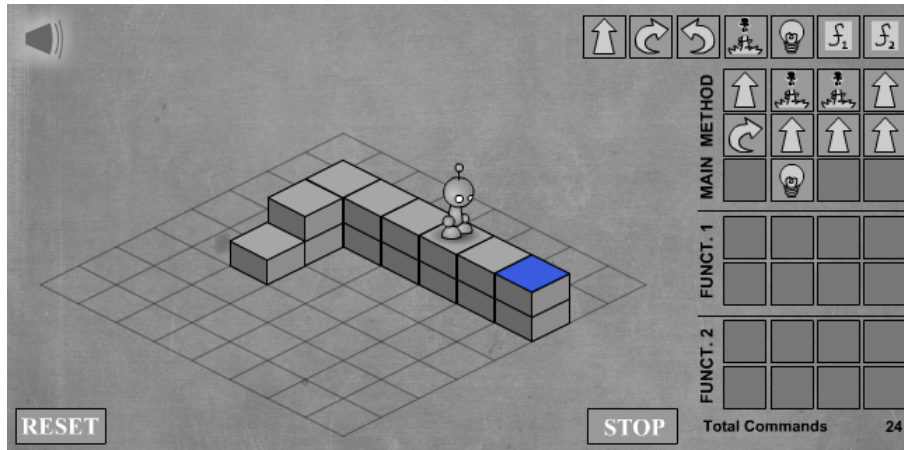


Figure 3.11: Screenshot of a puzzle of LightBot version 1

specific locations of the virtual world for their future exhibition at a museum. To this end, the player uses a micro-language to operate a robot vehicle that will collect the items in its turn.

The game supports all programming structures including conditionals, switches, and loops. The student can construct the programs in the environment with the support of a programming tool, presented in Figure 3.12, which combines a drag-and-drop program editor and a step-by-step interpreter that displays the effects of each instruction in the virtual world.

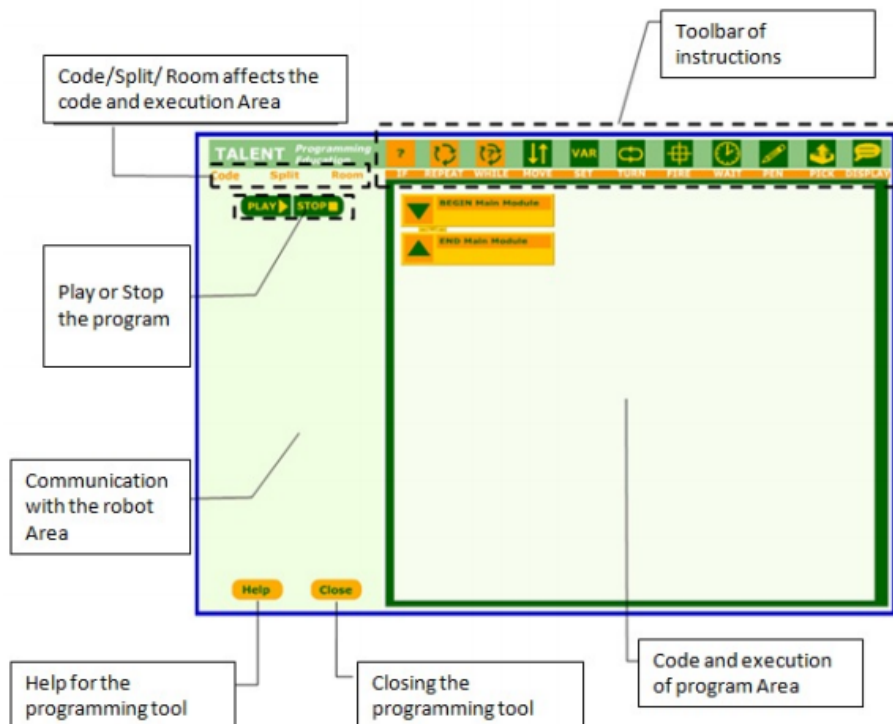


Figure 3.12: Interface of TALENT programming tool

The game provides an agent that guides the player through the missions and helps him/her when necessary.



### 3.3.4.9 RoboCode

RoboCode [90] is a 2D multiplayer game, developed and released by IBM in 2001, in which users create **SAs** for robot-tanks using Java programming language and an API provided by the game. These robot-tanks are placed in a small rectangular battlefield together with other players' robot-tanks, as shown in Figure 3.13. The goal of the game is to be the last robot-tank standing on the battlefield, by destroying all the others, and preventing itself of getting shattered.



Figure 3.13: Screenshot of a battle in RoboCode

The skeleton of the robot-tank is given in the Robot class, and several sample **SAs** that perform specific tasks are also provided. Students can test their code against these bots, and/or combine their code to create a more intelligent bot. The code must be written on an **IDE** chosen by the student, and it is tested using an **IDE**-specific plugin or using a tool provided by the game developers. The feedback of each battle is provided as a 2D frame-by-frame movie.

Typically, competitions are organized with two (head to head) or more robot-tanks per battle. The losers of each battle are progressively sent-off until one robot-tank remains (the winner).

### 3.3.4.10 Code Rally

Code Rally [202] is a 2D car racing game developed by IBM and presented at the 2003 ACM International Collegiate Programming Contest (ICPC) World Finals. The player controls a vehicle either with a Java or Node.js **SA** to race around various virtual tracks against opponents' **SAs**, as shown in Figure 3.14. The only goal is to win the race.

Players can make decisions about when to speed up, turn left/right, slow down, and hit other cars based on the location of other players or checkpoints, obstacles on the roads, their current



Figure 3.14: Screenshot of a race in Code Rally

fuel level, among others. Before submitting the SA to the server, players can test their car locally against a number of sample rally cars.

The game provides users with an API to operate the vehicles with an higher level of abstraction as well as an API to attach handlers to race events (e.g., `onCarCollision(otherCar: Car)`, `onObstacleInProximity(obstacle: Obstacle)`, etc).

#### 3.3.4.11 MUPPETS

The MUPPETS (Multi-User Programming Pedagogy for Enhancing Traditional Study) [164] is a 3D desktop collaborative virtual environment that aims to teach object construction and other basic concepts of object-oriented programming using Java language. Students are required to write and manipulate objects to be able to use them later in fights against opponents' objects, in a virtual arena.

The goals of the MUPPETS system are twofold. On the one hand, it provides a complex, interactive, collaborative playground in which novice programming students can learn the principles of object-oriented programming, with emphasis upon encapsulation, inheritance, and polymorphism, by creating their objects. These objects can be shared both with peers and more experienced students. On the other hand, the system provides a mechanism through which more experienced students can contribute to the success of beginner students, through the development of extensions as well as complex artifacts intended for use within introductory courses.



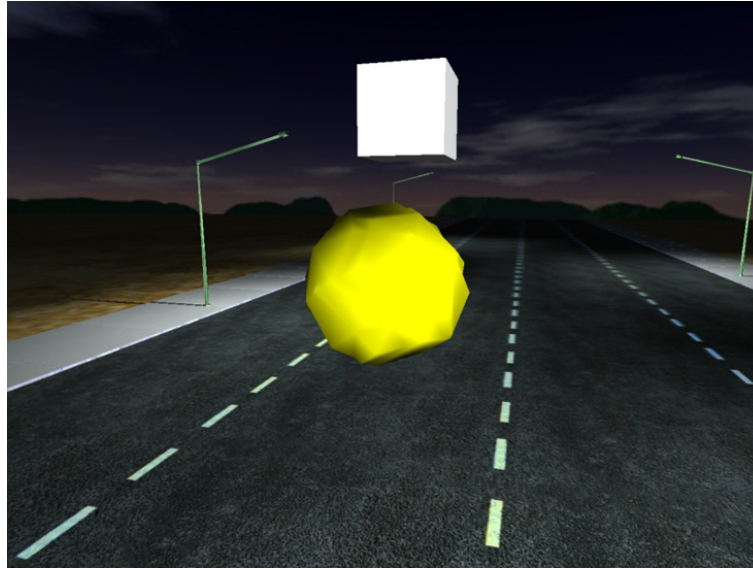


Figure 3.15: Fight between two MUPPETS objects

#### 3.3.4.12 PlayLogo 3D

PlayLogo 3D [157] is a 3D role-playing multiplayer game designed primarily for children aged 6-13 years to teach them basic concepts of programming. Even though PlayLogo 3D is a multiplayer game, voice and text messages between players are not supported during the game to prevent disturbing the learners. Players are required to code their characters in the LOGO language - an educational programming language, widely known for its use of turtle graphics, in which commands for movement and drawing produced line graphics - and operate them in a virtual world. In particular, the virtual world is a spaceship X-15 located on a constellation of the Andromeda galaxy, where a competition is held each year among pilot-robots.

The game starts with an introductory video which shows the game scenario to the player. After that, the main characters of the game explain the game rules to the player in an indirect way. The full description of the rules can later be consulted in the help file. During the match, the player sees his/her robot in a third-person view (as exhibited in Figure 3.16) as well as the surrounding environment, which means that the robot holds its orientation in space and, thus, players cannot examine the whole virtual scene at a time. The right mouse key can be used to rotate user's point of view in all directions.

Matches are played between two players who act in turns trying to collide with the opponent's robot. Robots have a similar shape to that of the human body and respond to LOGO locomotion commands introduced by the users. Hence, reaching the goal involves three simple steps: orientate in 3D space, lock the current position of the opponent, and try to reduce the distance between the robots, avoiding possible obstacles.



Figure 3.16: Interface of the PlayLogo 3D game

#### 3.3.4.13 Gidget

Gidget [127] is a web-based game where players program a robot, named Gidget, using a simplified programming language developed specifically for the game. It aims to teach users how to design and analyze basic algorithms by challenging them to fix the internal code of Gidget, which had its logic damaged. The goal of the robot is to clean up a chemical spill at the factory and save the animals.

The interface of Gidget, presented in Figure 3.17, is composed of three vertically-split panels. The first panel contains the IDE with the damaged set of instructions to fix, the list of goals written as unit tests that are checked after executing the code, and the buttons to control the level of execution of the code. These buttons allow to run a single instruction at a time, one line of code, or the entire program, and halt the program. The second panel shows the graphical feedback, including the game world, all the characters and objects of the level, and a large speech bubble where a detailed explanation of the execution of each instruction is provided. The last panel shows the attributes of Gidget (e.g., energy, rotation, scale, and position) and any other object selected from the world panel. These attributes are updated after each instruction.

The game also provides an integrated development environment – Gidget Puzzle Designer (GPD) – used to create and edit Gidget levels. The interface of GPD is similar to the Gidget game, but also contains a collection of objects, sounds, and grounds from which the author can add elements to the game world, a widget to insert mission texts, and a range selector to set the initial energy.

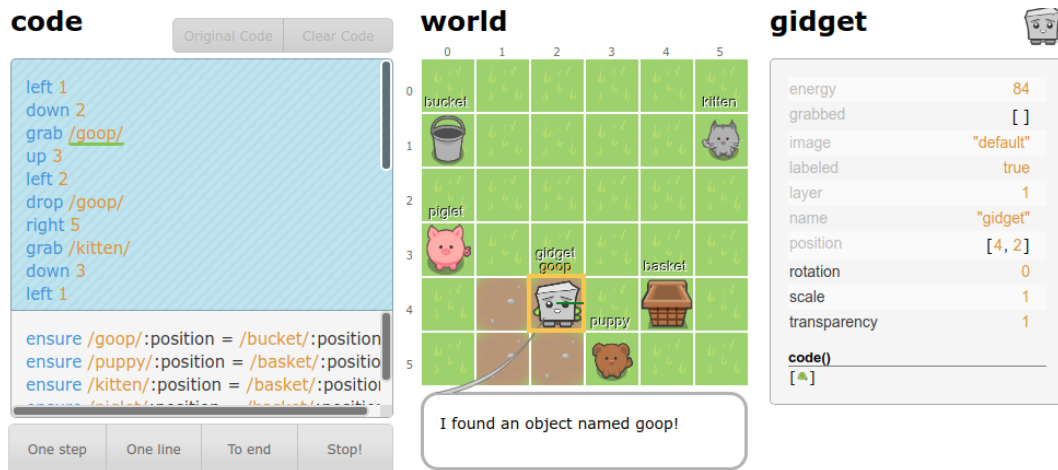


Figure 3.17: Interface of the Gidget game

#### 3.3.4.14 Leek Wars

Leek Wars<sup>10</sup> is a web-based multiplayer game where the player controls a character (a leek) through coding. The goal of the game is to beat other players' leeks during fights, awarding points to climb to the top of the ranking. The result of the fight is displayed as a 2D frame-by-frame movie (see Figure 3.18).



Figure 3.18: Leek Wars: fight between two leeks in a two-axis environment

As the level of the player evolves, he can buy and add equipment to his leek, such as weapons,

<sup>10</sup>[www.leekwars.com](http://www.leekwars.com)

chips, potions, and hats, and improve skills of his leek, namely life points, science, strength, magic, wisdom, frequency, agility, turn points, resistance, and movement points.

The user can develop several **SAs** in JavaScript using functions provided by the platform API to manipulate the leek and to get information of the environment of the fight. Then, the user can choose which **SA** to use in a fight, and a default to use in fights requested by other players. These **SAs** can be experimented against system-defined leeks, before entering in a fight.

The game has a solo tournament everyday in which users can freely register, and a cooperative mode where teams can fight against each other in the same battle field. These modes aim to captivate the player to come back regularly.

#### **3.3.4.15 Summary**

The previous sub-subsections present 14 games that teach concepts of programming. Each of these games uses a particular programming language, challenges students in a different way, or has some unique feature that distinguishes it from the others. Table 3.1 summarizes the main aspects of each game.

Game	Teaching Concept	Language	Challenge	Key Features
Catacombs	variables; conditionals; loops	micro-language	multiple-choice questions; fill-in the blanks of a code block	3D; explanatory messages; scoreboard; code scaffolding
Saving Sera	variables; conditionals; recursion; loops; quicksort	micro-language	find bugs (multiple-choice questions); fill-in the blanks of a code block; drag-and-drop sorting of code blocks	code scaffolding; authoring new quests
EleMental	depth-first search; recursion	C#	navigate the avatar in a binary tree; depth-first search walk	3D; interactive; explanatory messages;
Prog & Play	variables; conditionals; loops; structured programming	Ada, C, Java, OCaml, Scratch, Compalgo	write code; code battles	3D; multiplayer; authoring scenarios;
Wu's Castle	loops; arrays	C++	array management; navigate avatar; find bugs (multiple-choice questions)	interactive
RoboZZle	functions; recursion; conditionals	-	navigate avatar (graphical SA)	authoring new levels; scoreboard
LightBot	functions; recursion; conditionals	-	navigate avatar (graphical SA)	3D; authoring new levels; scoreboard
TALENT	variables; conditionals; loops	micro-language	drag-and-drop code; write code	explanatory messages; embedded IDE
RoboCode	structured programming; object-oriented programming; IDE usage	Java	write SA; code battles	competitive; multiplayer;
Code Rally	structured programming; object-oriented programming; IDE usage	Java; Node.js	write SA; code races	competitive; multiplayer;
MUPPETS	object-oriented programming	Java	create and manipulate objects; write code; compile; code battles	3D; multiplayer; collaborative; competitive; embedded IDE; authoring extensions
PlayLogo 3D	basic concepts of programming	LOGO	create avatar; navigate avatar with commands	3D; multiplayer; competitive;
Gidget	basic concepts of programming	micro-language	fix code	embedded IDE; authoring new levels; explanatory messages
Leek Wars	structured programming; object-oriented programming;	JavaScript	write SA; code battles; buy items and improve skills	embedded IDE; competitive; collaboration; skills & items scoreboard

Table 3.1: Overview of the games to teach specific programming concepts



## Chapter 4

# Previous Work

The current dissertation presents a game-based assessment environment for programming challenges. This environment is intended to be integrated in an ecosystem of e-learning tools already described in the literature.

In this chapter each of these tools is introduced, starting with a brief description and going through the essential aspects of them to the current work. Section 4.1 presents Mooshak 2, which will handle the evaluation and creation of game exercises. Section 4.2 describes the integrated learning environment – Enki –, in which the current work will be inserted.

### 4.1 Mooshak

Mooshak [124] is a web-based system that supports assessment in computer science. It can evaluate programs written in several programming languages, such as Java, C, C++, C#, and Prolog. Apart from the feature mentioned above, Mooshak provides means to answer clarification requests of a problem, reevaluate programs, track printouts, and much more.

Initially, Mooshak was designed to be a programming contest management system for International Collegiate Programming Contests (ICPCs). Later, support for other types of programming contests was added. Since then, it was used to manage several competitions all over the world, including ICPC regional contests, IEEEExtreme contests, and local contests. Eventually, it started being used in undergraduate programming courses as a teaching assistant tool [75, 83, 125] to create competitive learning environments, to give instant feedback on practical classes and to receive, validate and mark attempts to solve exercises.

Recently, version 2 of Mooshak<sup>1</sup> was released. In this version, the code base of Mooshak was completely reimplemented in Java with graphical user interfaces in Google Web Toolkit (GWT). Also, specialized environments, such as a computer science languages learning environment – Enki [156] –, were included. The novelty goes even further since other kinds of languages, such

---

<sup>1</sup><https://mooshak2.dcc.fc.up.pt/>

as diagrammatic languages, are now supported. This is particularly useful for teaching subjects that involve modeling, such as theory of computation (DFA), databases (EER) and software modeling (UML).

The next subsections describe some important details of Mooshak to the current work. Specifically, subsection 4.1.1 details how the evaluation engine works. Subsection 4.1.2 describes the feedback provided by Mooshak. Subsection 4.1.3 presents the interface for creating exercises. Finally, subsection 4.1.4 overviews the facilities that Mooshak offers for interacting with other e-learning tools.

### 4.1.1 Evaluation Engine

The cornerstone of Mooshak is the evaluator engine, whose role is to grade a submission by following a set of rules while generating a report of the evaluation for further validation from a human judge. The evaluation follows a black-box approach, i.e., the internal structure, design, and implementation of the program being tested are unknown to Mooshak. The process consists of two types of analysis: static, which checks for integrity of the source code of the program and produces an executable program, and dynamic, that involves the execution of the program with each test case loaded with the problem.

Firstly, static analysis checks if both the submitted problem is already solved and no more than one accepted solution should be considered, in which case the evaluation ends without classification. Then, it tests whether the metadata of the submission (team id, problem id, and language) is valid. If this test fails, the student will receive an “Invalid Submission”, and the evaluation ends. Otherwise, the size of the source code is checked. The classification given to programs which exceed the maximum size is “Program Size Exceeded”. Finally, if all the prior verifications succeed, the submitted source code is compiled using the command line defined in the administration interface. The compilation can be more or less stricter according to the defined flags. If the compilation issues a warning or an error, the evaluation aborts, and a “Compile Time Error” is written in the submission report. The final result of this analysis is an executable program, except for interpreted languages.

After static verifications, the dynamic analysis takes place. This part consists of running, for each test case, the command line defined by the administrator for execution, which typically uses the executable program obtained in the previous step. Every run will receive the input data of a test via the standard input and listen to the output of the program in standard output. Once collected, the output of the program is compared to the output of the test.

During the execution of the program, several errors can occur. These errors have different severity levels, which determine the final classification of the submission (highest severity rank found). Table 4.1 enumerates the classification attributed to different types of execution errors, ordered by increasing severity.

If the program’s output is the same as the test output, the submission is marked as “Accepted”.



Severity	Classification	Description
1	Presentation Error	The output seems to be correct but it is not presented in the required format. Since it is not always easy to distinguish this message from the wrong answer message, it is only sent in obvious cases.
2	Wrong Answer	The program runs through one or more test cases without a run-time error but the output did not match the expected output.
3	Output Limit Exceeded	The program generated an output too long for this problem; the limits are dependent on the test cases, but are usually low (default limit is around 100KB).
4	Memory Limit Exceeded	The program exceeded the allocated amount of memory.
5	Time Limit Exceeded	The program did not finish within the allocated amount of time.
6	Invalid Function	The program or evaluator has called an invalid function and/or an internal error occurred.
7	Invalid Exit Value	The program terminated with an invalid code.
8	Runtime Error	The program crashed, i.e. it exited prematurely due to a run-time error.
9	Requires Reevaluation	For some reason the program has to be re-evaluated.

Table 4.1: Classification for execution errors thrown during dynamic analysis

Otherwise, the normalized version of both, i.e., a version with all formatting characters<sup>2</sup> stripped off, is compared. If these outputs are equal then it is classified as a “Presentation Error”, else a “Wrong Answer” is assigned.

Besides the standard evaluation procedure, Mooshak enables the problem’s creator to define two types of special evaluators: static and dynamic correctors. Both are external programs that run in different steps of the evaluation process, and their exit status affects directly the classification of the submission, according to Table 4.2. Static correctors act between static and dynamic analysis to classify the source code of the program. These can be used to examine code quality, run unit tests, check used libraries, among others. On the other hand, dynamic correctors are executed after each test run and can be used to handle non-determinism, e.g., if a set of values can be printed in any order then dynamic corrector can convert the output to a standard form.

<sup>2</sup>Generally white characters, but also letters and punctuation, in problems that only involve digits

Exit Code	Classification
$-100 \leq N \leq 0$	Accepted (mark is equal to $ N $ )
1	Presentation Error
2	Wrong Answer
3	Evaluation Skipped
4	Output Limit Exceeded
5	Memory Limit Exceeded
6	Time Limit Exceeded
7	Invalid Function
8	Invalid Exit Value
9	Runtime Error
10	Compile Time Error
11	Invalid Submission
12	Program Size Exceeded
13	Requires Reevaluation
14	Evaluating

Table 4.2: Mapping from special evaluator's exit codes to classification

### 4.1.2 Feedback

In the early days of Mooshak, even though feedback was timely, it was often insufficient, consisting only of an error status. This amount of feedback is adequate in a competition, such as **ICPC**, where all teams have preconfigured workstations with the same compiler and compilation flags as those used for judging their submissions. Although, in heterogeneous environments, such as college classes, a simple classification is not enough. For example, a compilation error may or may not happen depending on the compiler version and flags.

Since Mooshak started to be used for pedagogical purposes, better feedback has been set as a major requirement, and some improvements have already been made. The first improvement consisted of including an error message along with the classification. By default, details of the errors are only added in compilation errors, but this can be configured in the administrator interface to be included in every type of error. A second improvement consisted of including information about failed and passed test cases.

In Mooshak 2.0, feedback is still a priority. Therefore, new specialized evaluators have enhanced feedback specific for their use case. For example, the diagram evaluator summarizes the differences between the attempt and the solution focusing on specific issues, and progressively disclosures more detail about them each time the student makes the same mistake. This feedback is presented both in visual and textual form.

### 4.1.3 Exercise Authoring

Exercises are the fundamental component of any tool for managing programming contests. However, despite some efforts to define a standard format for describing programming exercises [64, 196], most of these systems use their own format to store them. Hence, Mooshak also has its format to describe programming exercises. An exercise package of Mooshak usually contains a PDF or HTML statement, one or more solutions, tests, skeletons, and a manifest file describing the contents of the package.

Mooshak 2 contains a specialized environment, named Creator, for creating exercises that conform to its package specification. This environment, whose user interface is presented in Figure 4.1, allows the author of the activity to fill in the metadata as well as to attach external files either using drag-and-drop or writing directly in the embedded code editor. The editor contains a preview and an editing mode to enable the author to edit and see the outcome of the HTML code, PDFs, or other special formats. Furthermore, the editor mode varies according to the type of exercise being created. For instance, in diagram modeling activities, the solution is built on a visual diagram editor instead of the usual code editor.

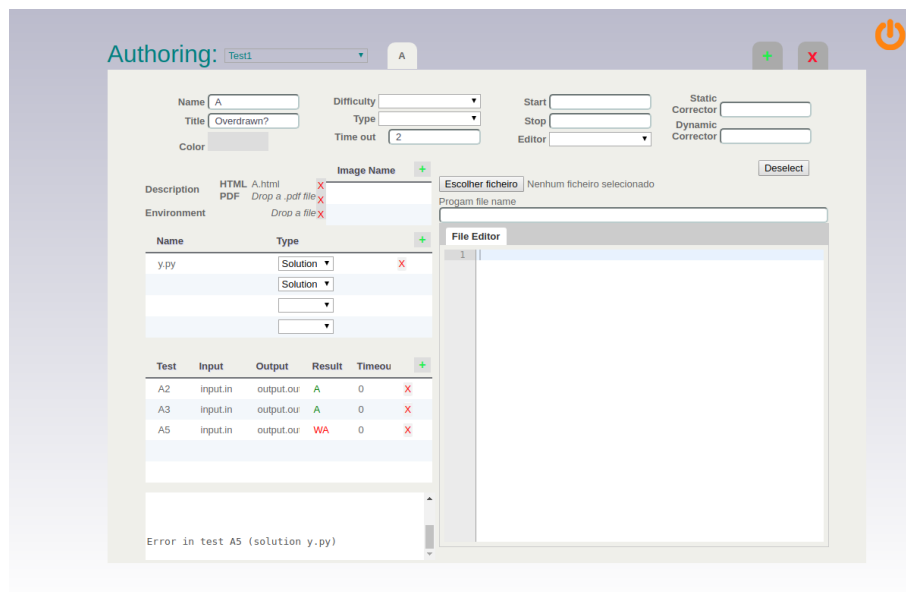


Figure 4.1: User interface of the Creator environment of Mooshak 2

As soon as a solution or test is modified, Creator automatically checks the solution against the provided test cases. The result of each test is written on the respective line of the data table and possible compilation errors are exhibited below.

### 4.1.4 Interoperability

Many academic institutions have already adopted a Learning Management System (LMS) in their courses, to organize and share resources, hand out assignments, and report student's

performance [53]. Hence, new tools, such as Mooshak, that aim to enter in an e-learning ecosystem, must be capable of interoperating with the existing ecosystem.

Consequently, some research came up with different approaches to couple LMS with other tools, such as redefining LMS based on service-oriented architectures [4, 39, 80], including a service layer in the LMS architecture [46], and providing support for interoperability specifications [123]. The latter is primarily based on Instructional Management System (IMS) specifications, particularly the Learning Tools Interoperability (LTI).

Since there is no standard way to integrate with LMSs, Mooshak 2 implements LTI specification, which is the less obtrusive of the mentioned options and supported by most LMSs. Although, some systems that implement the LTI were not able to fully integrate with Mooshak 2, only Moodle and Sakai coupled successfully with it [168].

## 4.2 Enki

Enki [156] is a web-based environment of Mooshak 2.0 for learning computer science languages, with a graphic user interface that mimics an Integrated Development Environment (IDE). This environment blends assessment and learning, integrating with several kinds of tools, as depicted in Figure 4.2. It integrates with a gamification service – Odin [155] – to support the creation of leaderboards, to reward students for their successes, among others. Also, it interacts with an educational resources sequencing service – Seqins [167] – to offer different learning rhythms according to the skills of each student.

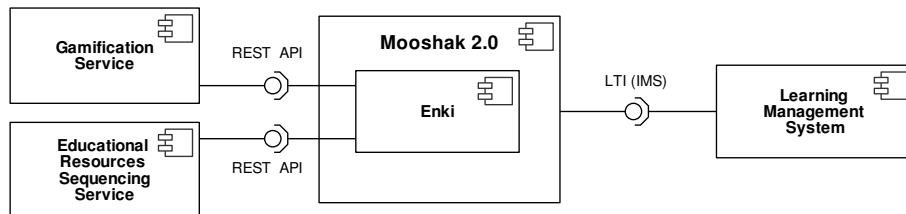


Figure 4.2: Components diagram of the network of Enki

### 4.2.1 Graphical User Interface

The Graphical User Interface (GUI) of Enki, presented in Figure 4.3, is one of its key features. Enki was built to be applied in heterogeneous contexts, from introductory high school and college courses to massive online open courses. So, it doesn't assume anything about the prior knowledge of the student neither about the device they are using. Also, students may be in the presence of an instructor or alone. Despite these limitations, it aims to introduce students to the IDEs, which are typically used for programming enterprise applications.

As a typical IDE, such as Eclipse and NetBeans, the GUI of Enki is divided into regions, each

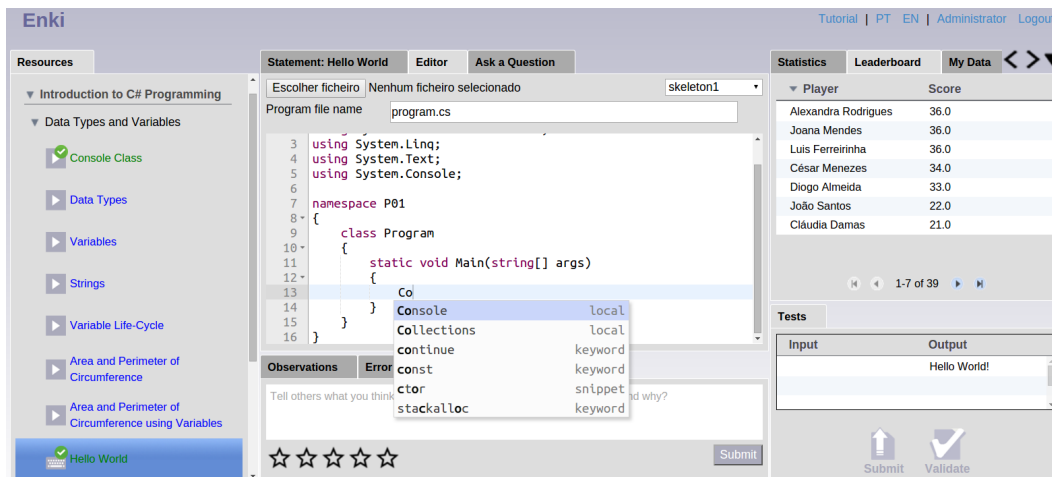


Figure 4.3: Graphical user interface of Enki

one containing several overlapping windows, organized using tabs. These regions are resizable, and their windows can be moved among different regions.

The components of the interface are widgets arranged in different windows, that communicate with each other. On the left side of the GUI, there is a particularly important widget, the resources tree. This widget drives the student interactions, by presenting the course contents organized by levels. Each level may hold educational resources of different types, such as text (HTML or PDF), multimedia or exercises. The resources are presented in the cell tree with an appropriate icon and color, reflecting its type and state (available, solved/seen, unavailable or recommended), respectively.

The center region contains the main widgets, which are directly related to the selected resource: expository or evaluative. On the hand, expository resources are presented in a single window with specialized viewers, such as a PDF viewer or a video player. On the other hand, evaluative resources are distributed through several windows, each with a different role: show the statement, edit the solution, submit a clarification request, and see the previous submissions. This modular design allows to easily change the editor from a coding exercise to a diagram modeling exercise, or even add another window with an additional feature.

On the bottom region, it has also widgets which are commonly found in IDEs, such as the observations/feedback window and the error list. Although, it also contains a widget to rate and comment a resource, and to link to related resources.

The right region is divided into two regions: upper and bottom. The upper part holds gamification-related windows, such as the leaderboard and the achievements, and windows that summarize global and personal progress, such as profile data and statistics of the problem. The bottom part can contain additional widgets, such as a widget to write input test cases.

### 4.2.2 Gamification Service

The gamification service plays an important role in the architecture of Enki. It is responsible for engaging students in the learning process through gamification features, such as leaderboards and achievements. There are already many gamification services that can leverage on their authentication services and massive user base. Although, these services that depend on external authentication do not fit into a network of e-learning systems since they already operate on a single sign-on system.

This situation has led to the creation of a new gamification service, called Odin [155]. This service provides a REST API, similar to the Google Play Games Service API, that requires institutions to be authenticated instead of end-users (i.e. students).

Odin is written in Java using Jersey, an open-source framework that is the reference implementation of the Java API for RESTful Web Services, extending it with additional features and utilities to further simplify RESTful service. The data is stored on a Redis NoSQL database, which provides an open-source and advanced key-value storage and cache solution.

## Chapter 5

# Asura

Asura is an automatic assessment environment for game-based programming challenges. Its main goal is to provide a means to engage students through games, requiring instructors a comparable effort to that of creating an International Collegiate Programming Contest (ICPC)-like problem. Hence, this environment enables students to enjoy of learning activities using unique features of games, such as graphical feedback, game-thinking, and competitiveness, while including a set of tools designed to support authors during the process of creating Asura challenges, such as a framework and a Command-Line Interface (CLI) tool.

Authors are required to develop the referee and state management logic of their specific game, as part of the authoring process. Additionally, they can code Software Agents (SAs), wrappers for SAs, and skeletons for SAs. The SAs from teachers, also known as control SAs, are used to evaluate the submissions of the students, to check if they meet the requirements to participate in other matches. Typically, SAs that do not break game rules against any of the control SAs get an ACCEPTED and, thus, are able to enter the tournament and be used in validation matches. Wrappers provide an higher level of abstraction for playing a game. They take care of writing and reading messages to/from the game manager, offer methods to check game state and execute actions, and much more. Skeletons, which are already supported in Mooshak 2 for traditional programming exercises, define a structure for the SA on which players can build on.

During the development of the SA, students can validate its effectiveness by executing matches against any previously submitted SAs that can be selected from a board in Enki, containing all accepted SAs. A match runs on the Asura Evaluator, which evaluates the code of the SA, starts a process with it, initiates the opponents' processes, and leverages the rest of the evaluation on the game manager. The outcome of the match is a JavaScript Object Notation (JSON) object adhering to the JSON Schema defined for game movies (see Section D.1). This object is given to the Asura Viewer by Enki, which transforms it into an adequate format to display to the learner.

Once the time to submit SAs ends, instructors are able to organize tournaments among the submitted SAs, similar to those found on traditional games and sports. The tournament is displayed in an interactive Graphical User Interface (GUI), allowing the viewers to control which

games they want to see, navigate through stages, and check the ranking.

The rest of this chapter describes the architecture of Asura and its components. Section 5.1 presents the architecture of Asura. Section 5.2 details the component for authoring game-based programming challenges, named Asura Builder. Section 5.3 outlines the main characteristics of Asura Evaluator. Section 5.4 describes the Asura Tournament Manager. Section 5.5 overviews the design and implementation of Asura Viewer.

## 5.1 Architecture

Asura follows a multi-component architecture composed of three individual components, named Asura Viewer, Asura Builder, and Asura Tournament Manager, and a component designed to integrate transparently with the automatic assessment of Mooshak, the Asura Evaluator. This architecture aims to guarantee that every component is easily replaceable and extendable while allowing them to be integrated independently in different e-learning ecosystems.

Figure 5.1 depicts the implemented architecture, in which Asura integrates with two other systems already described, Mooshak 2 and Enki. Asura Viewer, Asura Evaluator, and Asura Tournament Manager provide Java interfaces with the necessary methods to integrate with other components. These interfaces are presented in Figure C.1.

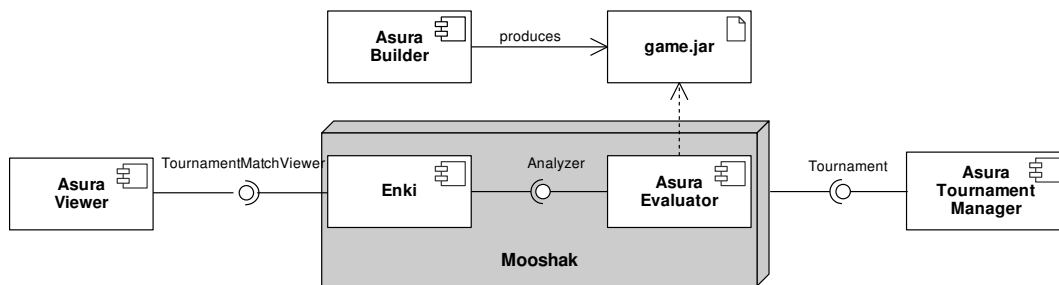


Figure 5.1: Diagram of components of the architecture of Asura

Asura Evaluator is a small server-side package developed inside Mooshak 2, whose main class is the `AsuraAnalyzer`, a specialization of the main analyzer of Mooshak 2 – `ProgramAnalyzer` – for conducting the dynamic analysis using the provided Java ARchive (**JAR**). Both `AsuraAnalyzer` and `ProgramAnalyzer`, which grades submissions to **ICPC**-like problems, implement a common interface – `Analyzer` – that allows evaluator consumers to integrate seamlessly with any of them. In this case, the consumer is `Enki`.

The Asura Tournament Manager is a Java library providing an interface `Tournament` for its consumers to organize tournaments. It receives a set of players and the structure of the tournament to carry out, and returns the matches to run on the Asura Evaluator one after another. The library is used by Mooshak 2, which obtains the structure and players of the tournament from a wizard embedded in the administrator **GUI**.



Asura Viewer is an external Google Web Toolkit (**GWT**) widget that can integrate in any **GWT** environment, supplying it with an interface `Viewer` that enables them to display either tournaments or matches. This interface expects a **JSON** object complying either with match **JSON** Schema (shown in Section D.1) or tournament **JSON** Schema (presented in Section D.2).

Asura Builder is a set of tools that aims to facilitate the creation of Asura challenges. This component is completely independent of the rest of the ecosystem, since its main function in the architecture is to produce the **JAR** containing the game implementation to be used by the Asura Evaluator.

## 5.2 Asura Builder

The Asura Builder is a standalone component composed of multiple tools dedicated to the authoring of game-based coding challenges. It includes a Java framework that provides a game movie builder, a general game manager, several utilities to exchange complex state objects between the manager and the **SAs** as **JSON** or eXtensible Markup Language (**XML**), and general wrappers for **SAs** in several programming languages. The framework is accompanied by a **CLI** tool to easily generate Asura challenges and install specific features, such as support for a particular programming language, a default turn-based game manager, among others. Even though the authors are required to program the challenges in Java, players can use their preferred programming language to code their **SAs**.

Each of the next subsections describes a feature of Asura Builder that aims to reduce the additional hurdle of building game-based programming challenges. In order to better understand how Asura Builder supports the development of Asura challenges, the steps involved in the creation of a Tic Tac Toe game to teach how to use arrays in Java in an *Introduction to Programming* course are reported throughout the subsections according to the part of the component described in them. Subsection 5.2.1 describes the builder of graphical feedback. Subsection 5.2.2 details the referee and state management of the game. Subsection 5.2.3 presents the communication between the game manager and the players. Subsection 5.2.4 introduces the two kinds of wrappers that Asura supports to facilitate **SA** development. Subsection 5.2.5 provides an overview of the **CLI** tool. Subsection 5.2.6 summarizes the integration between the various components of the game described along this section.

### 5.2.1 Game Movie Builder

Most of the necessary effort for building video games is required by graphics. They determine the players' first impression on the game and they provide the best feedback of the actions executed during the game. Asura games are not exceptions. In Asura, graphics are abstracted as a game movie. A game movie consists of a set of frames, each of them containing a set of sprites annotated with information about their location and transformations. In this way,

the representation of the game movie is very compact since each frame is just a collection of objects, completely described by four numbers (`x`, `y`, `rotate`, and `scale`) and an optional object (`view_window`) containing four numbers (`start_x`, `start_y`, `width`, and `height`). Besides that, a movie also includes metadata information, such as `title`, `background`, `width`, `height`, `fps` (i.e., number of frames to display per second), `anchor_point` (i.e., sprite point relative to which coordinates are given), the set of players, and the set of sprites.

This abstraction facilitates the construction of graphical feedback by defining a standard way to describe it, independently of the game. Furthermore, Asura Builder offers an interface (and an implementation) to easily build these game movies, as presented in the Unified Modeling Language (UML) class diagram of Figure C.2. The interface includes methods to manage metadata information, add a frame to the movie – `addFrame()` –, insert items into a frame applying the necessary transformations – `addItem(sprite: String, x: int, y: int, rotate: double, scale: double)` –, include messages to players (e.g., logs of their SAs) – `addMessage(playerId, message)` –, set the observations and classification of a player, push/pop frame states from a stack – `saveFrame()/restoreFrame()` –, terminate the game movie indicating an error – `failedEvaluation(e: BuilderException)` or `failedEvaluation(e: PlayerException)` –, among many others.

When an evaluation fails due to an exception in the Builder component or in the game itself (i.e., `failedEvaluation(e: BuilderException)`), the status of all submissions is automatically set to `REQUIRES_REEVALUATION`. Otherwise, if the problem is caused by the player (i.e., `failedEvaluation(e: PlayerException)`), the exception contains the classification to assign to the faulty SA according to the issue (e.g., `TIME_LIMIT_EXCEEDED`, `WRONG_ANSWER`, etc.).

An instance of the game movie builder is passed as a parameter to the game state management object to be able to update the movie during the game loop. The author does not need to be aware of the type of match being executed since there is no distinction between game movies built for validations or tournament. Once the evaluation is completed, the movie can be obtained as a JSON string either through `toString()` or directly written into a stream with `toFile(stream: OutputStream)`.

Considering the Tic Tac Toe example, initially, the metadata such as `title`, `frames-per-second`, `width`, `height`, `anchor point`, `X` and `O` sprites, and `players` needs to be set. Then, in each game round, the necessary work to build the movie sums up to four instructions: (1) add a frame, (2) restore the previous frame from the stack, (3) add the new symbol, and (4) save the frame to the stack. After the game ends, it is just required to add a frame, restore the previous frame from the stack, and assign points (i.e., using `setPoints(player: String, points: int)`), classifications (i.e., using `setClassification(player: String, classification: MooshakClassification)`), and observations (i.e., using `setObservations(player: String, observations: String)`) to the winner and the loser accordingly.

### 5.2.2 Game Manager

Most games and sports require a referee or game manager (when referring to video games) to ensure that the game rules are followed, decide which player takes the next turn, and declare the winner(s). However, a game manager has several other tasks such as to process player actions, keep track of the game state, inform players about the current game state, provide feedback, among others. In Asura, all these tasks are on the responsibility of the `GameManager`, which also needs to grade students' **SAs** (i.e., assign a mark and a classification, and give some optional remarks to learners).

Even though most of this work is specific to each game and consequently needs to be performed by the author, the Asura Builder provides an abstract class defining the expected interface of a game manager to integrate with the Asura Evaluator and implementing the generic functionality. The structure of the `GameManager` comprises the following methods:

`getGameName() : String` provides the full name of the game. By default, the abstract `GameManager` returns the name of the implementing class, excluding the suffix “Manager”. Nevertheless, if the new challenge is scaffolded with the **CLI** tool (see Subsection 5.2.5), this value is overridden by the provided game name.

`getMinPlayersPerMatch() : int` and `getMaxPlayersPerMatch() : int` define the minimum and maximum number of players per match, respectively. These methods are specific to each game and, by default, the minimum is one whereas the maximum is unset.

`manage(players: Map<String, Process>)` is the method that starts the game evaluation. It encompasses several steps including (1) the initialization of the game state, (2) the connection of its input stream to the output stream of the players' processes to receive their actions, (3) the binding of its output stream to their input stream to update them with changes on the game state, and (4) the invocation of `manage(players: Map<String, Process>, state: GameState)` which actually runs the match with the given players' processes, attaching fallbacks to set players' statuses (i.e., mark, classification, and observations) in case that a `BuilderException` or a `PlayerException` is raised. Only the invoked method needs to be implemented by specialized managers to define the playing order and manage the game state accordingly. Some of these specialized managers, such as a turn-based game manager, are provided by the framework and can be easily integrated into a new challenge, using the **CLI** tool.

`getPlayerStatus(id: String) : GamePlayerStatus` returns the status of the player in the game, which is automatically obtained from the last frame of the game movie.

`getGameMovie() : GameMovie` returns the movie of the game.

`getGameMovieString() : String` obtains the movie of the game execution as string.

Furthermore, the game manager provides methods to read the actions from the players into Java objects that already deal with common issues such as timeout, bad formatting, and runtime exceptions. A method to send state update objects to players which handles possible unexpected process terminations is also foreseen.

Every action executed by a player changes the game state accordingly. In order to manage the game state, which is unique to each game, the `GameManager` leverages on the game-specific implementation of the `GameState` interface. This interface defines several methods, particularly:

`prepare(b: GameMovieBuilder, players: Map<String, String>)` allows initializing the state before the game starts. This typically includes fill-in the header of the game movie and set up the initial values on local variables.

`execute(b: GameMovieBuilder, playerId: String, action: PlayerAction)` updates the game state according to the action of a concrete player.

`getStateUpdateFor(player: String): StateUpdate` obtains the object that needs to be sent to a certain player to update it about changes to the game state.

`endRound(b: GameMovieBuilder)` ends the round when all players' commands were executed.

`isRunning(): boolean` checks if the game is still running.

`finalize(b: GameMovieBuilder)` finishes the game and sets the final status information of each player.

Most of these methods receive a game movie builder as a parameter, allowing the state object to manage the frames of the movie, reflecting any changes made to it.

As a referee, when a player breaks the rules of the game (e.g., does an invalid action), the game manager must also act. If the violation of the player prevents the game from continuing, the game ends marking the infraction of the **SA** in the game movie. This can be achieved by raising a `PlayerException` without catching it, since it is later processed on the abstract `GameManager`. Otherwise, the game proceeds but the faulty **SA** gets a “Wrong Answer” at the end.

At the beginning of a match, the game manager of Tic Tac Toe has to decide who plays with Xs (the other player takes Os). Then, it queries players in turns, starting with the Xs, until someone either places 3 equal symbols in a row, column, or diagonal, or puts the 9th mark. In each turn, the manager: obtains an update object from the game state with the last position marked to send to the player (i.e., using `getStateUpdateFor(player: String): StateUpdate`); reads the action from the player; leverages the execution of the action on the game state with `execute(b: GameMovieBuilder, playerId: String,`

`action: PlayerAction);` and terminates the round using the `endRound(b: GameMovieBuilder)`. The actual implementation of the Tic Tac Toe game manager and the game state are detailed in Section E.2 and Section E.3, respectively.

### 5.2.3 Communication Manager-Players

The dynamic analysis of Asura relies on interprocess communication between the master (i.e., the game manager) and its slaves (i.e., the SAs participating in the evaluation). This communication is carried out by having each process normally reading messages from its input stream and writing messages to its output stream, after crossing these as described in Section 5.3. This model of communication follows a master-slave approach, but slaves can send messages to the master when it authorizes them.

The exchanged messages can be either JSON or XML formatted strings separated by a newline character, complying to the data models depicted in Figure 5.2. On the one hand, the `GameManager` sends state updates to the players, containing a type, which identifies the state update, and a comprehensive description of the current game state. On the other hand, the messages sent from the players to the `GameManager` contain the action (a command) that the player wants to execute as well as a list of messages for debugging purposes.

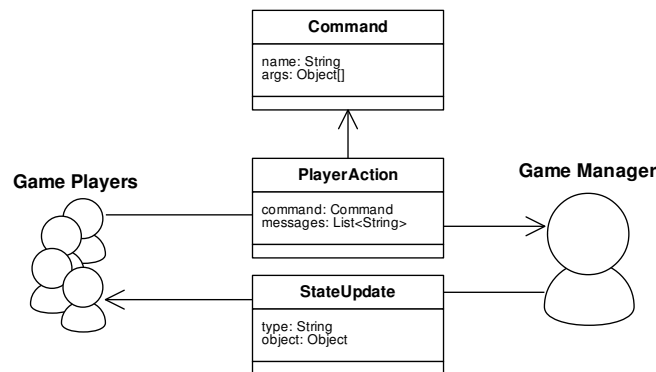


Figure 5.2: Data models exchanged between the game manager and the players

The implementation of this communication channel aims for complete abstraction of the underlying process from both authors and players. In this way, only methods to send and read any object or processed value are visible whereas the conversion to a valid data model and formatting as JSON or XML is done by the framework (and possibly further obfuscated by wrappers) under the hood.

In the Tic Tac Toe example, update messages sent from the game manager to the players contain a fixed type “LAST\_PLAYED” and an object with an integer value between 1 and 9. Actions dispatched by the players to the game manager include a command of type “PLAY” with a single argument which is the position to place the symbol. Both kinds of messages are sent as JSON and automatically transformed into a Java object on the recipient.

### 5.2.4 Wrappers

In Asura, a wrapper consists of a set of methods that aim to give players an higher level of abstraction so that they can focus on solving the real challenge instead of spending time processing I/O or doing other unrelated tasks. They can also be used to increase or decrease the difficulty of the problem by changing the way that **SAs** interacts with the game, without modifying the game itself. For instance, a wrapper for a car of a racing game with collectible items could have a method `headingTo(item: object): number` which returns the necessary offset in the heading to catch the item, in order to alleviate the hardness of the exercise.

Hence, the Asura **SAs** have been designed to follow a 3-tier architecture in which they constitute the top layer (or logic layer), that is implemented by the player, and the two remaining levels are associated with the distinct kinds of wrappers supported in Asura i.e., the Game-specific Wrapper layer (or business layer) and the Global Wrapper layer (or data access layer), as depicted in Figure 5.3.

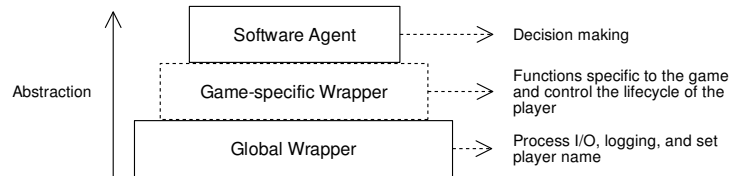


Figure 5.3: 3-tier architecture of an Asura SA

Global wrappers are generic wrappers implemented by the framework which are not dependent on the game being played. They are on the lowest level of this architecture, which means that there is no abstraction to access data except the input/output capabilities of the language. The purpose of the layer is to provide simplified access to data for upper layers, particularly by offering a means to obtain or send an object (in terms of object-oriented programming) or a processed value from the input stream or to the output stream without the need to be aware of the format of the data being exchanged or process I/O. In addition to that, these wrappers already provide functionality that is common to players of any game such as a method to log messages, sending the player's identification to the game manager.

Game-specific wrappers are optional wrappers, developed by the challenge author, that are only related to the context of the game in production. They build on top of the syntactic sugar defined in the bottom layer to provide an API to access particular parts of the current game state and perform supported player actions. As an example, a wrapper for a Go player might define a method `addStone(x: int, y: int)` to add a stone of the player in position  $(x, y)$  as well as a method `lastOpponentStone(): object` to get the last move of the opponent. Moreover, these wrappers also implement the player lifecycle, i.e., the player loop that reads updates and asks the **SA** for their actions.

Following up the game challenge of this section, the wrapper of a Tic Tac Toe **SA** defines two helper methods, `getLastPlayed(): int[]` which provides the  $x$  and  $y$  position of the last

`move` and `play(x: int, y: int)` that sends the action to the game manager to play in the given position. The wrapper also implements the methods `update` and `run` to store the last played position when an update is received and specify the execution lifecycle of the **SAs**, respectively. A detailed implementation of this wrapper is presented in Section E.4.

### 5.2.5 CLI Tool

The **CLI** tool of Asura Builder consists of a command-line utility for scaffolding Asura challenges. It has been developed in Python using `click`<sup>1</sup> – a Python package for creating **CLIs** in a composite way requiring a minimal amount of code – and `cookiecutter`<sup>2</sup> – a command-line application and Python package to create projects from cookiecutters (project templates).

This tool allows generating the barebones project for a new Asura challenge with one of the recommended infrastructures in a single command line `asura-cli create [TEMPLATE]`, where the optional argument `TEMPLATE` indicates the template from which the project should be derived. Currently, there are three templates to choose from, which differ in the build automation and dependency management tools they use, namely Maven (`maven`), Gradle (`gradle`), or none (`simple`). After sending this command, a series of queries are issued to the author in order to obtain the required information to generate the project (see Listing E.1). The outcome of the command is a completely structured and ready-to-use project, as presented in Listing E.2, with lots of boilerplate code, only lacking the implementation of the game logic and optionally game-specific wrappers and **SA**.

Furthermore, the **CLI** tool intends to support the author during the full development process, including the creation, implementation, and deployment of the challenge. For this purpose, it also includes a command `asura-cli import-manager TYPE` for importing pre-built game managers from a collection containing a turn-based game manager in which players act by turns, an all-at-once game manager in which players act all at the same time frame, among others, and commands to add and remove support for programming languages – `asura-cli add-language LANGUAGE` and `asura-cli remove-language LANGUAGE`.

Concerning the deployment phase, the tool also provides instructions to generate a sample problem statement – `asura-cli add-statement` – and package the game into a **JAR** ready to import to Mooshak 2 – `asura-cli package`.

The scaffolding of Tic Tac Toe requires a single command `asura-cli create maven`, since the language Java is already added by default. The answers to the questions issued by this command as well as the resulting project structure are presented in Section E.1.

---

<sup>1</sup><http://click.pocoo.org/>

<sup>2</sup><https://github.com/audreyr/cookiecutter>



### 5.2.6 Overview of Tic Tac Toe

A match of Tic Tac Toe encompasses three participants: one Game Manager and two **SAs**. Each of these components runs on an independent process forked by the evaluator and communicates in a master-slave approach through **JSON** messages.

Looking the **SAs** in detail, they can be separated into three components: the global wrapper (i.e., `PlayerWrapper`), the Tic Tac Toe wrapper, and the **SA** itself. The former does a number of tasks as the process starts, including (1) initialize the specific **SA**, (2) obtain the name provided in its implementation of the method `getName()`, (3) send the name to the game manager, (4) call the `init()` method of the **SA** which allows any initializations to be performed before the actual match starts, and (5) invoke the `run()` method implemented by the Tic Tac Toe wrapper to start the playing loop. Furthermore, it is also the closest component to the communication channel with the game manager, which means that it has to preprocess the incoming and outgoing messages to convert them from **JSON** into Java objects and vice-versa.

The Tic Tac Toe wrapper (activated by the `run()` call) enters a loop which (1) waits a state update from the game manager, (2) calls the `execute()` method of the **SA** when the update is received, and (3) sends the action to the global wrapper to forward it to the game manager. This component does not have to process **JSON** since the global wrapper already provides it with the corresponding Java objects and accepts Java objects or plain values as actions.

The **SA** (as described in Section E.5) reads the last played position from the Tic Tac Toe wrapper using the helper method `getLastPlayed() : int[]` and plays on the next available position with `play(x: int, y: int)`.

Figure 5.4 presents a simplified **UML** sequence diagram of the interactions between the game manager and the **SA** playing Os. This diagram includes the initialization of the **SA** as well as the process involved in the first play.

## 5.3 Asura Evaluator

The evaluator engine of Mooshak grades a submission by following a set of rules while generating a report of the evaluation for further validation from a human judge. This evaluation follows a black-box approach. The process consists of two types of analysis: static, which checks for integrity of the source code of the program and produces an executable program; and dynamic, that involves the execution of the program with each test case loaded with the problem.

The Evaluator component of Asura inherits the static analysis of the Mooshak's evaluator engine. The only difference is that the compile command line can include a language-specific player wrapper, present in the `game.jar` file, for complex games. However, the dynamic analysis is completely reimplemented. Instead of test cases with input and output text files, Asura Evaluator receives as input a list of paths to opponents' submissions. The type of evaluation



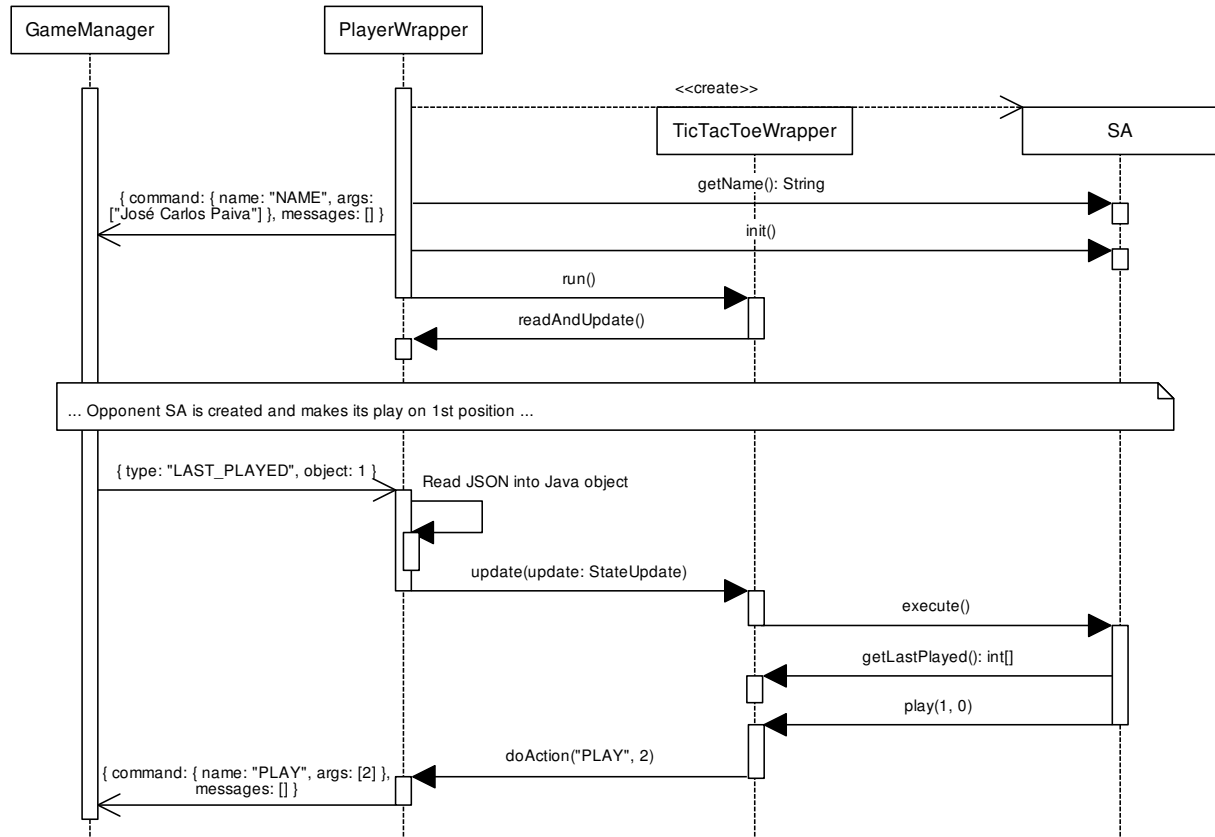


Figure 5.4: Sequence diagram of the interaction between Tic Tac Toe game components

either a validation or a submission determines the source of the competitors. Validations do not count for evaluation purposes but are rather a means for the learner to experiment how his/her **SA** behaves in a match against any existing **SA**. The opponents are selected by the learner itself from a list containing all the last accepted **SAs** from the students as well as the control **SAs** included by the author. On the contrary, submissions are considered as attempts to solve the challenge and as such are evaluated equally for all participants. In this case, the opposing contestants are the control **SAs** provided by the author.

The input paths refer to opponents' submissions already compiled and, thus, the component only initializes a process from the compiled set sources. After that, it organizes matches containing the current submission and a distinct set of the selected opponents' submissions. The length of this set depends on the minimum and maximum number of players per match, which are specified by the game manager. At this point, the evaluation proceeds on an instance of the specific game manager, which is instantiated from the `game.jar`.

The game manager receives the list of player processes indexed by the player ID and crosses the input and output streams of each of these processes with itself, as demonstrated in Figure 5.5. This allows the game manager to write state updates to the input stream of the **SA** and read actions from its output stream, as typical I/O operations from the **SA** perspective. The execution of the game is completely controlled by the game manager, which is responsible for keeping the

SA's informed about the state of the game, querying the SAs for their actions at the right time, ensuring that the game rules are not broken, managing the state of the game, and classifying and grading submissions.

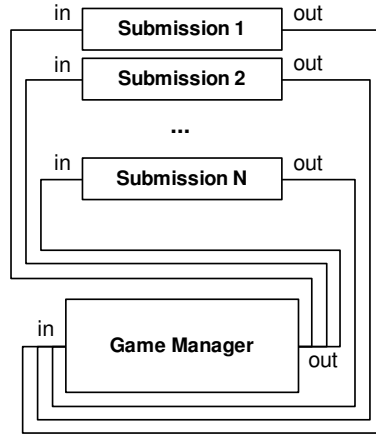


Figure 5.5: Diagram of the crossing of input/output streams between the game manager and the SAs

Finally, the status obtained from the matches containing the observations, mark, classification and feedback are compiled into a single status which is added to the submission report, and sent to the client.

## 5.4 Asura Tournament Manager

The Asura Tournament Manager is a Java library for organizing tournaments among SAs submitted and accepted on an Asura challenge. Tournaments aim to give a final objective to students by inviting them to engage in a contest realized at the end of the submission time. Hence, the challenge is not just about solving the problem, but also to prepare the SA to win a final competition. During the preparation phase, students can execute “friendly” matches against any previously approved SA from each other to get an idea of what they can expect to achieve in the tourney.

Tournaments of Asura follow similar models to those found on traditional games and sports competitions. They can have several stages, each of them arranged according to one of the available tournament formats: round-robin, Swiss system, single elimination, and double elimination. In group-based formats (i.e., round-robin variants and Swiss system), players are, typically, split into multiple groups and each of the groups executes an inner competition as specified by the format rules. A stage (or each group within the stage) is composed of a series of rounds in which all competitors either participate in a single match or have a bye (i.e., the contestant advances directly to the next round of the tourney in the absence of assigned opponents).

Matches are played between two or more participants. They are generated based on the tournament format, one after another, executed on the Asura Evaluator, and their outcome is

sent back to the Asura Tournament Manager. The result is a list of player data containing the points obtained by each player in the match as well as any additional feature values that could be used as tiebreakers, either for the match or rankings. The described data model is presented in Figure 5.6.

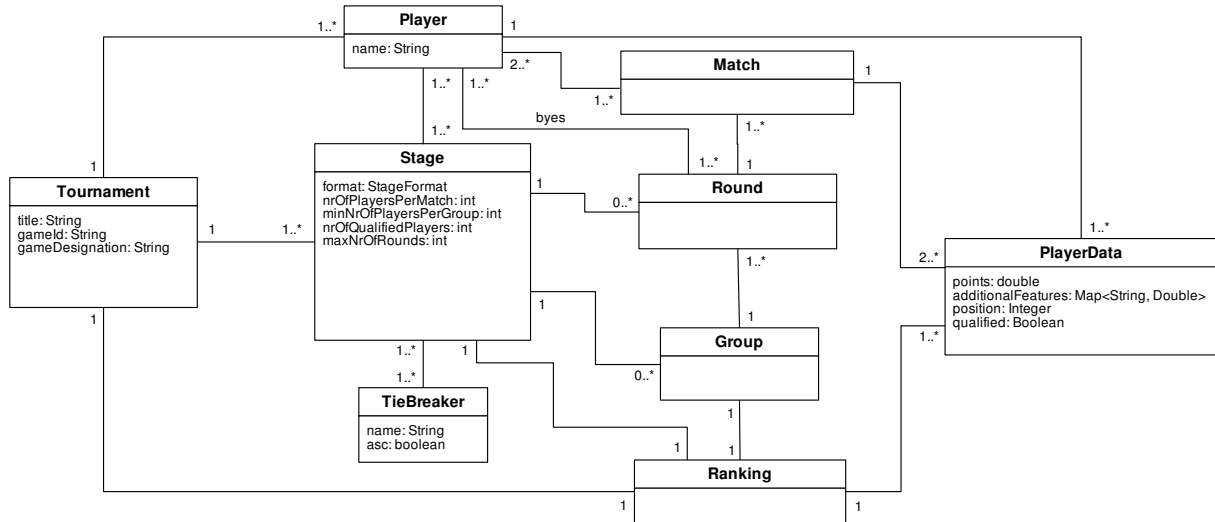


Figure 5.6: Data model of the Asura Tournament Manager

The interaction with the library is performed through the interface `Tournament`, which provides a sequential and seamless way to run the tournament. Firstly, the mandatory metadata of the tournament such as the title, game ID, game name, and participants should be provided to the tournament organizer. Then, the stages are added – `addStage(format: StageFormat)` –, configured (i.e., set the parameters to organize the stage), and started – `startStage()` – one by one. The necessary configuration options depend on the format of the stage and may include the number of players per match, the minimum number of players per group, the number of qualified players, the maximum number of rounds, the type of result of a match (e.g., win-draw-loss or position-based), and tiebreakers both for rankings and matches. The operations made on the tournament instance, after adding a stage, are delegated on the class implementing the specific tournament format, through the common interface `RunnableStage`, until the `endStage()` is called. Once the stage has started, the next match to execute can be obtained using `nextMatch()`, which returns a `MatchBuilder` containing the set of competitors participating in the match. The execution of the match produces the player data required to fill-in the `MatchBuilder`, which can then be built into a match in order to notify the tournament manager of the result – `endMatch(match: Match)`. The result of the match is analyzed by the `MatchAnalyzer` adequate to the result type, to update rankings accordingly. This procedure is repeated for all matches of the stage (i.e., while `hasMatches(): boolean` is `true`).

The output of the Asura Tournament Manager is a **JSON** object complying to tournament **JSON** schema (see description in Section D.2) formatted as a string, which is collected using `getJson(): String`. This schema is a formalization of the data model presented in Figure 5.6.

A simplified UML class diagram of the implementation of Asura Tournament Manager is depicted in Figure C.3.

The integration of Asura Tournament Manager in Mooshak 2 included the creation of a wizard to define the structure of the tournament in the administration GUI, as presented in Figure 5.7. The wizard includes fields for setting up each metadata parameter of the tournament as well as to configure the stages.

Figure 5.7: Wizard embedded in the Administration GUI to organize tournaments

While designing a tournament, it is important to find a right balance between the amount of resources that it would require (e.g., running time, number of games, etc.) and how fair it is to competitors (i.e., whether it finds the best player). Hence, each of the next subsections presents a different tournament format and describes how its implementation distinguished from the rest. Subsection 5.4.1 details round-robin. Subsection 5.4.2 introduces the Swiss system. Subsection 5.4.3 describes the single elimination. Subsection 5.4.4 outlines the dissimilarities between the single and double elimination.

### 5.4.1 Round-robin

In a round-robin tournament, all players (or teams) do an equal number of matches against each other. The number of times that a participant faces each opponent is, typically, one (single round-robin) or two (double round-robin). The results are cumulative through all matches played, and the final results determine the final standings.

This format is more effective when the number of entries is low or games are played quickly (e.g., badminton and table tennis). However, traditional sports such as hockey, football, or

basketball, in which matches take longer and the number of entries is around 20, also use this format for league play, having matches in separate days.

A round-robin format is problematic when the number of participants is high. For example, a tournament with 32 entries would have 496 matches. A common variation of this format that allows to reduce the number of matches consists of splitting participants into balanced groups and execute the round-robin in each of them separately. For instance, in the previous example dividing the players into groups of 4 would take 48 matches to complete (8 groups, 3 rounds per group, and 2 matches per group per round).

The Tournament Manager wizard enables to choose both types of round-robin formats, single and double. Groups can also be formed by specifying the parameter `minNrOfPlayersPerGroup`, which would split entries into balanced groups composed of at least `minNrOfPlayersPerGroup` participants. The default behavior of the tournament distribution consists of maximizing the number of groups (and minimizing the number of participants per group).

The parameter `nrOfQualifiedPlayers` determines the number of players that advance to the next stage of the tournament, who are taken from the top final standings. If participants are divided into groups, the selection of the qualified contestants is firstly by position within the group, and when the number of remaining qualifying places is less than the number of groups, the participants placed in the current position are sorted between them.

A constraint imposed by this format is that it only supports matches of two participants. Therefore, games that require matches of more than two players (or teams) will not be able to use it.

### 5.4.2 Swiss-system

A Swiss-system tournament is a non-elimination format like round-robin, which requires considerably less rounds (set by the organizer) to determine the winner since participants do not necessarily play all the others. Competitors are paired according to a set of rules designed to ensure that each contestant plays against opponents with a homogeneous skill level, but not the same opponent more than once. The winner is the competitor with the highest aggregate points earned in all matches.

This format aims to be applied in competitions where the number of participants is considered infeasible for a round-robin, and the elimination before the end of the tournament is undesirable. Early eliminations may result in the best possible competitor being cut out of the tournament due to a one-time mistake, as even good competitors can have a bad match. Hence, Swiss-system claims to provide a more legitimate winner than knockout tournaments with a smaller number of rounds than a round-robin.

In the first round, contestants are either seeded according to some prior order (e.g., rating or previous performance) or randomly. Then, the sorted list of all participants is split into

two halves, and participants are paired based on their position within the half (i.e., the top competitor of the first half plays against the top competitor of the second half, the second against the second, and so on). After the first round, participants are sorted according to the cumulative results of the previous rounds and divided into groups of identical scores. To each of these groups, it is applied the same method as in the first round, except for matches between competitors who already played against each other (e.g., if the second competitor of the first half already played against the second competitor of the second half, it checks the third, and so on). When groups have an odd number of entries, either the highest ranked contestant can be moved up one group or the lowest ranked moved down to balance the groups. In cases where the total number of competitors is odd, one of them gets a bye alternatively in each round, which means that the competitor awards the points of a win without playing in the round. During the whole tournament, a player can only have one bye.

The wizard enables the organizer to define the maximum number of rounds of the Swiss-system, through the parameter `maxNrOfRounds`. At the end of these rounds, the top `nrOfQualifiedPlayers` participants in the final standings advance to the next stage. This format can also be applied to multiple groups separately, each with `maxNrOfRounds` rounds, by setting the parameter `minNrOfPlayersPerGroup` to a value greater than 1. Then, the selection of the qualified contestants is firstly by position within the group, and when the number of remaining qualifying places is less than the number of groups, the participants placed in the current position are sorted between them.

As in round-robin, this format only supports matches of two participants, which may not be adequate for some games.

### 5.4.3 Single Elimination

The single elimination tournament is one of the most simple tournament formats. Initially, players are seeded either randomly or following some prior order. Then, the sorted list of participants is split into groups, and participants form matches according to their position in the group (i.e., if the game requires three players per match, there would be three groups, and the first match would be composed of the top-ranked competitor of the first, second, and third groups). At the end each round, losers are eliminated whereas winners continue on to the next round. The matches of each subsequent round are either composed of the winners of consecutive matches of the previous rounds or drawn randomly.

This format is best suited when the number of entries is large, time is short, and the amount of resources for executing matches is limited since it requires the fewest matches to obtain a winner. However, at least half of the remaining participants are eliminated in each round which can have a significant influence on the way that competitors face the match and be very frustrating for losers. Moreover, when games can have draws, tiebreakers or match repetitions must be considered to select a single winner. If the number of competitors does not allow to form matches evenly, some of the competitors may have a bye, which means that they make it to

the next round without playing (only one bye is allowed per competitor).

There is a wide variety of extensions to this format such as the inclusion of a consolidation match for losers of the semi-finals, possibility of having more than one competitor per match progressing to the next round (sometimes happens in shootout poker tournaments), or the double elimination format (described in Subsection 5.4.4). All of them aim to alleviate the penalty of a loss.

The implementation of this format enables the organizer to define three parameters: `nrOfPlayersPerMatch`, `nrOfQualifiedPlayers`, and `maxNrOfRounds`. The former defines the ideal number of players participating in a match. The `nrOfQualifiedPlayers` determines how many players qualify to the next stage, which is 1 by default. This can reduce the number of necessary rounds, since the tournament can be stopped when there are only `nrOfQualifiedPlayers` remaining. The parameter `maxNrOfRounds` works in a similar way as in the other formats, that is it removes all rounds after `maxNrOfRounds` and obtains the final standings from the previous round results.

#### 5.4.4 Double Elimination

There are at least two issues with single elimination. The first is that the most skilled participants can be eliminated if they make just a single mistake. The second is that typically half of the competitors make only one match during the whole competition, which may cause a considerable loss of interest. Double elimination format extends the single elimination to mitigate these problems by introducing a losers bracket. This bracket accommodates the losers of the main bracket and gives them a second chance to achieve the final, also following a single elimination format. This ensures that all entries make at least two matches and the best participants can still achieve the final after a single loss.

Nevertheless, this format also has several weaknesses. One of them is that it requires at least twice the number of matches of a single elimination, since each participant has to lose twice and the tournament only ends when a single participant remains. For instance, if there are  $n$  competitors there will be either  $2n - 2$  or  $2n - 1$  matches, depending on whether or not the winner was undefeated during the tournament. Due to this rule of repeating the final when the winner of the winners' bracket loses, it is unknown until the end of this match whether there will be another match or not, which may cause several organizational problems. Another disadvantage of the double elimination is that it does not guarantee that both finalists have played the same amount of matches, since the winner of the losers' bracket typically has performed more matches.

### 5.5 Asura Viewer

Asura Viewer is the component responsible for showing graphical feedback to learners, both in matches and tournaments. It consists of a **GWT** widget including a separated mode for each

kind of graphical feedback that can be embedded in any GWT-based environment, provided that the JSON data sent to it adheres either to the JSON schema of a tournament (see Section D.2) or a match (see Section D.1).

The match mode is where learners can see how their SAs behaved during the match. It displays the JSON output produced during the evaluation of the submission or validation as a dynamic movie, only distinguishable from a game in the fact that it can be pushed back and forth. The GUI, presented in Figure 5.8, mimics that of a media player including a slider, a play/stop button, buttons to navigate through the current playlist, and a full-screen button in the control toolbar (area no. 2), a box to show the current status (area no. 3), a box to display log messages of the SA (area no. 4), and a canvas where the movie is drawn (area no. 1).

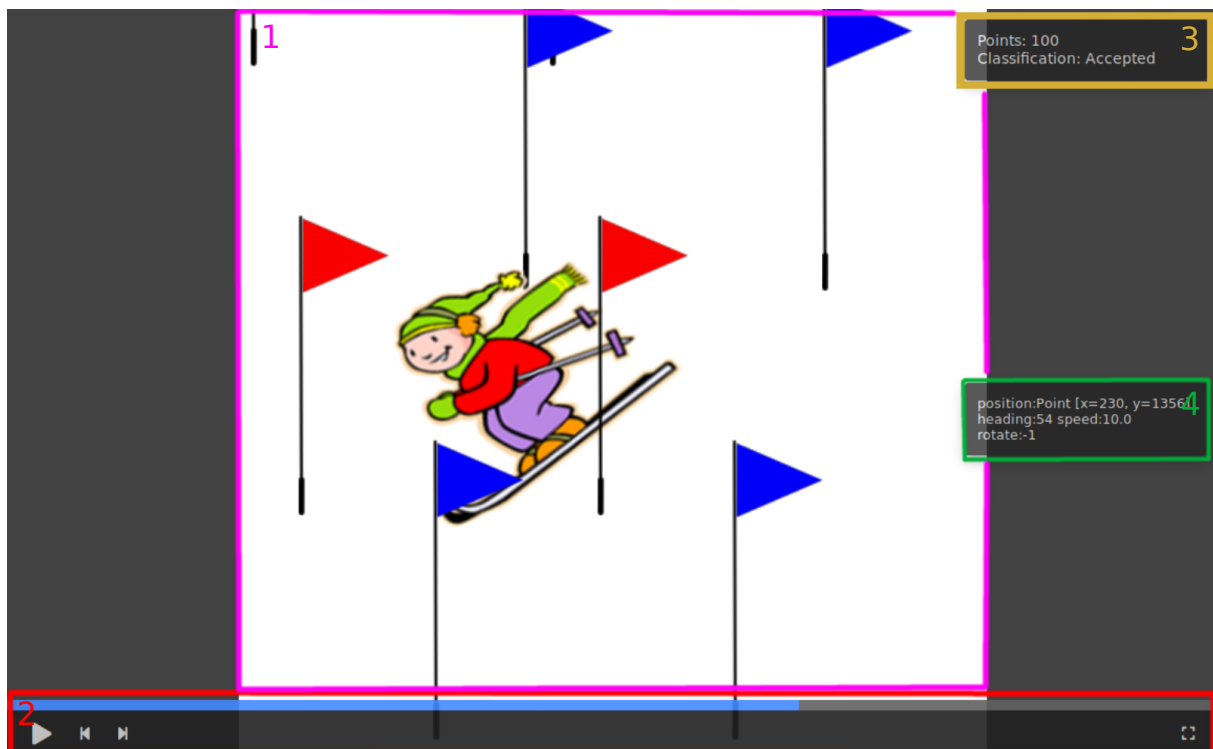


Figure 5.8: Match mode of Asura Viewer exhibiting the graphical feedback provided to a slalom skier game with its main areas highlighted in different colors and numbered

The necessary sprites of the game are included in the header of the JSON movie so that they are loaded only once, before the movie starts playing. In this way, sprites are drawn into and cleared from the canvas as needed without loading them repeatedly, while the movie is playing. In addition to that, the header also includes the title of the game, the original width and height of the movie, an optional background sprite that is drawn at the beginning of each frame before adding other sprites, the number of frames to display per second (property `fps`), and the offset position within the sprites relative to which the position is given in each frame (property `anchor_point`). Frames might contain information on where (`x` and `y`) and how (`rotate` and `scale`) sprites should be displayed as well as their visible window (`view_window`), even though only the positional properties are required. They can also include intermediate status



data and debugging messages of the **SA**, which are placed apart of the graphics in dedicated boxes.

The movie of the game is always displayed in the same aspect ratio as the original size to optimize its quality, independently of the available size in its parent container. To this end, the position and scale of the sprites are updated according to the current size right before being drawn in the canvas.

The tournament mode consists of an interactive **GUI** which allows students to navigate through the stages of the tournament, visualize specific matches or the whole course of a player, and check the rankings of each stage. The **GUI** comprises a navigation menu on the bottom right corner to swap the current stage, a contextual menu on the top right corner to switch between standings view (either final or relative to the current stage) and brackets view, and the main area where matches and ranking appear. Figure 5.9 presents the brackets view in a knockout stage whereas Figure 5.10 reveals the view of the final standings of the tournament.

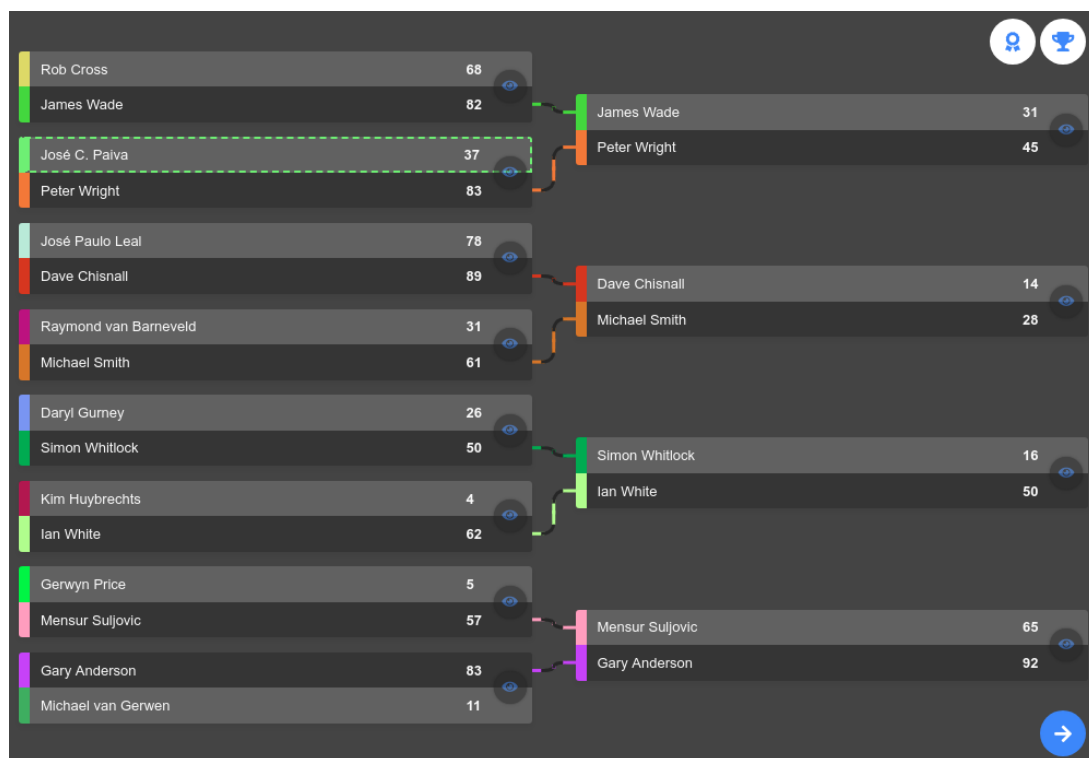
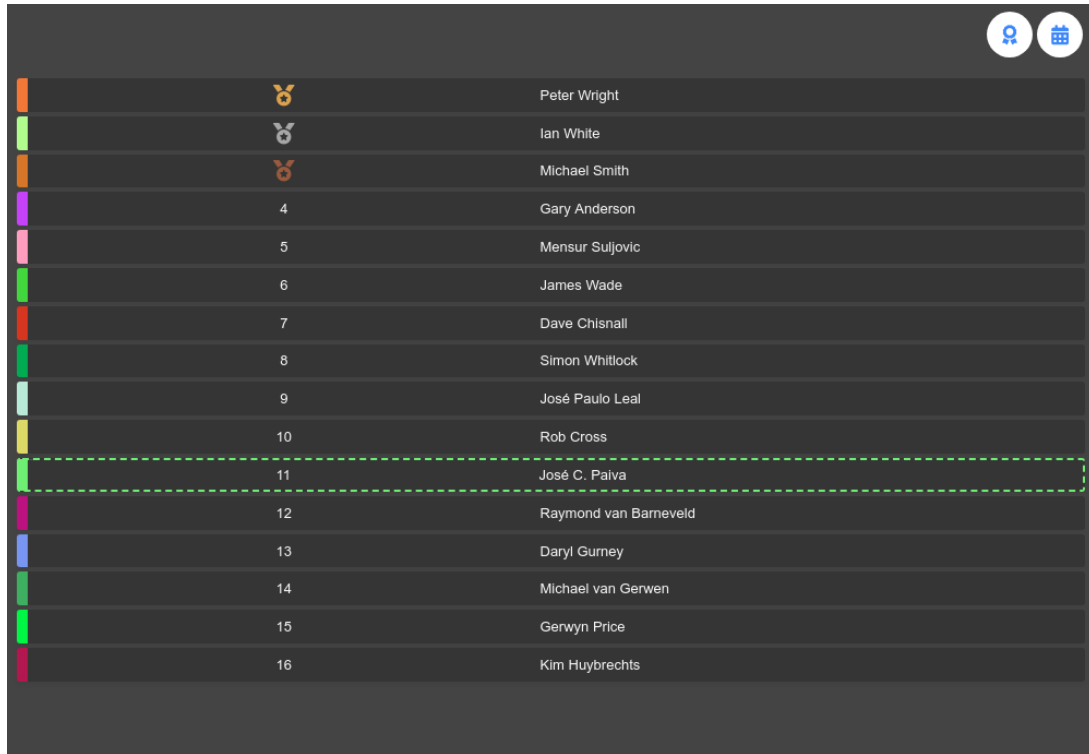


Figure 5.9: Tournament mode of Asura Viewer showing the layout of the matches of a knockout stage

Before rendering the tournament, a random color is assigned to each participant to facilitate their recognition. This color is used on the left border of any element related to a player as well as to fill those elements on hovering. The current player watching the tournament has an additional dashed border around the whole element to easily identify his/her own matches. When clicking a player element, every match of the player in the current stage or in the whole tournament is added to the watchlist, which starts playing automatically.



		Peter Wright
		Ian White
		Michael Smith
4		Gary Anderson
5		Mensur Suljovic
6		James Wade
7		Dave Chisnall
8		Simon Whitlock
9		José Paulo Leal
10		Rob Cross
11		José C. Paiva
12		Raymond van Barneveld
13		Daryl Gurney
14		Michael van Gerwen
15		Gerwyn Price
16		Kim Huybrechts

Figure 5.10: Tournament mode of Asura Viewer presenting the final standings of the tournament

Match elements, either in a group or a knockout stage format, consist of a block element divided into several rows (one row per participant) and a button vertically centered on the right to see that specific match. The competitor's row is composed of the name, obtained score, and, possibly, additional feature values used for tie-breaks.

Asura Viewer has been integrated into Enki's **GUI**, as shown in Figure 5.11. It has been wrapped in its own window and attached to the main region by default. This window is automatically enabled every time that Enki receives a result of a submission or validation, to display graphical feedback on learners' attempts. After receiving the result, Enki (re)activates the match mode and puts the feedback into the widget, which transforms it into visual output, filters messages related to the current participant, and adds any textual observations to the Observations window already existing in Enki.

Once the time to submit new solutions ends, the tournament mode of Asura Viewer is activated and Enki will input the **JSON** data of a tournament into it as soon as one is realized. The tournament data does not contain the **JSON** of each match, but rather a reference to find them on the persistent object database of Mooshak 2. Hence, when the students request the visualization of a match, this request is propagated to Enki, which dispatches it to the server and communicates the response back to the widget.

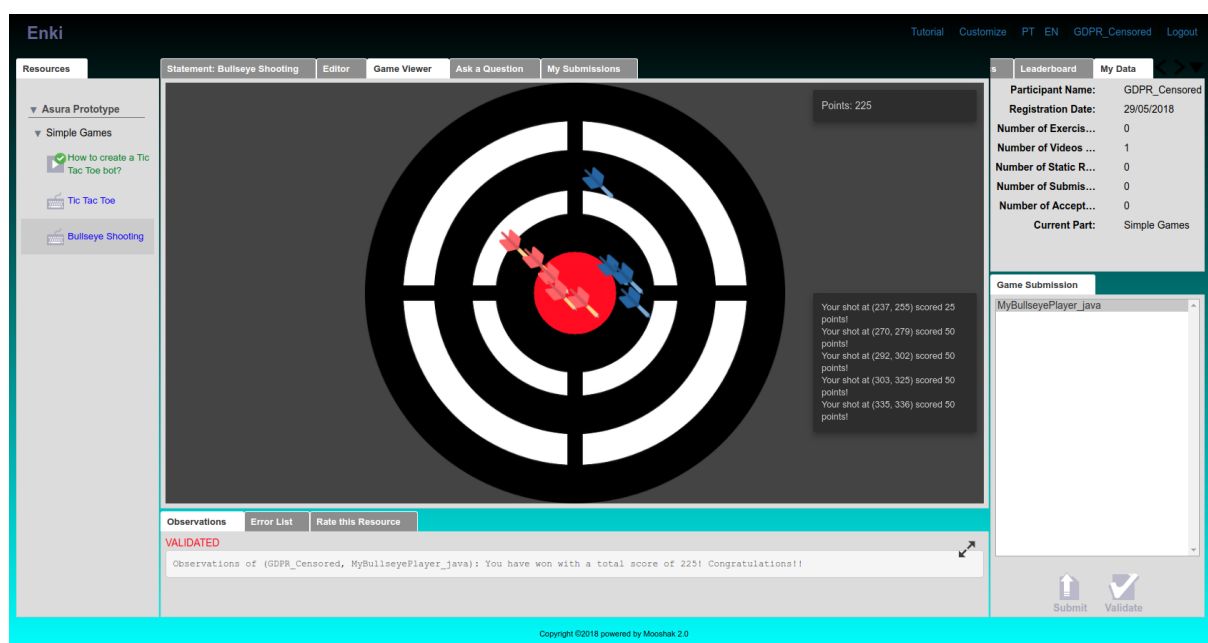


Figure 5.11: Asura Viewer integrated in the GUI of Enki after the validation of the SA in a two-player bullseye shooting game



## Chapter 6

# Validation

Asura aims to mitigate the lack of programming practice in undergraduate students by proposing a solution that focuses on both sides of the programming education. On the one hand, it attempts to foster students' motivation by providing an automated assessment environment for game-based programming challenges that capitalizes on competitive tournaments as a way to retain students' attention. On the other hand, it concentrates efforts on keeping the difficulty of creating this kind of challenges as close to that of creating traditional programming challenges as possible.

Hence, the validation of this environment is twofold, i.e., it consists of two separated experiments. The next sections describe both trials. Section 6.1 details the experiment conducted to measure the user acceptance and efficacy of Asura Builder as a means to reduce the hurdle of authoring Asura challenges to that of creating International Collegiate Programming Contest (ICPC) problems (see Hypothesis 1). Section 6.2 overviews the validation conducted to evaluate the effectiveness of Asura in enhancing students' motivation to practice (see Hypothesis 2).

### 6.1 User Acceptance and Efficacy of Asura Builder

The experiment to validate Hypothesis 1 was conducted in an open environment with undergraduate students of the Department of Computer Science of the Faculty of Sciences of the University of Porto, enrolled either in the *First Degree in Computer Science* or the *Integrated Master of Science in Network Engineering and Information Systems* programs. These students had an average background in Java acquired during the current semester in a Software Architecture course, and had no previous knowledge of Asura.

The test consisted of asking participants to author both an ICPC problem and an Asura challenge, following one or more of the provided statements A, B, or C, in increasing order of difficulty. These statements are presented in Section F.1. Even though they completely describe the challenges to be developed, the quality of the graphical feedback that the game should achieve is not specified. This was left to the creativity of the authors, who can start out from a set of

sprites included in the statement through a hyperlink.

At the date of the experiment, the Command-Line Interface (**CLI**) tool was not finished yet. Thus, a Maven archetype was used instead to generate the project, requiring the authors to configure Maven before starting. The process of scaffolding an Asura challenge using the Maven archetype as well as the installation of the necessary tools to develop the game have been described in a video<sup>1</sup>. A brief documentation of Asura Builder<sup>2</sup> has also been provided.

At the end of the experiment, students were asked to fill-in an online questionnaire, presented in Section G.1, to measure the user acceptance of the Asura Builder and collect usage data. The questionnaire follows Davis' model [55] for evaluating perceived usefulness and ease of use of a system in a 7-value Likert scale, extended with questions about time spent in each type of problem, multiple-choice questions to compare the difficulty of authoring both types (in a 7-value Likert scale), and open text questions to identify weaknesses and strengths, and provide suggestions.

Figure 6.1 presents the results of the user acceptability evaluation of Asura Builder, regarding three components: difficulty of authoring Asura vs **ICPC** exercises, usefulness of Asura Builder, and ease of use of Asura Builder. The overall results indicate a usefulness of 95.24% and ease of use of 76.19%. Regarding the comparison of the difficulty of developing an Asura challenge against that of creating an **ICPC**-like problem, a value of 73.81% was obtained (highest values are better). Nevertheless, statements regarding difficulty and time had a below average score, such as *I'm capable of developing an Asura challenge faster than an **ICPC** problem* (28.57%) and *It is easier to develop an Asura challenge than an **ICPC** problem* (71.42%). When asked to rate, in a Likert scale, the sentence *The additional hurdle of developing an Asura challenge is something that we can disregard considering the gains for students*, students agreed with 42.86%.

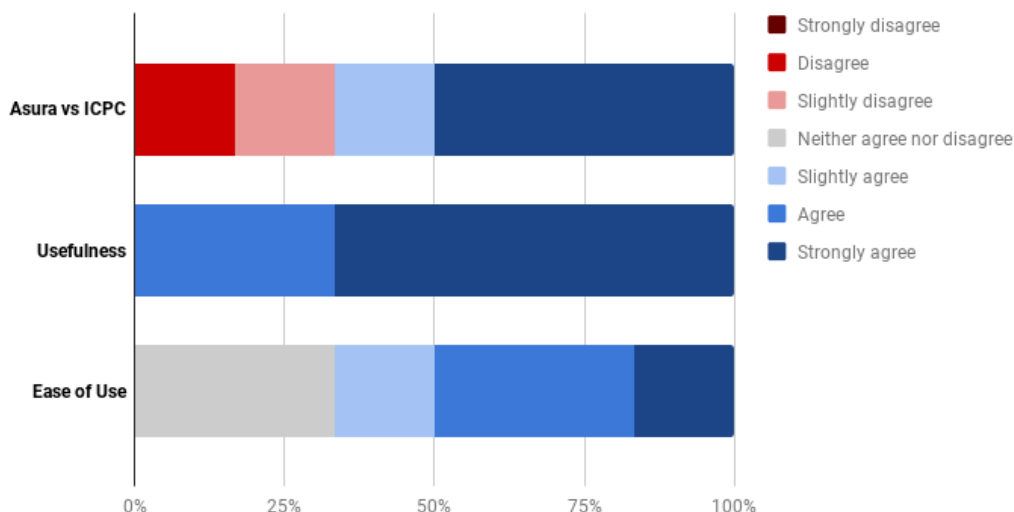


Figure 6.1: Results of the user acceptability evaluation of Asura Builder

<sup>1</sup>[https://www.youtube.com/watch?v=IicbuY\\_WK6M](https://www.youtube.com/watch?v=IicbuY_WK6M)

<sup>2</sup><https://mooshak2.dcc.fc.up.pt/asura/static/asura-builder-documentation.pdf>

The free text answers also highlighted the user acceptance of the Asura Builder. For example, *The Framemork for the development of challenges is very flexible and it's mechanics are easy to understand*. With regard to the weaknesses, a student complained about the quality of the documentation – *Not a real weakness, but documentation on the website should be more user friendly*.

Nevertheless, it was possible to notice some alarming differences between the time spent developing Asura's challenges and creating a similar ICPC problem. For instance, a student spent two hours in the latter whereas he has spent ten hours in the former.

## 6.2 Effectiveness and Usability of Asura

The experiment conducted to test the effectiveness of Asura in enhancing students' motivation and increasing practice time took the form of an open online course entitled "ES6 Course" with a duration of two weeks, free of charge, and without participation limits. The course aims to teach concepts introduced in version EcmaScript (ES) 6 of JavaScript. It has been announced three days before starting via email and social networks (Twitter and Facebook) to undergraduate students of the Department of Computer Science of the Faculty of Sciences of the University of Porto, enrolled either in the *First Degree in Computer Science* or the *Integrated Master of Science in Network Engineering and Information Systems* programs. These students had finished a Web Technologies course during the previous semester that provided them with some knowledge of JavaScript.

A total of 10 students registered in the course, of which 1 was female. These students were divided into two groups, control (4 males and 1 female) and treatment (5 males). Each of the groups made a separate branch of the course, which had the same expository resources but different evaluative resources. The expository resources are lecture notes about the concepts of ES6, including variable declaration, object and array destructuring, arrow functions, promises, and classes, and a compiled ES6 cheat sheet. The evaluative resources are either ICPC problems (control branch) or Asura challenges (treatment branch). At the end of the course, students from both groups were invited to do an exam composed of ICPC problems to assess the knowledge acquired during the course. The statements of each of these exercises can be consulted in Section F.2.

The next subsections provide additional details about the Asura challenges created for the course, the analysis carried out and its results. Subsection 6.2.1 details the Asura challenges created to teach ES6, which may also serve as an informal validation of the Asura Builder capabilities. Subsection 6.2.2 describes the analysis and results.

### 6.2.1 War of Asteroids

The War of Asteroids is based on Asteroids, an arcade space shooter released in November 1979 by Atari, Inc and one of the first major hits of the golden age of arcade games. The objective of Asteroids is to destroy asteroids and enemies (also known as saucers) to award score points. In the game, the player navigates a triangular ship, initially placed at the center, that can turn left and right, fire shots, and thrust forward. Once the ship gets propelled in a direction, it continues in that direction, unless the player applies thrust in a different direction. The graphics consist of simple white lines on a dark background, as shown in Figure 6.2.

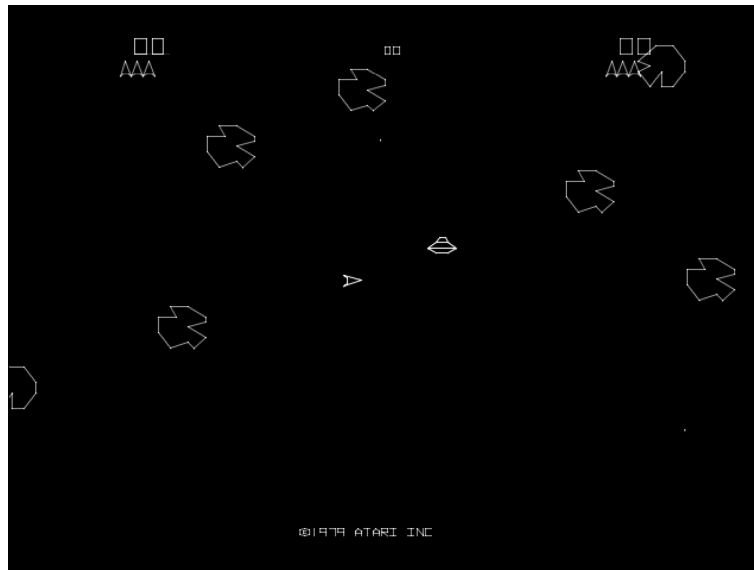


Figure 6.2: Screenshot of the original Asteroids version from Atari

The virtual world of Asteroids is a cyclic grid, which means that objects moving out of the grid disappear and reappear on the opposite location of the screen. The world is populated with asteroids, moving on a certain direction and constant speed, and saucers, which can change direction and speed. Both can only be destroyed with shots. However, large asteroids first break into medium asteroids, which can, in turn, be blasted into small rocks, that can actually be destroyed.

War of Asteroids keeps most of the original gameplay of Asteroids from Atari, but adds a number of features; improves graphics (see Figure 6.3); and replaces the input controls with a program. The game is now played by up to 4 contestants, which replace the saucers. Furthermore, the ships have two additional commands that can be used: activate the energy shield and fire bombs. The game ends after 10000 units of time (frames) or when a player gets alone in the space.

At the beginning, ships are placed randomly close to one of the four corners of the game world headed to the center, and have 100 of energy and health. The state of the ship, accessible through `state()`, consists of its position  $(x, y)$ , energy, health, unitary velocity, heading angle, and remaining time of shield protection (0 if inactive). Each thrust increases the unitary velocity



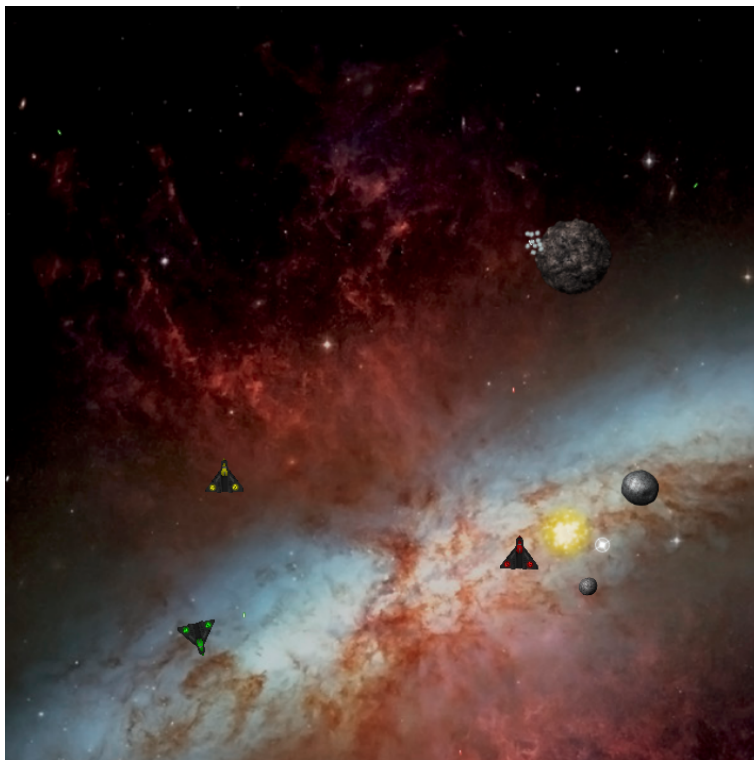


Figure 6.3: Screenshot of the War of Asteroids

of the ship by 0.5, up to a maximum of 5. The minimum interval between thrusts is 5 frames. Once propelled, the ship can lose speed if it collides with asteroids (if the shield activated) or bullets (but it can also gain depending on the trajectory and speed of the collision), in addition to the original mechanics. Steering left/right rotates the heading of the ship by 4 degrees.

The ship has two weapons, hereafter called primary and secondary. The primary weapon sends small laser shots which last for 200 frames. They travel at a speed of 2.5 increased by a fraction of the speed of the ship at the moment of shooting. They damage the health of enemy ships by 10 and blast asteroids into smaller parts. The recharge time of this weapon is 125 frames. The secondary weapon throws bomb, which travel at a speed of 1 increased by a fraction of the speed of the ship at the moment of shooting. They have power to decrease the health of a ship by 40 or destroy any asteroid. However, bombs also decrease the health of actors in a radius of 50 from the explosion (including the thrower), depending on the distance (the greater the distance, the lower the damage). Bombs explode after 500 units of time or after they hit something, and cost 50 of energy.

The ship has a limited amount of energy (100) which can also be used for activating the shield. Each activation of the shield lasts for 20 frames and consumes 0.5 of energy per frame. The shield protects the ship from any outside enemy, but prevents the ship from firing. Every unit of time the ship recharges its energy by 0.1, if the shield is deactivated. Ships are equipped with a radar able to detect the position of objects nearby. It can discover asteroids anywhere in the world, ships within a range of 250, and bullets/bombs within a radius of 50. Bullets fired

from the primary weapon have a sensor to report the result, i.e., if they had hit a target or not, or if they were not even fired because the weapon was locked.

The API defined to control the ship contains 7 commands, particularly, `thrust()` thrusts the ship, `steerLeft()` adds -4 degrees to the heading of the ship, `steerRight()` adds 4 degrees to the heading of the ship, `shield()` activates the shield, `firePrimary()` fires a bullet, `fireSecondary()` throws a bomb, and `log(message)` logs a message. The method `firePrimary()` returns a promise, which resolves with the result (`NO_HIT`, `HIT_ASTEROID`, `HIT_SHIP`, `HIT_SHIELD`, `DESTROYED_ASTEROID`, or `DESTROYED_SHIP`), if the bullet was fired, or rejects if the recharge time has not ended or the shield is activated.

The ways of earning points as well as the number of points awarded per accomplishment were also modified. Players award 2 points per asteroid hit; 5 per ship hit; 4 per destroyed asteroid; 10 per destroyed ship; 4 when they successfully protect from a bullet; 2 when they shoot on a protected ship; 2 when they successfully protect from an asteroid; and 2 per each 5 remaining health points at the end.

The capabilities of Asura Builder allowed building the entire game movie in less than 25 instructions, including explosions, bullets' hits and rotating asteroids animations. The movie is recorded at 60 frames-per-second where 3 minutes weights approximately 5MB without compression and 51KB if compressed with Gzip<sup>3</sup>.

This game has been divided into four chapters, each introducing a new concept of the ES6 and a new feature of the game. The statements of each chapter are presented in Section F.2.2. The first three chapters are only introductory solo missions, whereas the final chapter expects students to create a complete competitive Software Agent (SA) to beat their opponents. Figure 6.4 shows a battle of the final chapter integrated in Enki.

## 6.2.2 Data Analysis and Results

The data gathered during the experiment consists of usage data and questionnaire responses. The usage data is automatically captured by Mooshak 2 into the activity log based on every request sent to the server. This enables the extraction of several metrics such as the number of submissions, number of validations, date and time of activity, and submissions' results. The questionnaire is based on the Lund's model [130], including one section per metric (i.e., *Usefulness*, *Ease of Use*, *Ease of Learning*, and *Satisfaction*) with questions to classify sentences in a 7-value Likert scale and an additional section with free-text questions to collect students' feedback about Asura regarding weaknesses, strengths, and points of improvement. This questionnaire is presented in Section G.2.

The questionnaire is part of the exam to guarantee that only students who complete their course and ask for the exam would fill it in. Two of the students (one from each group) have

---

<sup>3</sup><http://gnu.org/software/gzip/>

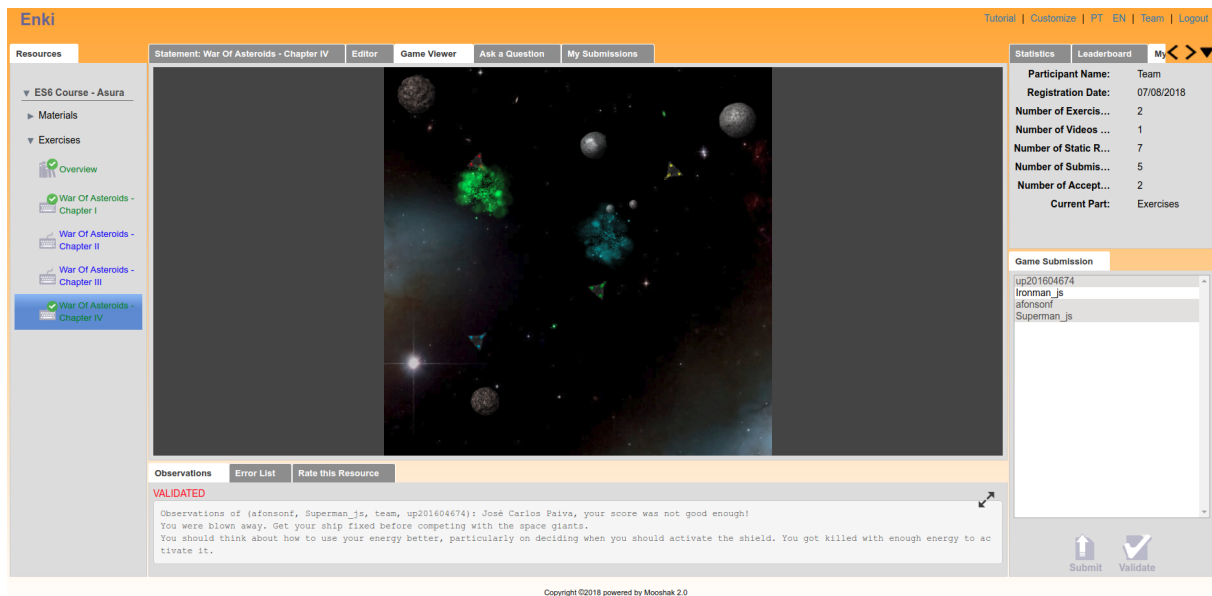


Figure 6.4: Validation of an SA of the War of Asteroids against two students' SAs and a control SA

not taken the exam and, thus, they did not answer the survey. The remaining students have finished both the exam and the questionnaire. The outcome from the questionnaire is presented in Figure 6.5 separated by group, control on the left and treatment on the right.

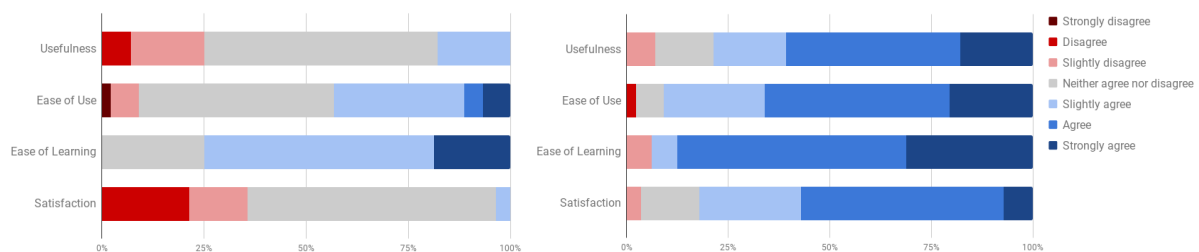


Figure 6.5: Results of the usefulness, ease of use, ease of learning, and satisfaction questionnaire of Asura: the control group (on the left) and the treatment group (on the right)

The results obtained from the questionnaire reveal a great improvement in the four metrics: usefulness, ease of use, ease of learning, and satisfaction. For instance, the usefulness of the control group had an average rating of 3.86 whereas the treatment group was 5.50 and the satisfaction valued 3.46 against 5.43, on the control and treatment groups respectively. From the free-text questions, it is possible to identify the main reason for these differences as being the feedback quality since some students in the control group pointed out feedback as a weakness (e.g., “Observations and program input/output could be improved” and “Feedback messages are scarce”) while learners in treatment enjoyed the graphical feedback and let suggestions to further improve it in the chosen game (e.g., “On the game map, insert the name of the player above the ship” and “Insert a board that displays the remaining energy of the ship”). However, students have complained about the difficulty of getting started with the game and ES6 at the same time (e.g., “The idea is good, but when it is used to learn JavaScript since the start, it

can be confusing because you have to learn JavaScript, learn the game, and program for the 2. You have to think about the two.”). The user interface of Enki has been criticized by students of both groups either because it lacks some features that they were expecting to have (e.g., terminal and debugger) or it looks bad on some devices (e.g., MacBook Air 13”). In regards to strengths, students emphasized the possibility of learning JavaScript through a game like the War of Asteroids (e.g., “Basic game to learn JavaScript, easy to get used to and easy to program against to”).

The analysis of the usage data aims to estimate the real efficacy of Asura in increasing practice time and, possibly, its impact on knowledge acquisition and retention. It is based on 6 variables, including the group (either control or treatment), number of validations, number of submissions, number of different days in which students have made an attempt (also known as return days), ratio of solved exercises in the course (a number between 0 and 1), and score obtained in the exam (an integer between 0 and 3), and a calculated attribute defined by the sum of the number of submissions and the number of validations, the number of attempts. The comparisons of the number of attempts, return days, the ratio of solved exercises, and exam grades between both groups demonstrate improvements in each of these quantitative metrics, as depicted in the boxplots of Figure 6.6.

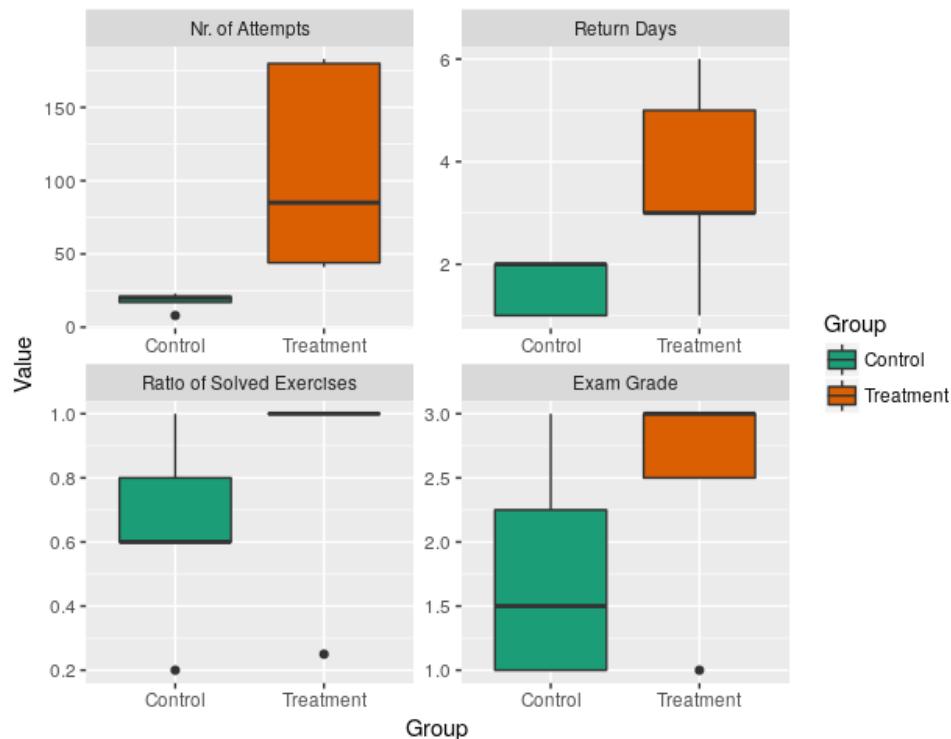


Figure 6.6: Quantitative comparison of metrics per group: number of attempts, return days, ratio of solved exercises, exam grade

The treatment group made 370 submissions of which 65 have been accepted whereas the control group submitted 44 times of which 16 succeeded. Adding this information to the percentage of exercises that were solved and the return days in both groups, it can be concluded

that the students found it difficult to solve both kinds of challenges, but they struggled more in the treatment group than in the control group to overcome their difficulties. Furthermore, students in control only submitted until they solved the exercises while in the treatment they have made 40+ submissions than necessary, all of them in Chapter IV. The amount of validations also reflects these trends, yet the analysis suggests that students did not correctly understand the concept of Asura validations (i.e., friendly matches against the opponents) because they only validated 15 times in Chapter IV. The exam, which has been realized only by 4 learners of each group, shows that 75% of the students in treatment have achieved a good grade (between 2 and 3) and 50% in control. Nevertheless, presumably, the differences are so expressive due to the low number of participants that allowed other factors to be highlighted in the analysis, such as the students' familiarity with Enki, the connection with the instructor, or the personality of the students.

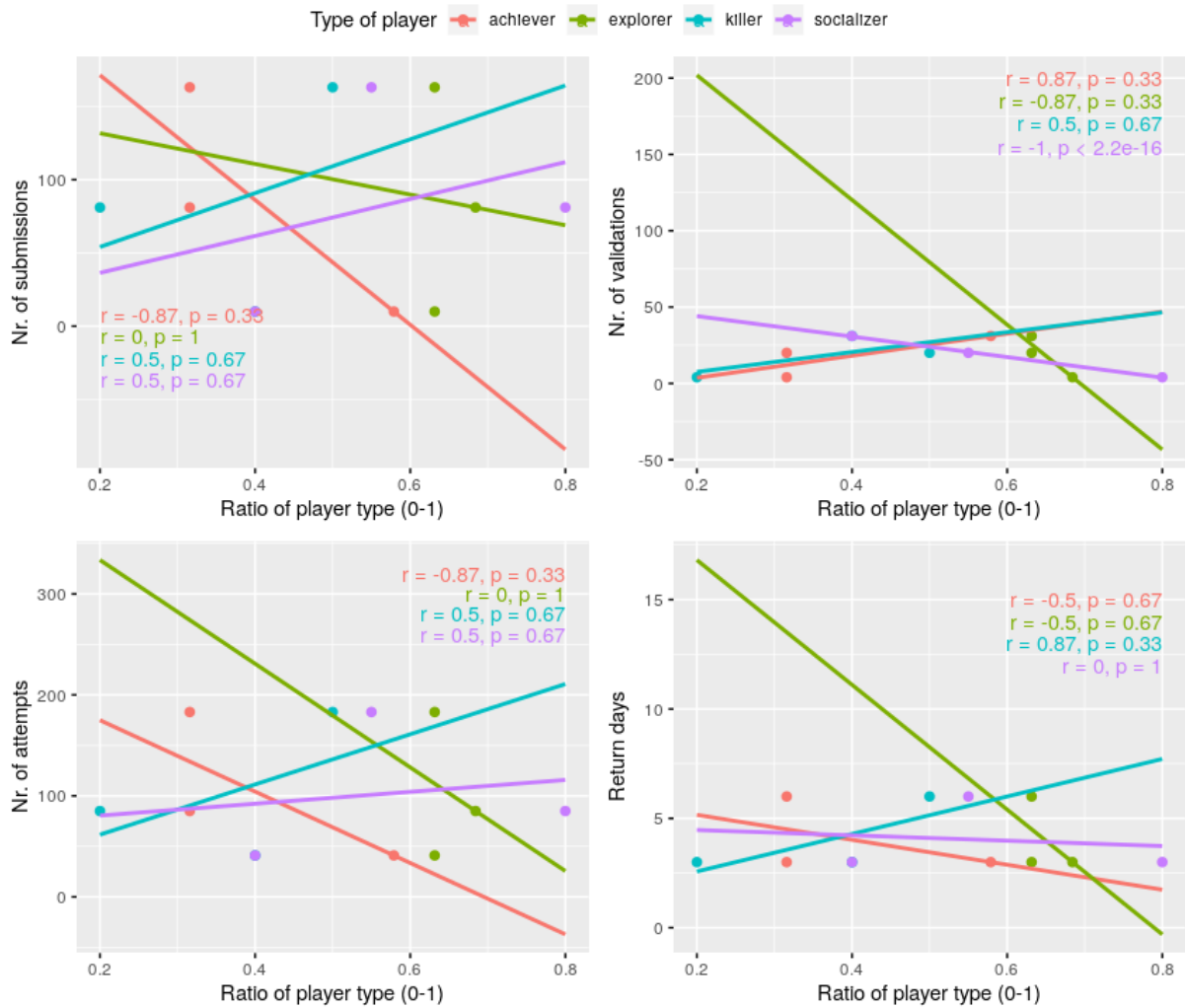


Figure 6.7: Spearman's correlation between the different player types and the number of submissions, number of validations, total attempts, and return days

Students in the treatment group were also invited to fill in a survey (presented in Section G.3) to evaluate their gamer personality. This survey follows the Bartle Test [134] to classify players

of multiplayer online games into categories based on Bartle's taxonomy of player types (i.e., achiever, explorer, killer, and socializer) [14]. It estimates the percentage that the player's behavior relates to the default behavior of each type of player. The main objective of this analysis was to detect possible correlations between the personality of the learner as a gamer and his/her behavior during the experiment. To this end, 4 variables were selected: number of submissions, number of validations, number of attempts, and return days. The resulting plots of Spearman's correlation for the distinct taxonomies are presented in Figure 6.7, annotated with Spearman's correlation coefficient,  $r$ , and  $p$ -value. For instance, it is possible to verify a positive monotonic relationship between the killer's ratio and each of the measured values as well as a negative monotonic relationship between the explorer's ratio and the number of validations and return days. Unfortunately, only 3 learners have completed this test which makes the results statistically insignificant.

## Chapter 7

# Conclusion

The first part of this chapter summarizes the work developed in the context of this thesis as well as its results. Meanwhile, the rest of this chapter presents the next steps to enhance the described work.

### 7.1 Conclusions

The lack of motivation to practice programming outside the classroom is one of the main reasons for failure and dropout in programming courses. Even students who complete their degrees tend to reveal serious flaws in writing computer programs and designing software solutions due to the low programming experience.

Researchers have endeavored to encounter solutions to mitigate these issues. One of the most widely spread and used approaches to foster students' motivation is gamification, i.e., bring games or game elements into learning activities. Nevertheless, not every gamified educational activity engages students nor has positive and longterm effects. Hence, an extensive literature review on gamification has been conducted to find out how games engage players, what effectively motivates students, and how games are being applied in programming learning.

The results indicate that more effective gamification methods typically rely on elements that intrinsically motivate users such as graphical feedback, clear and well-defined goals, and rules. In the programming context, an already tested and successful technique is to challenge learners to code an Software Agent (SA) to play a full-fledged game, and evaluate their knowledge based on the performance of the SA. However, the development of this kind of games, particularly to teach specific concepts, requires a great amount of effort which can hinder their use. Furthermore, there are almost no tools to support educators during this process.

This thesis aims to provide a way to create and assess game-based programming challenges that increase students' motivation and practice time having a similar cost to that of traditional programming exercises. To this end, an automated assessment environment for game-based

programming challenges, named Asura, has been developed. This environment offers teachers a framework and a Command-Line Interface (CLI) to author game-based programming challenges in which learners code SAs to play a game. These challenges introduce a competitive element, in the form of tournaments similar to those realized in games and sports, to further enhance the endeavor to develop better solutions that can beat other opponents' SAs.

A twofold validation has been conducted to evaluate the effectiveness of Asura in reducing the hurdle of creating game-based programming challenges as well as in motivating students and increasing practice time. The data collected during the validation consisted of questionnaire responses and usage data. The outcome of the analysis demonstrated relative success in both hypotheses, but also revealed a large margin of progression.

The experiment conducted to validate the user acceptance and efficacy of Asura Builder, even though with a very low number of participants due to the final exams, has demonstrated its usefulness and ease of use. Nevertheless, the development of game-based programming exercises still takes considerably more time than the creation of International Collegiate Programming Contest (ICPC) problems. This might be slightly alleviated by the use of the CLI tool which was unavailable at the date of the experiment, but further improvements and extra features are required, particularly in the game movie builder. The collected comments have also revealed the need to support multiple programming languages in the creation of games.

The twofold course carried out to test the effectiveness of Asura has proven its effectiveness in motivating a small sample of students to overcome their difficulties through practice. The majority of the students of the control group expressed dissatisfaction with the amount and quality of feedback provided, whereas students of the treatment group have complained about the difficulty of getting started with the specific game, the War of Asteroids, while also learning new concepts of EcmaScript (ES) 6. Both groups pointed out a few issues in the Graphical User Interface (GUI) of the underlying system, Enki.

## 7.2 Future Work

The assessment environment for game-based programming challenges presented in this thesis still has a long way to go in view of its complete acceptance by the academic community as an effective tool in the objectives it proposes. As a result, this section highlights a number of areas in need of attention to achieve this goal.

**Validation in an undergraduate programming course.** The work described in this thesis requires a thorough evaluation in a real undergraduate programming course, as initially designed for the validation of both hypotheses. This validation would require teachers to develop Asura's challenges as well as to create ICPC problems for students to practice programming concepts taught in the course. The students would then be divided into two groups: a control group, in which students would solve ICPC problems, and a



treatment group that would work on Asura challenges. As the course finishes, the usage records would be analyzed in order to observe the discrepancies in the number of delivered submissions and strategies applied. The teachers' feedback on the perceived difficulty of authoring both kinds of exercises would be considered for testing hypothesis H1. This study is actually being designed to be conducted during the semester that began in September 2018, and will culminate in the elaboration of a scientific paper to submit to the ITiCSE'19.

**Enrichment of a repository of Asura challenges.** There is already a vast set of traditional programming exercises developed to practice the various programming concepts taught during the courses. Hence, teachers often choose a subset of previously created problems to use in their lessons, making almost no effort. As Asura proposes a new format of challenges, there is still no repository of exercises that teachers can rely on to apply to their classes, which can hinder their use. For this reason, the enrichment of the repository of Asura challenges with exercises having similar pedagogical purposes to those existing in other formats is of utmost importance.

**Support authoring Asura challenges in different programming languages.** Currently, the Asura Builder framework supports the development of games exclusively in Java. This is a significant lack of the framework since several teachers are unfamiliar with Java and, thus, incapable of creating additional challenges. In order to remove this barrier, two different approaches will be studied. The first is the translation of the framework into two other popular languages such as Python, JavaScript, or C++. This would require to maintain three frameworks, replicating any changes to one of them in the others. A more encouraging approach is the design of a Domain Specific Language (DSL) to define Asura challenges, particularly, the game manager, game state, and wrappers. Nevertheless, the design of such a language needs a thorough and complex investigation.

**Use a lighter format for game movies.** Although the schema for game movies is outlined to be compact and simple, movies with high frame rates can still result in a large file, as it happened with the War of Asteroids where one movie file reached the 5MB. A remedy would be to compress the movie with gzip in the export from the game movie builder and decompress it on the viewer. Further research also needs to be conducted to check whether the use of MessagePack<sup>1</sup> – a binary serialization format that allows to exchange data among multiple languages like JSON – is beneficial.

**Improvement of graphical feedback capabilities.** During the validation, a few students have suggested to insert a text label containing the player's name over the ship on the War of Asteroids' movie. Unfortunately, this feature was left behind in favor of compactness and simplicity when designing the specification for game movies. The advantages of adding and shaping text labels clearly surpass its negative impact on the movie size, so it will be one of the next tasks. Furthermore, the support for scalable vector graphics' paths is also on the list of improvements.

---

<sup>1</sup><https://msgpack.org>



## Appendix A

# Overview of Gamified Learning Systems

In an attempt to understand the current state of the literature regarding gamification in learning environments, a search for experimental studies that apply gamification methods in educational environments was conducted. The search was performed in Google Scholar and ScienceDirect databases, using combinations of the following keywords: gamification, serious games, education, and learning. From the returned results, only studies since 2014 which have reported findings in high school and university subjects of Computer Science/Information Technology (CS/IT), Mathematics (Maths), Foreign Languages (FL), Communication and Multimedia (C&M), Psychology (P), and Medicine/Biology (M/B) were considered. The result was a set of about 50 empirical research studies, of which 21 were selected according to their relevance.

The following tables summarize the characteristics and findings of the selected studies. Table A.1 presents the game elements used in each study. Table A.2 details the experiments conducted to evaluate the impact of the gamification. Finally, Table A.3 summarizes the reported outcomes.

	CS/IT					Maths					FL		C&M				P	M/B			
Game Elements	[191]	[5]	[10]	[21]	[85]	[156]	[186]	[68]	[200]	[44]	[160]	[91]	[162]	[12]	[89]	[94]	[104]	[118]	[163]	[148]	[24]
Badges	✓	✓	✓	✓	✓	✓	✓		✓			✓	✓	✓	✓	✓				✓	
Collaboration	✓			✓			✓		✓				✓	✓	✓	✓			✓	✓	
Competition*							✓														
Content Disclosure / Bonus Content		✓		✓		✓	✓							✓							
Exhaustible Resources								✓			✓	✓		✓	✓		✓				
Leaderboards	✓	✓		✓		✓			✓	✓		✓		✓	✓	✓		✓	✓	✓	
Levels							✓		✓				✓	✓		✓	✓			✓	
Points**	✓	✓	✓	✓	✓	✓	✓	✓	✓		✓	✓	✓	✓	✓	✓	✓		✓	✓	
Progress Indicator	✓	✓		✓								✓		✓	✓	✓	✓				✓
Quests							✓						✓	✓	✓	✓					✓
Real-world Rewards																			✓		
Serious Games / Immersion							✓	✓		✓			✓	✓							✓
SLP***			✓			✓						✓		✓	✓	✓					
Status	✓						✓		✓										✓		

CS/IT – Computer Science/Information Technology; Maths – Mathematics; FL – Foreign Languages; C&M – Communication and Multimedia; P – Psychology; M/B – Medicine/Biology.

\* Only core competition, i.e., one player (or team) plays against other, if one wins, the other loses.

\*\* Includes experience points (XP), skill points (score), influence points (rating, reputation), and automatic grades.

\*\*\* Similar Learning Path (SLP) consists of using data from other/past students to predict and display the current progress of a student (excluding leaderboards).

Table A.1: Game elements used in selected research papers

Research Paper	Learning Subject	Academic Level	Time	Participants	Activity	CBL vs. CuBL	Evaluation Mode
[101]	Introduction Programming	HS	2 weeks	38	Online class activities	CuBL	Q + C/T
[5]	Data Mining	U	1+1 semester*	45 (17F:28M) + 22 (4F:18M)	Support in exercise activities	CuBL	Q + C/T
[10]	Data Structures and Algorithms	U	1+1 semester	254 + 215	Online learning environment		Q + C/T
[21]	3D Modeling	U	2 weeks	55 (24F:31M)	Online class module	CuBL	Pre-test + Q + C/T + Post-test
[85]	Data Structures and Algorithms	U	1 semester	281	Online homework exercises		Q + C/T
[156]	Introduction C#	U	2 weeks	70 (28F:42M)	Open online course	CuBL	Q + Post-test
[186]	Algorithm Design and Analysis + Computer Organization	U	1+1+1+1 semester	190 (79F:111M)	Support in class activities	CBL* + CuBL	C/T
[68]	Quadratic Formula	U	1 class	30	Open classroom quizzes		Pre-test + C/T + Post-test
[200]	Teaching Principles and Methods (Mathematics)	U	1 semester	97 (76F:21M)	Blended Learning (E-Learning and Classroom)		Pre-test + C/T + Post-test
[44]	General Mathematics	U	Single activity	80 (80F)	Single quiz on general mathematics	CuBL	Pre-test + C/T + Post-test
[160]	Differentiation	HS	2 hours	108 (45F:123M)	Single quiz on differentiation		Q + Calculation of evolution
[91]	English	U	Open	27	Open Android application	CuBL	Q
[162]	French	U	50+50+50 minutes	11 (7F:3M)	Open workshop of French (mobile activity)		Q
[12]	Multimedia Content Production	U	130+142+156 days	35+52+54	Online class activities	CuBL	Q + Clustering students by behavior
[89]	Communication	U	1 semester	80 (23F:57M)	Classroom activities	CuBL	Pre-test + C/T + Mid-test + Post-test
[94]	Political Theory + Information Studies	U	1+1 semester	292 (122F:170M) + 231 (101F:130M)	Support class activities	CuBL	Resource usage analysis
[104]	Adobe Photoshop	U + G	Open	114 (40F:74M)	Open online learning		Pre-test + C/T + Post-test + Q
[118]	Psychology research	G	1 semester	86 (65F:21M)	Write wiki page about a topic	CuBL	C/T
[103]	Medical microbiology	U	22 classes	91 (40F:45M)	Powerpoint questions (fast clicker answers)	CuBL	Q
[148]	General medicine	U + G	1 year	128	Online quizzes	CuBL	Resource usage analysis
[24]	Biology (crime-scene)	HS + U	(?) lab classes	149+57+91	Laboratory simulation		Q + Pre-test + C/T + Mid-test + Post-test

CBL – Competitive-based Learning; CuBL – Competition-based Learning. See Section 2.7 for more details on this.

HS – High School; U – Undergraduate; G – Graduate.

C/T – Control/Treatment groups or similar approaches; Q – Questionnaire.

\*Excludes pilot study of 1+1 semester.

\*\*CuBL used to earn extra grade points.

Table A.2: Details of the experiments conducted in selected research papers

Research Paper	Engagement	Participation	Time Dedicated	Learning Outcome
[191]	+	+	+	+
[5]		+		+
[10]	+	+	+	+
[21]	+			+
[85]	+	+	+	~
[156]	+	+	+	
[186]	+	+	+	+
[68]	+			+
[200]	+	+		~
[44]				~
[160]	+			+
[91]	+	+	+	+
[162]	+			
[12]	+	+	+	~
[89]	- (long-term)			-
[94]	+	+	+	+
[104]	+	+	-	+
[118]	+	+	+	+
[163]	+	+		
[148]	+	+	+	+
[24]	+			+

? – The authors supposed that it happen, but did not confirm it.

Table A.3: Reported outcomes in selected research papers

## Appendix B

# Overview of Platforms for Open Online Programming Courses

This chapter summarizes the key features of platforms for learning programming in open online courses which either do not apply gamification or only use extrinsic motivators such as badges, leaderboards, or progress indicators. The reviewed features include the elements of games used, types of learning materials, types of exercises, programming languages supported, whether they provide certifications, and how can courses be created.

Platform	Game Elements	Learning Material	Exercises	Prog. Lang.	Certification	Authoring
<b>Coursera</b>	Progress	Videos, Lecture Notes	Quiz, Programming (I/O tests)	multiple	yes (paid)	requires institution-level affiliation
<b>Udacity</b>	Progress	Videos, Lecture Notes	Quiz, Programming Tasks	multiple	yes (paid)	requires approval
<b>edX</b>	Progress	Videos, Lecture Notes	Quiz, Programming (I/O tests)	multiple	yes (paid)	requires approval
<b>Codecademy</b>	Progress, Badges, Unlock Content	Learn-by-doing	Programming (I/O tests + static checks)	multiple	yes (paid plans)	-
<b>Khan Academy</b>	Progress, Badges	Videos, Learn-by-doing	Quiz, Programming (static checks)	JS, HTML, CSS	no	built-in form
<b>Free Code Camp</b>	Points, Heat Map	Learn-by-doing	Quiz, Programming (static checks)	JS, HTML, CSS	yes	-
<b>Academy LMS</b>	Levels, Badges, Leaderboards, Learning Paths	Videos, Static Content	Quiz and Automated Assessment extensions	-	possible	built-in tool
<b>Moodle</b>	Levels, Badges, Leaderboards	Videos, Static Content	Quiz and Automated Assessment extensions	-	possible	allows
<b>Matrix</b>	Levels, Badges, Leaderboards, Adaptive Learning	Videos, Static Content	Quiz and Automated Assessment extensions	-	-	dedicated environment
<b>P2PU</b>	Badges	Videos, Static Content	Quiz	-	-	markdown, html
<b>Enki</b>	Leaderboards, Progress, Badges, Unlock Content	Videos, Static Content	Quiz, Diagrams, Programming (I/O tests + static checks)	multiple	no	dedicated environment

Table B.1: Overview of the platforms for open online programming courses



## Appendix C

# Asura Implementation Diagrams

This chapter contains additional diagrams of the implementation of Asura. Section C.1 presents the diagram of the Java interfaces of Asura components.

### C.1 Java Interfaces of Asura Components

The components of Asura aim to be independent of the underlying architecture. To this end, they provide common interfaces to their consumers. The diagram of Figure C.1 presents the Java interfaces of Asura Viewer, Asura Evaluator, and Asura Tournament Manager.

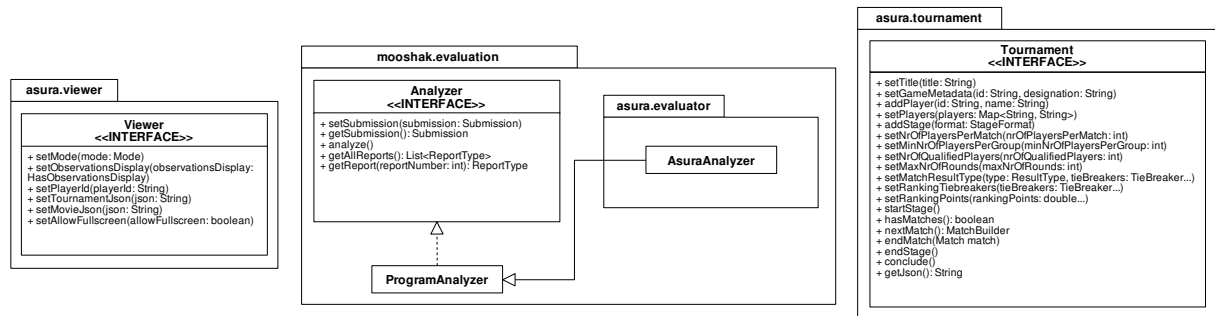


Figure C.1: Java interfaces of Asura Viewer, Asura Evaluator, and Asura Tournament Manager

### C.2 Game Movie Builder

The Game Movie Builder is the part of the Asura Builder framework responsible for the construction of the game movie. It consists of an interface `GameMovieBuilder`, its implementation, and a data model which is directly translated into JavaScript Object Notation (JSON) complying to the match JSON schema (see Section D.1). The interface defines a number of methods to build the movie during the execution of the game such as methods to set metadata properties, add frames, manage the contents of the current frame, push/pop frames from stack, set status of players,

add messages, finish movie due to an exception, and export the movie. The data model contains separate models for the root element – GameMovie, movie metadata – GameMovieHeader, frames – GameMovieFrame, items of a frame – GameFrameItem, view window of an item – GameItemViewWindow, and status of a player – GamePlayerStatus. Figure C.2 shows the Unified Modeling Language (UML) class diagram of the implementation of the game movie builder.

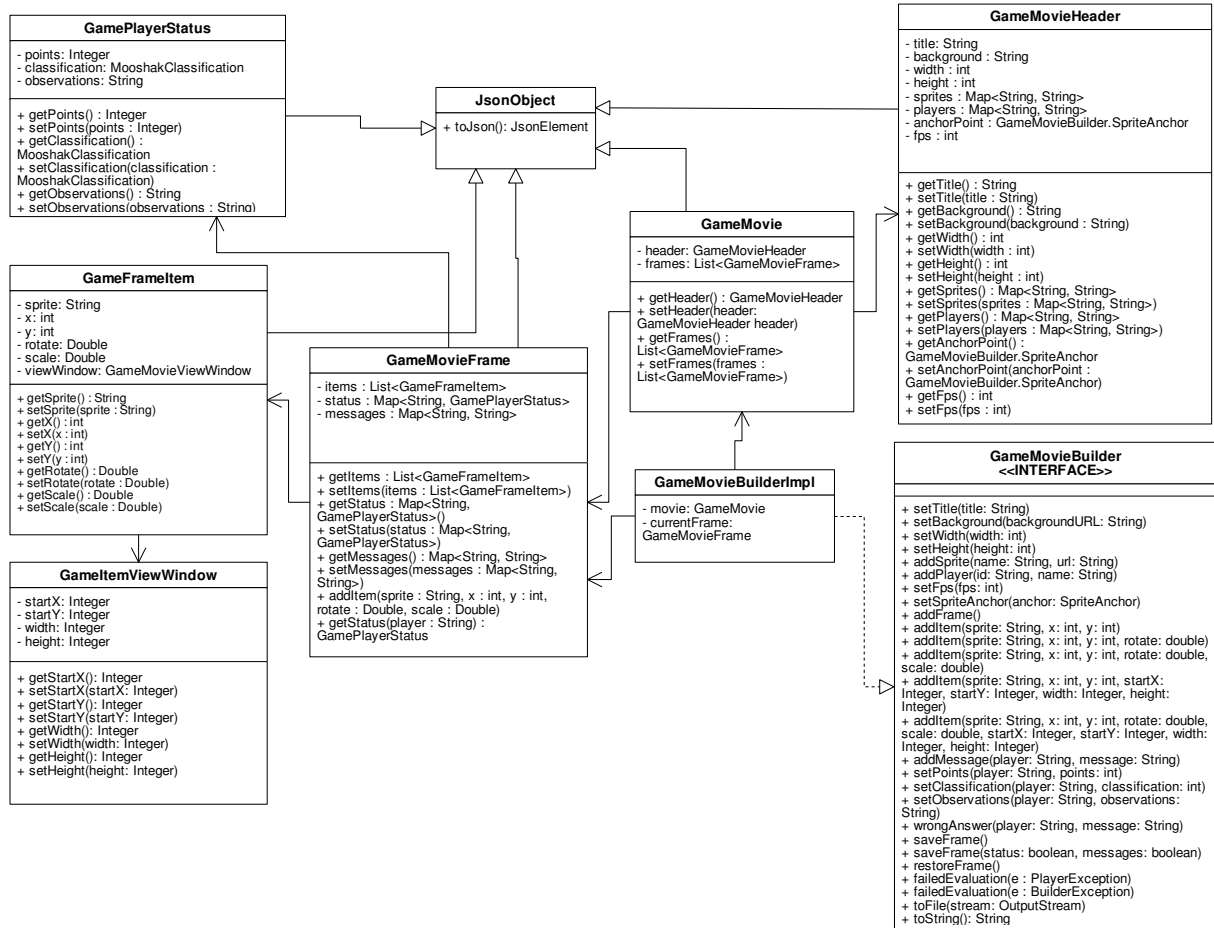


Figure C.2: UML class diagram of the game movie builder

### C.3 Asura Tournament Manager

The Asura Tournament Manager supports the organization of tournaments composed of multiple stages, which can be structured according to a round-robin, Swiss system, single elimination, or double elimination format. The consumers of this component must initially set the tournament metadata and, then, add and execute each stage one after another. The execution of a stage encompasses the query to the component about the next match to run and the update on its result. When Asura Tournament Manager receives the outcome of a match, it automatically updates the rankings according to an analysis of the outcome. Figure C.3 depicts a simplified version of the UML class diagram of the Asura Tournament Manager, omitting properties and

methods of the various tournament formats implemented.

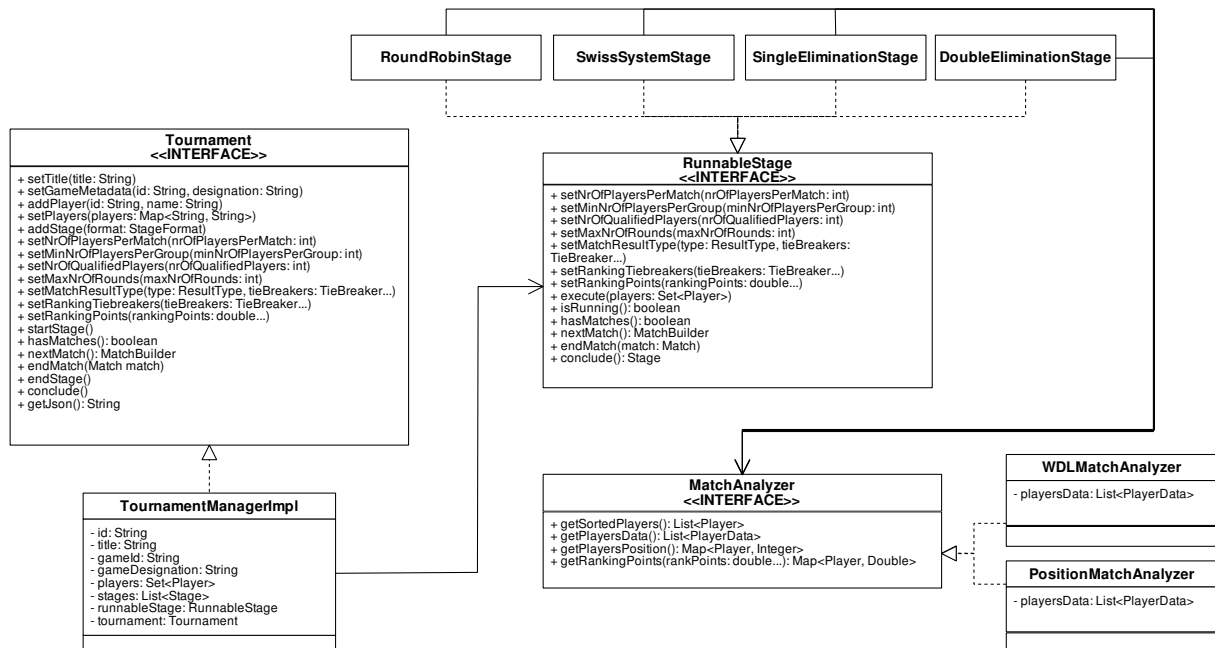


Figure C.3: Simplified UML class diagram of the Asura Tournament Manager

## C.4 Asura Viewer

The Asura Viewer displays both tournaments and matches, allowing its consumers to switch between the different modes and handle a number of events fired by it (e.g., the request of a match by ID). Figure C.4 shows the UML class diagram of the implementation of Asura Viewer.

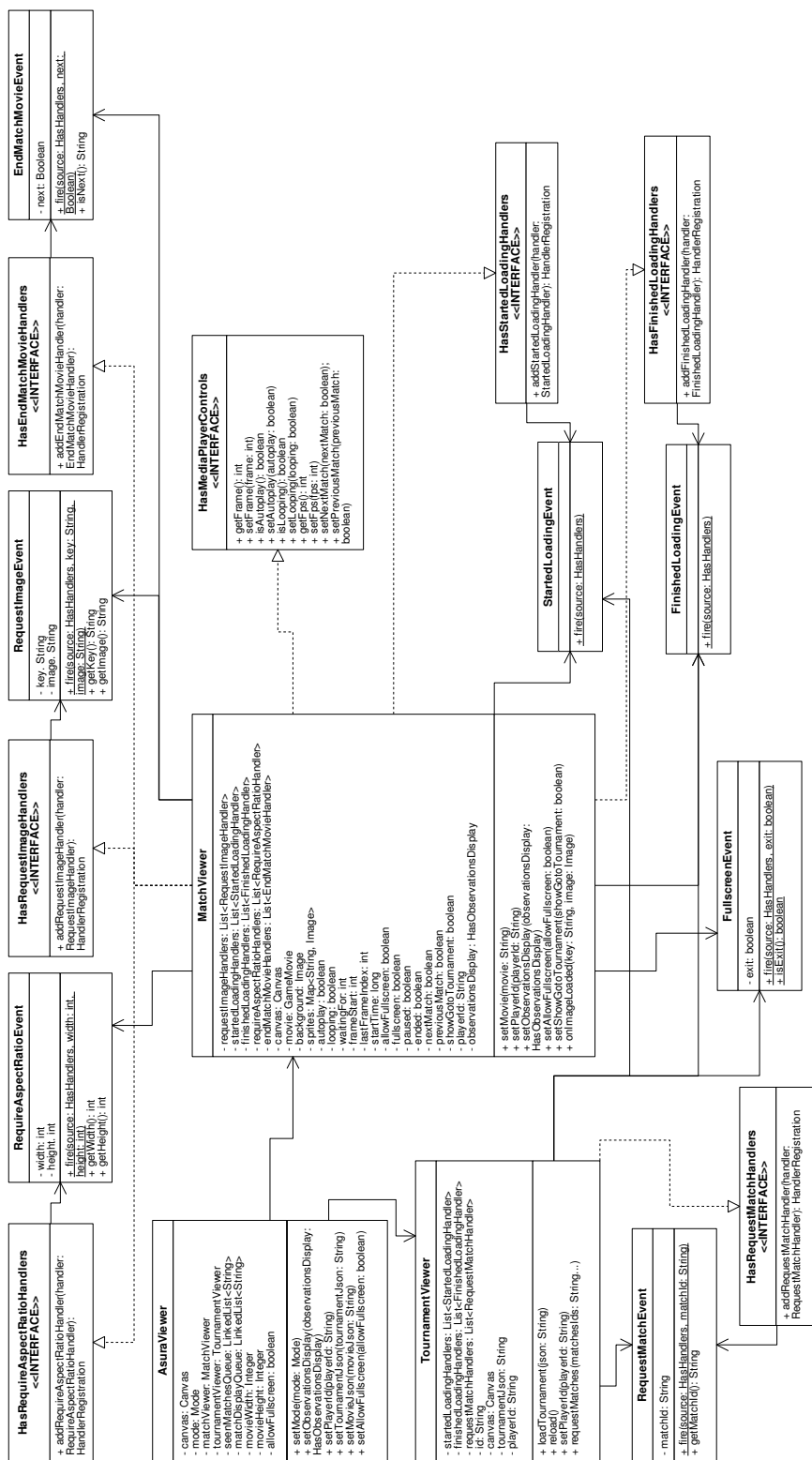


Figure C.4: UML class diagram of the Asura Viewer

## Appendix D

# Object Definitions

During the development of Asura, several formats for exchanging objects between the various components of Asura have been defined. This chapter presents the proposed object definitions. Section D.1 shows the JavaScript Object Notation (JSON) schema defined for match movies. Section D.2 introduces the JSON schema defined for tournaments.

### D.1 Match JSON Schema

Match objects contain the full description of the game movie, including the metadata and frames. Listing D.1 presents the JSON schema defined for matches.

Listing D.1: JSON Schema for matches

```
{
  "$id": "https://mooshak2.dcc.fc.up.pt/match-schema#",
  "$schema": "http://json-schema.org/draft-07/schema#",
  "title": "JSON schema for a Match movie of Asura",
  "description": "This JSON schema describes each frame of a movie of a
    match, containing the metadata, position of each sprite, status
    and messages of the players, etc.",
  "type": "object",
  "required": [
    "header",
    "frames"
  ],
  "definitions": {
    "header": {
      "type": "object",
      "title": "The schema of the header game movie",
      "description": "The header of a game movie contains metadata
        information about the movie, common to all frames.",
      "required": [
```

```

    "title",
    "fps",
    "players"
  ],
  "properties": {
    "title": {
      "type": "string",
      "description": "The title of the game movie."
    },
    "background": {
      "type": "string",
      "description": "An image in Mooshak 2.0."
    },
    "width": {
      "$ref": "#/definitions/positiveInt",
      "description": "The width of the game movie."
    },
    "height": {
      "$ref": "#/definitions/positiveInt",
      "description": "The height of the game movie."
    },
    "fps": {
      "$ref": "#/definitions/positiveInt",
      "description": "The number of frames per second of the game
        movie."
    },
    "players": {
      "type": "object",
      "description": "Map of players in the game indexed by ID.",
      "patternProperties": {
        "^[^\\|/?%*:|\\\"<>\\.]+$": {
          "type": "string",
          "description": "A more readable description of the
            player."
        }
      },
      "additionalProperties": false,
      "minProperties": 1
    },
    "sprites": {
      "type": "object",
      "description": "Map of sprites in the game indexed by ID.",
      "patternProperties": {
        "^[^\\|/?%*:|\\\"<>\\.]+$": {
          "type": "string",
          "description": "An image in Mooshak 2.0."
        }
      }
    }
  }
}

```

```

    },
    "additionalProperties": false
  },
  "anchor_point": {
    "enum": [
      "TOP_LEFT",
      "TOP",
      "TOP_RIGHT",
      "LEFT",
      "CENTER",
      "RIGHT",
      "BOTTOM_LEFT",
      "BOTTOM",
      "BOTTOM_RIGHT"
    ],
    "description": "The anchor point of the sprites.",
    "default": "CENTER"
  }
},
"additionalProperties": false
},
"frame": {
  "type": "object",
  "title": "The schema of a frame of the game movie",
  "description": "A frame of a game movie contains the elements to
    be drawn in the canvas, messages to be displayed, etc.",
  "properties": {
    "items": {
      "type": "array",
      "description": "The items (i.e., elements) of a frame.",
      "items": {
        "allOf": [
          {
            "$ref": "#/definitions/frameItem"
          }
        ]
      }
    }
  },
  "status": {
    "type": "object",
    "description": "The status of a frame of a game movie
      contains the status of each player at that moment.",
    "patternProperties": {
      "^[^\\|/?%*:|\\\"<>\\.]+$": {
        "$ref": "#/definitions/status",
        "description": "A status of a player."
      }
    }
  }
}

```

```

    },
    "additionalProperties": false
  },
  "messages": {
    "type": "object",
    "description": "A frame of a game movie contains the elements
      to be drawn in the canvas, messages to be displayed,
      etc.",
    "patternProperties": {
      "^[^\\|/?%*:|\"<>\\.]+$": {
        "type": "string",
        "description": "A message to a player."
      }
    }
  },
  "additionalProperties": false
}
},
"additionalProperties": false
},
"frameItem": {
  "type": "object",
  "title": "The schema of an item of a frame of a game movie",
  "description": "An item of a frame of a game movie is a sprite
    which will be drawn in a certain position of the canvas with
    a given rotation and scale.",
  "required": [
    "sprite",
    "x",
    "y"
  ],
  "properties": {
    "sprite": {
      "type": "string",
      "description": "An ID of a sprite added in the header"
    },
    "x": {
      "type": "integer",
      "description": "The horizontal position of the sprite in the
        frame."
    },
    "y": {
      "type": "integer",
      "description": "The vertical position of the sprite in the
        frame."
    },
    "rotate": {
      "type": "number",

```



```
    "description": "The rotation of the sprite in the frame."
  },
  "scale": {
    "type": "number",
    "description": "The scale of the sprite in the frame."
  },
  "view_window": {
    "description": "View window of the sprite to display.",
    "$ref": "#/definitions/view_window"
  }
},
"additionalProperties": false
},
"view_window": {
  "type": "object",
  "description": "Window of the sprite to display.",
  "required": [
  ],
  "properties": {
    "start_x": {
      "type": "integer",
      "description": "The x coordinate of the upper-left corner of
        the window."
    },
    "start_y": {
      "type": "integer",
      "description": "The y coordinate of the upper-left corner of
        the window."
    },
    "width": {
      "type": "integer",
      "description": "Width of the window."
    },
    "height": {
      "type": "integer",
      "description": "Height of the window."
    }
  }
},
"status": {
  "type": "object",
  "title": "The schema of the status of a player",
  "description": "The status of a player in a certain moment.",
  "properties": {
    "points": {
      "type": "number",
      "description": "Points of the player."
    }
  }
}
```

```

    },
    "classification": {
      "type": "string",
      "description": "Classification of the player."
    },
    "observations": {
      "type": "string",
      "description": "Observations of the player."
    }
  },
  "additionalProperties": false
},
"positiveInt": {
  "type": "integer",
  "exclusiveMinimum": 0
}
},
"properties": {
  "header": {
    "description": "The header of a game movie.",
    "$ref": "#/definitions/header"
  },
  "frames": {
    "type": "array",
    "description": "The set of frames of a game movie.",
    "items": {
      "allOf": [
        {
          "$ref": "#/definitions/frame"
        }
      ]
    },
    "minItems": 1
  }
},
"additionalProperties": false
}

```

## D.2 Tournament JSON Schema

Tournament objects contain the results of the tournament organized by stages, rounds, and matches. It also contains a reference to each match of the tournament. Listing D.2 presents the JSON schema defined for tournaments.

Listing D.2: JSON Schema for tournaments

```

{
  "$id": "https://mooshak2.dcc.fc.up.pt/tournament-schema#",
  "$schema": "http://json-schema.org/draft-07/schema#",
  "title": "JSON schema for a Tournament of Asura",
  "description":
    "This JSON schema describes a summary of the results of a
    tournament in a structured way with stages, rounds, and
    matches. It also has partial and complete rankings.",
  "type": "object",
  "required": ["metadata", "stages"],
  "definitions": {
    "metadata": {
      "title": "The schema of the metadata of the tournament",
      "description": "Metadata information about a tournament.",
      "type": "object",
      "required": ["game_id", "players"],
      "properties": {
        "title": {
          "description": "The title of the tournament.",
          "type": "string"
        },
        "game_id": {
          "description": "ID of the game of the tournament.",
          "type": "string"
        },
        "game_designation": {
          "description": "Designation of the game of the tournament.",
          "type": "string"
        },
        "players": {
          "type": "object",
          "description": "Players in the tournament indexed by ID.",
          "patternProperties": {
            "^[^\\|/?%*:|\\\"<>\\.]+$": {
              "type": "string",
              "description": "A more readable description of the
              player."
            }
          },
          "additionalProperties": false,
          "minProperties": 1
        }
      },
      "additionalProperties": false
    },
    "stage": {

```

```

"title": "The schema of a stage of the tournament",
"description":
  "Stage is a phase of the tournament which runs under the same
    tournament format",
"type": "object",
"required": ["type", "format", "ranking"],
"properties": {
  "type": {
    "description": "Type of stage.",
    "enum": ["POOL", "KNOCKOUT"]
  },
  "format": {
    "description":
      "Format of the tournament (e.g., Single elimination,
        Round-robin, Swiss system, etc.)",
    "enum": [
      "SINGLE_ROUND_ROBIN",
      "DOUBLE_ROUND_ROBIN",
      "SWISS_SYSTEM",
      "SINGLE_ELIMINATION",
      "DOUBLE_ELIMINATION"
    ]
  },
  "tiebreakers": {
    "type": "object",
    "description":
      "Additional features of a player used for breaking ties in
        the ranking of this stage. These features should be
        present in the outcome of a match, and are accumulated
        during the matches of this stage.",
    "patternProperties": {
      "^[^\\|/?%*:|\\\"<>\\.]+$": {
        "type": "boolean",
        "description": "Are lower values of this feature better?"
      }
    },
    "additionalProperties": false
  },
  "match_tiebreakers": {
    "type": "object",
    "description":
      "Additional features of a player used for breaking ties in
        matches of this stage. These features should be present
        in the outcome of a match.",
    "patternProperties": {
      "^[^\\|/?%*:|\\\"<>\\.]+$": {
        "type": "boolean",

```

```
        "description": "Are lower values of this feature better?"
    },
    "additionalProperties": false
},
"groups": {
    "description": "Groups of a stage group.",
    "type": "array",
    "items": {
        "allOf": [
            {
                "$ref": "#/definitions/group"
            }
        ]
    }
},
"rounds": {
    "description": "Rounds of the stage.",
    "type": "array",
    "items": {
        "allOf": [
            {
                "$ref": "#/definitions/round"
            }
        ]
    }
},
"ranking": {
    "description": "Ranking of the stage.",
    "type": "array",
    "items": {
        "allOf": [
            {
                "$ref": "#/definitions/player_data"
            }
        ]
    },
    "minItems": 1
}
},
"additionalProperties": false
},
"group": {
    "title": "The schema of a group",
    "description":
        "A group is a subset of players in a group stage that can play
        against each other in that stage.",
```

```

"type": "object",
"required": ["rounds", "ranking"],
"properties": {
  "rounds": {
    "description": "The rounds of the group.",
    "type": "array",
    "items": {
      "allOf": [
        {
          "$ref": "#/definitions/round"
        }
      ]
    }
  },
  "ranking": {
    "description": "The ranking of the group.",
    "$ref": "#/definitions/ranking"
  }
},
"round": {
  "title": "The schema of a round",
  "description":
    "A round is a set of matches that can happen at the same time,
    where each player plays at most one of the matches and
    every player should play one match (unless there is no
    possibility of arranging a match with the remaining
    players).",
  "type": "object",
  "required": ["matches"],
  "properties": {
    "matches": {
      "description": "Matches of the round.",
      "type": "array",
      "items": {
        "allOf": [
          {
            "$ref": "#/definitions/match"
          }
        ]
      }
    }
  },
  "byes": {
    "description":
      "Players that do not participate in the round but pass to
      the next.",
    "type": "array",

```

```
    "items": {
      "allOf": [
        {
          "type": "string"
        }
      ]
    }
  },
  "additionalProperties": false
},
"match": {
  "title": "The schema of a match",
  "description":
    "A match is an execution of a game with a set of players. The
    associated movie can be fetched from Mooshak 2.0.",
  "type": "object",
  "required": ["id", "data"],
  "properties": {
    "id": {
      "description": "ID of the match in Mooshak 2.0 to fetch the
        movie.",
      "type": "string"
    },
    "data": {
      "description": "Information about players' statistics during
        the match.",
      "type": "array",
      "items": {
        "allOf": [
          {
            "$ref": "#/definitions/player_data"
          }
        ]
      },
      "minItems": 1
    }
  }
},
"ranking": {
  "title": "The schema of a ranking",
  "description": "A ranking of a match, stage or tournament. ",
  "type": "array",
  "items": {
    "allOf": [
      {
        "$ref": "#/definitions/player_data"
      }
    ]
  }
}
```

```

    }
  ]
}
},
"player_data": {
  "title": "The schema of player data",
  "description": "Information relative to a player in a match,
    group or stage.",
  "type": "object",
  "required": ["player_id", "points"],
  "properties": {
    "player_id": {
      "description": "ID of the player.",
      "type": "string"
    },
    "points": {
      "description": "Points of the player.",
      "type": "number"
    },
    "position": {
      "description": "Position of the player.",
      "type": "integer"
    },
    "qualified": {
      "description": "Player has qualified to the next
        round/stage?",
      "type": "boolean"
    },
    "additional_features": {
      "description": "Additional values obtained during matches
        relative to a player.",
      "type": "object",
      "properties": {
        "^[^\\|/?%*:|\\\"<>\\.]+$": {
          "type": "number",
          "description": "The value of the additional feature. If
            it is a group or stage, this is the accumulated value
            of the feature on all matches of the player."
        },
        "additionalProperties": false
      }
    }
  }
}
},
"properties": {
  "metadata": {

```



```
    "description": "The metadata information of the tournament.",
    "$ref": "#/definitions/metadata"
  },
  "stages": {
    "type": "array",
    "description": "The set of stages of the tournament.",
    "items": {
      "allOf": [
        {
          "$ref": "#/definitions/stage"
        }
      ]
    },
    "minItems": 1
  },
  "ranking": {
    "description": "Global ranking.",
    "type": "array",
    "items": {
      "allOf": [
        {
          "$ref": "#/definitions/player_data"
        }
      ]
    },
    "minItems": 1
  }
}
```



## Appendix E

# Developing a Tic Tac Toe game with Asura Builder

This chapter describes each of the steps involved in the creation of a Tic Tac Toe challenge with Asura Builder, including the complete code of all modified files. This challenge aims to teach the use of arrays in Java to undergraduate students in an *Introduction to Programming* course.

### E.1 Scaffolding Project

The Command-Line Interface (CLI) tool of Asura Builder is currently able to scaffold Java projects for creating Asura challenges following one of three different templates: Maven-based, Gradle-based, and simple Java project. The generation of a new Asura challenge prompts the user to introduce the desired options for configuring the project, as detailed in Listing E.1 for the Tic Tac Toe game.

Listing E.1: Scaffolding a new project with Asura Builder CLI

```
$ asura-cli create maven
author_name [John Doe]: José Carlos Paiva
author_email [johndoe@myorganization.com]: josepaiva94@gmail.com
group_id [pt.up.fc.dcc.asura]:
game_name [Awesome Game]: Tic Tac Toe
game_classname [TicTacToe]:
game_slug [tic-tac-toe]:
game_package_name [tictactoe]:
game_package [pt.up.fc.dcc.asura.tictactoe]:
game_package_dir [pt/up/fc/dcc/asura/tictactoe]:
game_version [0.0.1]:
```

Although the three provided structures differ in a few files, most of their contents are the same to all templates. Hence, the resulting project structure from the scaffolding of Tic Tac Toe

would look like the one presented in Listing E.2.

Listing E.2: General project structure generated by Asura Builder CLI

```

tic-tac-toe/
|-- src/
|   |-- main/
|   |   |-- java/
|   |   |   |-- pt/
|   |   |   |   |-- up/
|   |   |   |   |   |-- fc/
|   |   |   |   |   |   |-- dcc/
|   |   |   |   |   |   |   |-- asura/
|   |   |   |   |   |   |   |   |-- tictactoe/
|   |   |   |   |   |   |   |   |   |-- TicTacToeManager.java
|   |   |   |   |   |   |   |   |   |-- TicTacToeState.java
|   |   |   |   |   |   |   |   |   |-- package-info.java
|   |-- resources/
|   |   |-- tictactoe/
|   |   |   |-- skeletons/
|   |   |   |   |-- java/
|   |   |   |   |   |-- SkeletonTicTacToePlayer.java
|   |   |   |   |   |-- ... more languages ...
|   |   |   |   |-- solutions/
|   |   |   |   |   |-- ... any solutions ...
|   |   |   |   |-- wrappers/
|   |   |   |   |   |-- java/
|   |   |   |   |   |   |-- TicTacToePlayer.java
|   |   |   |   |   |   |-- ... more languages ...
|   |   |-- images/
|   |   |   |-- mooshak.png
|   |   |-- index.html
|   |   |-- log4j.properties
|   |   |-- player.js
|-- test/
|   |-- java/
|   |   |-- pt/
|   |   |   |-- up/
|   |   |   |   |-- fc/
|   |   |   |   |   |-- dcc/
|   |   |   |   |   |   |-- asura/
|   |   |   |   |   |   |   |-- tictactoe/
|   |   |   |   |   |   |   |   |-- TicTacToeManagerTest.java
|   |   |   |   |   |   |   |   |-- TicTacToeStateTest.java

```

## E.2 Game Manager

Firstly, the game manager of the Tic Tac Toe has to decide who takes the Xs and Os, which is done in a random fashion. After that, it queries the SA playing with Xs and the SA playing with the Os alternately and updates the game state with their move, until the end of the match.

Listing E.3: Game Manager class of Tic Tac Toe

```
public class TicTacToeManager extends GameManager {

    @Override
    public String getGameName() {
        return "Tic Tac Toe";
    }

    @Override
    public String getGameStateClassName() {
        return TicTacToeState.class.getCanonicalName();
    }

    @Override
    public int getMaxPlayersPerMatch() {
        return 2;
    }

    @Override
    public int getMinPlayersPerMatch() {
        return 2;
    }

    @Override
    protected void manage(GameState state, Map<String, Process> players)
        throws BuilderException, PlayerException {

        if (players.size() < getMinPlayersPerMatch() || players.size()
            > getMaxPlayersPerMatch())
            throw new BuilderException("Invalid number of players: " +
                players.size());

        try (Streamer streamer = new Streamer(players)) {

            // collect player names
            Map<String, String> playerNames = new HashMap<>();
            for (String player : players.keySet()) {
                String name = getName(streamer.readActionFrom(player));
                playerNames.put(player, name);
            }
        }
    }
}
```

```

        // prepare state
        state.prepare(movieBuilder, getGameName(), playerNames);

        // TODO: run game

        // flip coin
        List<String> playerIds = new ArrayList<>(players.keySet());

        int crosses = (int) Math.round(Math.random());

        PlayerAction xSymbolAction = new PlayerAction();
        xSymbolAction.setCommand(new Command("SYMBOL", "X"));
        state.execute(movieBuilder, playerIds.get(crosses),
            xSymbolAction);

        PlayerAction oSymbolAction = new PlayerAction();
        oSymbolAction.setCommand(new Command("SYMBOL", "O"));
        state.execute(movieBuilder, playerIds.get((crosses + 1) %
            2), oSymbolAction);

        // run
        int turn = crosses;
        while (state.isRunning()) {
            String player = playerIds.get(turn);

            streamer.sendStateUpdateTo(player,
                state.getStateUpdateFor(player));

            state.execute(movieBuilder, player,
                streamer.readActionFrom(player));

            state.endRound(movieBuilder);

            turn = (turn + 1) % 2;
        }

        // END TODO

        // finalize state
        state.finalize(movieBuilder);

    } catch (IOException e) {
        throw new BuilderException(e.getMessage());
    }
}
}

```

## E.3 Game State

The game state of Tic Tac Toe implements the 6 methods defined by the `GameState` interface. The `prepare` method sets the metadata on the game movie builder, particularly the title, the number of frames per second, the default size, the position of the sprite relative to which the coordinates will be, the 3x3 board background, the X and O sprites, and the players. The `execute` handles the possible actions from the player. In this case, it supports an action `PLAY` that contains the position in which the player wants to play and an artificial action `SYMBOL` sent by the game manager to indicate who is X and who is O. The `getStateUpdateFor` returns the update to send to a given player which, in this case, consists of the position where the last symbol has been placed. The `endRound` generates the frame of the current round based on the previous frame, but adding the new symbol. The `isRunning` checks if the number of rounds has exceeded or there is a winning combination, in which case it is not running. The `finalize` checks who is the winner and sets the statuses of the players accordingly.

Listing E.4: Game State class of Tic Tac Toe

```
public class TicTacToeState implements GameState {
    private static final int WIDTH = 600;
    private static final int HEIGHT = 600;
    private static final String SPRITE_BG = "board.png";
    private static final String SPRITE_X = "x.png";
    private static final String SPRITE_O = "o.png";
    private static final int[] POSITIONS = new int[] {0, 210, 415};
    private static final int[][] WINNING_COMBOS = new int[][] {
        new int[] {1, 2, 3}, new int[] {4, 5, 6}, new int[] {7, 8,
            9},
        new int[] {1, 5, 9}, new int[] {3, 5, 7},
    };

    private Map<String, String> players;
    private Map<String, String> playerSymbols = new HashMap<>();

    private char[] board = "          ".toCharArray();
    private int lastPlayed = 0;
    private int round = 0;

    @Override
    public void prepare(GameMovieBuilder movieBuilder, String title,
        Map<String, String> players) {

        this.players = players;

        movieBuilder.setTitle(title);
        movieBuilder.setFps(1);
    }
}
```

```

        movieBuilder.setHeight(HEIGHT);
        movieBuilder.setWidth(WIDTH);
        movieBuilder.setSpriteAnchor(SpriteAnchor.TOP_LEFT);
        movieBuilder.setBackground(SPRITE_BG);
        movieBuilder.addSprite("X", SPRITE_X);
        movieBuilder.addSprite("O", SPRITE_O);

        for (String playerId: players.keySet()) {
            movieBuilder.addPlayer(playerId, players.get(playerId));
        }

        movieBuilder.addFrame();
    }

    @Override
    public void execute(GameMovieBuilder movieBuilder, String playerId,
        PlayerAction action) {

        switch (action.getCommand().getName()) {
            case "SYMBOL":
                String symbol = action.getCommand().getAsString(0);
                if (playerSymbols.containsKey(playerId))
                    throw new PlayerException(playerId,
                        MooshakClassification.WRONG_ANSWER,
                        "Cannot change symbol during game.");
                playerSymbols.put(playerId, symbol);
                break;
            case "PLAY":
                int position = action.getCommand().getAsInt(0);
                if (position < 1 || position > 9)
                    throw new PlayerException(playerId,
                        MooshakClassification.WRONG_ANSWER,
                        "Invalid position.");
                if (board[position] != ' ')
                    throw new PlayerException(playerId,
                        MooshakClassification.WRONG_ANSWER,
                        "Already used position.");
                board[position] = playerSymbols.get(playerId).charAt(0);
                lastPlayed = position;
                break;
        }
    }

    @Override
    public StateUpdate getStateUpdateFor(String player) {
        return new StateUpdate("LAST_PLAYED", lastPlayed);
    }

```



```
@Override
public void endRound(GameMovieBuilder movieBuilder) {
    movieBuilder.addFrame();

    movieBuilder.restoreFrame();

    movieBuilder.addItem(String.valueOf(board[lastPlayed]),
        POSITIONS[(lastPlayed - 1) % 3],
        POSITIONS[(lastPlayed - 1) / 3]);

    movieBuilder.saveFrame(true, false);

    round++;
}

@Override
public boolean isRunning() {

    if (round >= 9)
        return true;

    return !hasWinner();
}

private boolean hasWinner() {

    for (int[] combo: WINNING_COMBOS)
        if (board[combo[0]] != ' ' &&
            board[combo[0]] == board[combo[1]] &&
            board[combo[1]] == board[combo[2]])
            return true;

    return false;
}

@Override
public void finalize(GameMovieBuilder movieBuilder) {

    String winnerSymbol = hasWinner() ?
        String.valueOf(board[lastPlayed]) : null;

    movieBuilder.addFrame();
    movieBuilder.restoreFrame();
    for (String playerId: players.keySet()) {
        if (winnerSymbol == null) {
            movieBuilder.setObservations(playerId, "It's a tie!");
```

```

        movieBuilder.setPoints(playerId, 50);
    } else if
        (winnerSymbol.equals(playerSymbols.get(playerId))) {
        movieBuilder.setObservations(playerId, "WINNER!");
        movieBuilder.setPoints(playerId, 100);
    } else {
        movieBuilder.setObservations(playerId, "You've lost
            : (");
        movieBuilder.setPoints(playerId, 0);
    }
    movieBuilder.setClassification(playerId,
        MooshakClassification.ACCEPTED);
}
}
}

```

## E.4 Game Wrapper

The game wrapper of Tic Tac Toe defines two methods to support Software Agents (SAs), `getLastPlayed(): int[]` which provides the  $x$  and  $y$  position of the last move and `play(x: int, y: int)` that sends the action to the game manager to play in the given position. Furthermore, it also overrides the methods `update` and `run` of the global wrapper. The method `update` stores the last played position when an update is received whereas `run` determines the execution lifecycle of the SAs.

Listing E.5: Java game wrapper of Tic Tac Toe

```

public abstract class TicTacToePlayer extends PlayerWrapper {

    private int[] lastPlayed = null;

    @Override
    public final void update(StateUpdate update) {
        int position = update.getObjectAsInt();
        if (position == 0) return;
        lastPlayed = new int[] { (position - 1) % 3, (position - 1) / 3
        };
    }

    @Override
    public final void run() {

        // game flow
        while (true) {
            readAndUpdate();

```

```
        execute();
        sendAction();
    }
}

// ---- Provide any additional functions below ----

public int[] getLastPlayed() {
    return lastPlayed;
}

public void play(int x, int y) {
    doAction("PLAY", y * 3 + x + 1);
}
}
```

## E.5 Control SA

This section presents an example (see Listing E.6) of what students must achieve to complete the challenge. This SA is also packaged within the Java ARchive (JAR) to compete against students' submissions in order to check if they are working correctly.

Listing E.6: Control SA of Tic Tac Toe

```
public class MyTicTacToePlayer extends TicTacToePlayer {

    private int[][] board;

    @Override
    public String getName() {
        return "José Carlos Paiva";
    }

    @Override
    public void init() {
        board = new int[3][3]; // 0 empty, 1 me, 2 opp
    }

    @Override
    public void execute() {
        if (getLastPlayed() != null)
            board[getLastPlayed()[1]][getLastPlayed()[0]] = 2;

        for (int i = 0; i < 3; i++) {
            for (int j = 0; j < 3; j++) {
```

```
        if (board[i][j] == 0) {  
            play(j, i);  
            board[i][j] = 1;  
            return;  
        }  
    }  
}  
}
```

## Appendix F

# Validation Exercises Statements

The validation of Asura demanded the creation of several problem statements to explain the tasks to the testers. The next sections assemble the statements written for both the validation of the Asura Builder framework and the EcmaScript (**ES**) 6 Course used to validate Asura.

### F.1 Asura Builder

The experiment conducted to validate the usability and efficacy of Asura Builder required testers to create an International Collegiate Programming Contest (**ICPC**) problem as well as a similar Asura challenge. They had to choose one of the three statements including both tasks. These statements have different difficulty levels in regards to implementation, particularly, easy – *A*, medium – *B*, and hard – *C*, which are presented below.

Faculty of Sciences, University of Porto  
Department of Computer Science  
May 2018

**Author: José Carlos Paiva** (*Note: this statement has been adapted from ToPAS'18*)

Email: [up201200272@fc.up.pt](mailto:up201200272@fc.up.pt)

# PROBLEM A

## BEAN GAME

---

### Description

---

#### Introduction: What is the Bean Game?

The Bean Game is played by two players who alternately take a full set of beans from one end of a sequence of bean piles. At first, there are an even number of bean piles, each with a different amount of beans. The game ends when the last pile of beans is removed, and the winner is the player who has accumulated more beans during the game. If both players have drawn the same number of beans, there is a draw.



#### ICPC-like Challenge

Alex and Bela use different strategies. Alex always removes the pile with more beans. Bela's strategy is more elaborated: she starts by removing the pile with fewer beans; in his second move, she chooses the pile with more beans; in his third move, she takes the pile with less beans again; and so on, alternating between the smaller and larger piles. Alex is always the first to play because he's the youngest.

For example, if the starting sequence has six piles, with 6, 2, 3, 1, 12 and 5 beans, in this order, Alex and Bela's moves are as follows:

1. Alex could pull out the pile with 6 beans or the pile with 5 beans. As he always chooses the larger pile, he removes the one with 6 beans.
2. Bela always begins to remove the hill with fewer beans. Since  $2 < 5$ , she removes the pile with 2 beans.
3. Then, Alex removes the pile with 5 beans (because  $5 > 3$ ).
4. In the next move, Bela chooses the larger pile, which has 12 beans.
5. When there are only two piles, Alex removes the one which is composed of 3 beans.
6. Finally, Bela removes the remaining pile, which has only 1 bean.
7. This game is won by Bela, who accumulates 15 beans ( $2 + 12 + 1$ ), since Alex only collects 14 beans ( $6 + 5 + 3$ ).

**Inputs:** The first line has an even integer,  $m$ , which is the number of mounds of beans when the game begins. The second line is composed of  $m$  integers,  $n_1, n_2, \dots, n_m$ , which indicate how many beans exist in each pile of the sequence.

### Restrictions

- $2 \leq m \leq 30$  Number of piles of beans
- $1 \leq n_i \leq 10$  Number of beans in pile  $i$

**Output:** A single line, whose shape depends on the numbers of beans accumulated by Alex and Belle at the end of the game, denoted by A and B, respectively. The line has the form:

- Alex wins with A against B                      if  $A > B$ ;
- Bela wins with B against A                      if  $B > A$ ;
- Alex and Bela draw with A                      if  $A = B$ .

Your task is to develop a program that, given the initial sequence of piles of beans, calculates the number of beans accumulated by each player, indicating if there is a tie or who is the winner. This program should be capable of generating a set of inputs and their respective outputs, and store them in separate files. This would be used to test attempts to solve this problem.

### Asura Challenge

Games are much interesting when we have graphical feedback, competition, and other players to brag about how we are good at it. People solving the previous challenge (to which you have coded the evaluator), will not feel the game excitement, right? So, let's make a real Bean Game.

For that, you will use the Asura Framework. You can refer to [this documentation](#) to get started.

Your task is to code a referee (Game Manager) for the game, which will maintain a Game State. This referee decides who should play at each time (choose a **random order** in this challenge), requiring the action from him and updating him about the state of the game. The Game State object offers methods to update the state of the game and you should use them also to build the graphical feedback of the game using the movie builder.

In this case, the Game Manager would send, at the beginning, a message containing the number of piles, and the amount of beans in each pile

$$m \ n_1 \ n_2 \ \dots \ n_m$$

Then, in each round, the manager sends to the player taking the turn a message containing the position of the last pile removed (the first is -1). The player will answer with the position of the pile that he chooses.

The movie can use any images that you would like. As a reference you can check these [example images](#). To facilitate building the movie and manage the state, you can define a class `BeanPile` that has a set of beans and is capable of drawing them on a frame, given the movie builder as a parameter. Be creative!

The game loop for each player should be done in the wrapper method `run()` while state updates are received in `update(StateUpdate update)`. You can add more methods to the wrapper, such as `choose(int pos)` which sends a message to the manager to remove pile at `pos` and `int getLastPileRemoved()` which returns the last pile removed.

After that, you can make a player like Alex and another like Bela in the solutions folder.

You can ignore wrappers for ES6 and Javascript. To see how your players play against each other update the tag `<commandlineArgs>` of the `pom.xml` with your players. E.g.:

```
<commandlineArgs>bean pt.up.fc.dcc.asura.bean.BeanManager java Alex.java alex java Bela.java
bela</commandlineArgs>
```

## Examples

<https://github.com/mooshak-dcc/asura-games>



Faculty of Sciences, University of Porto  
Department of Computer Science  
May 2018

**Author:** José Carlos Paiva

**Email:** [up201200272@fc.up.pt](mailto:up201200272@fc.up.pt)

# PROBLEM B

## HANGMAN GAME

---

### Description

---

#### Introduction: What is the Hangman Game?

The Hangman Game is played by  $N$  players who alternately say a letter to attempt to complete a word or sentence thought by a third-party. When there is a *miss*, a part of the body of a man is drawn on the gallows. If a player hits a letter of the word or sentence, every space corresponding to that letter is filled in with the letter and he continues playing. The game ends when the word is completed or the body of the man is completely drawn.

We will consider a variant of the game where each *miss* is awarded with -1 points, except the last which gets -3 points. A *hit* awards 1 point for each filled space, except the last which gets 3 points for each filled space.



#### ICPC-like Challenge

Alex, Ben, and Bela use different strategies. Alex always chooses the next letter of the alphabet. Ben always chooses the letter after the next available letter. Bela's strategy is more elaborated: she starts by choosing the vowels in order (a, e, i, o, u); after they have all been taken, she goes through the reversed alphabet (z, y, x, w, ...). Alex is always the first to play because he's the youngest. The next one is Ben, and finally Bela.

For example, if the starting word is hangman, Alex, Ben, and Bela's moves are as follows:

1. Alex would choose a and award 2 points.
2. Alex continues playing. He would now choose b and award -1 points. (*the head is drawn*)
3. Ben would choose d and award -1 points. (*the torso is drawn*)
4. Bela would choose e and award -1 points. (*the left arm is drawn*)
5. Alex would choose c and award -1 points. (*the right arm is drawn*)
6. Ben would choose g and award 1 point.
7. Ben keeps playing. He would choose h (f is available, the next is h) and award 1 point.
8. Ben keeps playing. He would choose i and award -1 point. (*the left leg is drawn*)
9. Bela would choose o and award -3 points. (*the right leg is drawn*)
10. Summing up, Alex awards 0 (2 - 1 - 1) points, Ben 0 (-1 + 1 + 1 - 1), and Bela -4 (-1 - 3).

**Inputs:** The first line has the word or sentence,  $w$ . The second line has the number of attempts,  $m$ . The third line is composed of the sequence of chosen letters, separated by space, during the whole game  $c_1 c_2 \dots c_m$ .

### Restrictions

- $2 \leq |w| \leq 50$  Number of letters in the word / sentence
- $1 \leq m \leq 26$  Number of attempts

**Output:** A single line, whose shape depends on the existence of a winner. Points awarded by Alex, Ben and Bela are denoted by A, B and C, respectively. The line has the form:

- Alex wins with A. Ben made B points. Bela made C points. *if*  $A > B, C$ ;
- Ben wins with B. Alex made A points. Bela made C points. *if*  $B > A, C$ ;
- Bela wins with C. Alex made A points. Ben made B points. *if*  $C > A, B$ ;
- Alex and Ben win with A. Bela made C points. *if*  $A = B > C$ ;
- Alex and Bela win with A. Ben made B points. *if*  $A = C > B$ ;
- Ben and Bela win with B. Alex made C points. *if*  $B = C > A$ ;
- Nobody wins. Alex, Ben, and Bela made A points each. *if*  $B = C = A$ ;

Your task is to develop a program that, given the initial word and sequence of attempts, calculates the points accumulated by each player, indicating if there is a tie or who is the winner. This program should be capable of generating a set of inputs and their respective outputs, and store them in separate files. This would be used to test attempts to solve this problem.

## Asura Challenge

Games are much interesting when we have graphical feedback, competition, and other players to brag about how we are good at it. People solving the previous challenge (to which you have coded the evaluator), will not feel the game excitement, right? So, let's make a real Hangman Game.

For that, you will use the Asura Framework. You can refer to [this documentation](#) to get started.

Your task is to code a referee (Game Manager) for the game, which will maintain a Game State. This referee decides who should play at each time (choose a **random order** in this challenge), requiring the action from him and updating him about the state of the game. The Game State object offers methods to update the state of the game and you should use them also to build the graphical feedback of the game using the movie builder.

In this case, the Game Manager would send, at the beginning, a message containing the number of spaces for the word. E.g.: 7. After that it sends a message to the player who should play with the attempts made until that moment and the word (including empty spaces). For instance, you can send a string with the number of attempts made,  $m$ , the attempts and the word,  $w$ .

$$m \ c_1 \ c_2 \ \dots \ c_m \ w$$

The movie can use any images that you would like. As a reference you can check these [example images](#). Be creative!

The game loop for each player should be done in the wrapper method `run()`, while state updates are received in `update(StateUpdate update)`. You can add more methods to the wrapper, such as `choose(char c)` which sends a message to the manager to try  $c$ , `char[] getWord()` which returns the word, and `char[] getAvailableLetters()` which returns letters not attempted.

After that, you can make a player like Alex, Ben, and Bela in the solutions folder.

You can ignore wrappers for ES6 and Javascript. To see how your players play against each other update the tag `<commandlineArgs>` of the `pom.xml` with your players. E.g.:

```
<commandlineArgs>hangman pt.up.fc.dcc.asura.hangman.HangmanManager java Alex.java alex java
Ben.java ben java Bela.java bela</commandlineArgs>
```

## Examples

<https://github.com/mooshak-dcc/asura-games>

Faculty of Sciences, University of Porto  
Department of Computer Science  
May 2018

**Author:** José Carlos Paiva  
**Email:** [up201200272@fc.up.pt](mailto:up201200272@fc.up.pt)

# PROBLEM C

# CANDY HUNT

---

## Description

---

### Introduction: What is the Candy Hunt?

The Candy Hunt is played in a  $m \times n$  grid by  $p$  players who alternately make a one square move (horizontally or vertically). At first, candies are randomly distributed by the grid as well as the  $p$  players. When a player enters in the position of the candy, he catches and eats the candy. The game ends when the last candy is eaten, and the winner is the player who has eaten more candies during the game (ignore health problems related with eating too much sugar). If two or more players have eaten the same number of candies, there is a draw between them.

### ICPC-like Challenge

Alex, Ben, and Bela will play this game. Alex is shy and he doesn't want to search in the middle of the field. So, he goes from the starting position directly to the nearest wall. From there, he turns to the right and goes all around the field (always in the border) until he reaches the position that he started this circuit. Then, he moves away one square from the wall and repeats. Ben goes to (0,0) and starts scanning the board (all to the right, then one down and all to the left, and so on). Bela's strategy is smarter: she always looks to the nearest candy. If it is eaten, she moves on to the next. Alex is always the first to move because he's the youngest. Then, it is Ben.

**Inputs:** The first line has two integers,  $m$  and  $n$ , which is the width and height of the grid, respectively. For the sake of simplicity. use small boards (maximum 20 x 20). The next  $n$  lines have  $m$  characters each, being

- # for candies

- . for empty cells
- a, b, or c for Alex, Ben, and Bela, respectively. If two or more players are in the same cell, use an asterisk \*. For instance, if you read the board and there is no a, b, or c, but there is an \*, they are all in the same position.

### Restrictions

- $5 \leq m, n \leq 20$  width and height of the grid
- There are never candies at the starting position of the players.
- If two or more players get the same candy at the same time, a mole eats the candy instead of them.

**Output:** For each round (i.e., each player makes a move), print the corresponding board. At the end, a single line, whose shape depends on the existence of a winner. Candies eaten by Alex, Ben and Bela are denoted by A, B and C, respectively. The line has the form:

- Alex wins with A. Ben eats B candies. Bela eats C candies. *if*  $A > B, C$ ;
- Ben wins with B. Alex eats A candies. Bela eats C candies. *if*  $B > A, C$ ;
- Bela wins with C. Alex eats A candies. Ben eats B candies. *if*  $C > A, B$ ;
- Alex and Ben win with A. Bella eats C candies. *if*  $A = B > C$ ;
- Alex and Bela win with A. Ben eats B candies. *if*  $A = C > B$ ;
- Ben and Bela win with B. Alex eats C candies. *if*  $B = C > A$ ;
- Nobody wins. Alex, Ben and Bela eat A candies each. *if*  $B = C = A$ ;

Your task is to develop a program that, given the initial grid, calculates the number of candies eaten by each player, indicating if there is a tie or who is the winner, and printing the board in each round. This program should be capable of generating a set of inputs and their respective outputs, and store them in separate files. This would be used to test attempts to solve this problem.

### Asura Challenge

Games are much interesting when we have graphical feedback, competition, and other players to brag about how we are good at it. People solving the previous challenge (to which you have coded the evaluator), will not feel the game excitement, right? So, let's make a real Candy Hunt.

For that, you will use the Asura Framework. You can refer to [this documentation](#) to get started.

Your task is to code a referee (Game Manager) for the game, which will maintain a Game State. This referee decides who should play at each time, requiring the action from him and updating him about the state of the game. The Game State object offers methods to update the state of the game and you should use them also to build the graphical feedback of the game using the movie builder.

In this case, the Game Manager could send, at the beginning, a message containing the  $m$ ,  $n$ , and the grid  $g$  in a single line

$$m\ n\ g_{1,1}\ g_{1,2}\ \dots\ g_{m,n}$$

Then, in each round, the manager sends to the players the updated grid. The players will answer with the direction they want to move U (up), R (right), D (down), L (left)

The movie can use any images that you would like. As a reference you can check these [example images](#). To facilitate building the movie and manage the state, you can define a method `printBoard()` that draws a frame according to the received board. To calculate the size of each square you can divide the movie height by  $n$  and the width by  $m$ . Be creative!

The game loop for each player should be done in the wrapper method `run()` whereas state updates are received in `update(StateUpdate update)`. You can add more methods to the wrapper, such as `move(char dir)` which sends a message to move the player in direction `dir` and `char getBoardAt(int x, int y)` which returns the object at position  $(x, y)$ .

After that, you can make a player like Alex, Ben, and Bela in the solutions folder.

You can ignore wrappers for ES6 and Javascript. To see how your players play against each other update the tag `<commandlineArgs>` of the `pom.xml` with your players. E.g.:

```
<commandlineArgs>candyhunt pt.up.fc.dcc.asura.candyhunt.Candy HuntManager java Alex.java alex
java Ben.java ben java Bela.java bela</commandlineArgs>
```

## Examples

<https://github.com/mooshak-dcc/asura-games>

## F.2 ES6 Course

The **ES6** course created to validate Asura in an open online environment consists of two separate branches, one for the control group and another for the treatment group. Both course's segments have the same expository learning materials. However, the evaluative resources are distinct. The next subsections present the exercise statements of both branches as well as of the common exam problems.

### F.2.1 ICPC Problems

The **ICPC** branch has 5 **ICPC**-like exercises about variable declaration – *A*, object destructuring – *B*, arrow functions – *C*, promises – *D*, and classes – *E*, respectively. These statements are presented below.

Faculty of Sciences, University of Porto  
Department of Computer Science  
July 2018

**Author:** José Carlos Paiva

**Email:** [up201200272@fc.up.pt](mailto:up201200272@fc.up.pt)

*(This exercise was adapted from Christophe Coenraets ES6 Tutorial)*

# PROBLEM A

## LOANSTAR - PART I

---

### Description

---

#### Introduction

LoanStar is a small company that makes fixed interest loans to individuals. It has a website with a payment calculator developed in JavaScript, in which customers can enter loan amount, loan term (years), and interest rate, to choose the best parameters for them.

They have finally heard about ES6 and they want to change the calculator code to this version.

#### Task

Replace all variable declarations in the following function to use the new keywords `let/const`.

```
var calculateMonthlyPayment = function (principal, years, rate) {  
  if (rate) {  
    var monthlyRate = rate / 12;  
  }  
  var monthlyPayment = principal * monthlyRate /  
    (1 - (Math.pow(1 / (1 + monthlyRate), years * 12)));  
  return monthlyPayment.toFixed(2);  
};
```



Faculty of Sciences, University of Porto  
Department of Computer Science  
July 2018

**Author:** José Carlos Paiva

**Email:** [up201200272@fc.up.pt](mailto:up201200272@fc.up.pt)

*(This exercise was adapted from Christophe Coenraets ES6 Tutorial)*

# PROBLEM B

## LOANSTAR - PART II

---

### Description

---

#### Introduction

The next step consists of changing the function that calculates the values of the amortization table. This function uses a modified version of `calculateMonthlyPayment(principal, years, rate)` which returns `{ monthlyRate, monthlyPayment }`.

#### Task

Write the missing parts on the following code using the new object destructuring syntax.

```
let calculateAmortization = function (principal, years, rate) {  
    // TODO: obtain the monthlyRate and monthlyPayment from  
    calculateMonthlyPayment with variable destructuring  
  
    let balance = principal;  
    let amortization = [];  
    for (let y = 0; y < years; y++) {  
        let interestY = 0;  
        let principalY = 0;  
        for (let m = 0; m < 12; m++) {  
            let interestM = balance * monthlyRate;  
            let principalM = monthlyPayment - interestM;  
            interestY = interestY + interestM;  
            principalY = principalY + principalM;  
        }  
    }  
}
```

```
        balance = balance - principalM;
    }
    // TODO: add principalY, interestY, and balance as an object
    to amortization
}
// TODO: return the monthlyPayment, monthlyRate, and
amortization
};
```

Faculty of Sciences, University of Porto  
Department of Computer Science  
July 2018

**Author:** José Carlos Paiva  
**Email:** [up201200272@fc.up.pt](mailto:up201200272@fc.up.pt)

# PROBLEM C

## HORSE RACES DOT COM

---

### Description

---

#### Introduction

Horse Races Dot Com is a website that aggregates results of several horse races over the year. They have a database with horse names and IDs, jockey name, trainer name, horse weight, and finish time for each race.

Bob has worked on the backend. He has built an API that returns the results of horse races in a list. The owner of the website now wants to display the results sorted by finish time and weight. Therefore, he is searching for a new frontend developer for that.

#### Task

You really need this job as many other developers. As a test for filtering out candidates, he wants you to build a function that takes as input a JSON array containing all races data as returned by Bob. See the format below as a reference:

```
[
  {
    id: "{{race id}}",
    data: [
      {
        horse_number: {{horse number}},
        horse_name:  "{{horse name}}",
        horse_id:    "{{horse id}}",
        jockey:      "{{jockey}}",
        trainer:     "{{trainer}}",
        weight:      {{weight}},
        finish_time: "{{finish time}}",
        race_id:     {{race id}}
      },
      ...
    ]
  }
]
```

```
    },  
    ...  
]
```

Every `{{...}}` is replaced by its respective value. Finish time is a string formatted as `mm.ss.MM` where `mm` are the minutes, `ss` the seconds, and `MM` the milliseconds. If the horse could not enter the race, the finish time is `---`

Write a method `printRanking(races)` that given the JSON array of races, writes the sorted horse IDs, horse name, weight, and finish times for each race. In case of ties, use the **weight** sorted by decreasing value to break them.

### Example

#### Input:

```
[  
  {  
    "id": "2016-409",  
    "data": [  
      {  
        "horse_number": 12,  
        "horse_name": "WIN CHANCE",  
        "horse_id": "P415",  
        "jockey": "M L Yeung",  
        "trainer": "A Lee",  
        "weight": 111,  
        "finish_time": "1.49.25",  
        "race_id": "2016-409"  
      },  
      {  
        "horse_number": 8,  
        "horse_name": "MALAYAN PEARL",  
        "horse_id": "N416",  
        "jockey": "K C Leung",  
        "trainer": "D Cruz",  
        "weight": 127,  
        "finish_time": "1.49.75",  
        "race_id": "2016-409"  
      }  
    ]  
  }  
]
```

```
},
{
  "horse_number":10,
  "horse_name":"HAPPY FRIENDSHIP",
  "horse_id":"S074",
  "jockey":"A Badel",
  "trainer":"D E Ferraris",
  "weight":127,
  "finish_time":"1.49.86",
  "race_id":"2016-409"
},
{
  "horse_number":14,
  "horse_name":"HOLY STAR",
  "horse_id":"T068",
  "jockey":"T H So",
  "trainer":"D J Hall",
  "weight":111,
  "finish_time":"1.50.04",
  "race_id":"2016-409"
},
{
  "horse_number":3,
  "horse_name":"LIGHTNING AND GOLD",
  "horse_id":"P354",
  "jockey":"K K Chiong",
  "trainer":"C H Yip",
  "weight":125,
  "finish_time":"1.55.05",
  "race_id":"2016-409"
},
{
  "horse_number":"NA",
```

```

        "horse_name": "CHOICE EXCHEQUER",
        "horse_id": "P088",
        "jockey": "U Rispoli",
        "trainer": "C H Yip",
        "weight": 133,
        "finish_time": "---",
        "race_id": "2016-409"
    },
    {
        "horse_number": 11,
        "horse_name": "MY FOLKS",
        "horse_id": "T323",
        "jockey": "D Whyte",
        "trainer": "C W Chang",
        "weight": 124,
        "finish_time": "---",
        "race_id": "2016-409"
    }
]
}
]

```

### Output:

```

2016-409
1 P415 WIN CHANCE 111 1.49.25
2 N416 MALAYAN PEARL 127 1.49.75
3 S074 HAPPY FRIENDSHIP 127 1.49.86
4 T068 HOLY STAR 111 1.50.04
5 P354 LIGHTNING AND GOLD 125 1.55.05
6 P088 CHOICE EXCHEQUER 133 ---
7 T323 MY FOLKS 124 ---

```

Faculty of Sciences, University of Porto  
Department of Computer Science  
July 2018

**Author:** José Carlos Paiva

**Email:** [up201200272@fc.up.pt](mailto:up201200272@fc.up.pt)

# PROBLEM D

## MESS, Inc.

---

### Description

---

#### Introduction

Mess, Inc. is a company in which chaos reigns. Nobody knows who is the boss neither to whom he or she should report. At the end of the day, the only one that really worked hard is the coffee machine.

Lisa, the new intern, after working a few days in the mess and not knowing who is her supervisor yet, had the idea of creating an API to store & retrieve the relations between employees of the company. Each employee was assigned an ID and his/her entry in the database has 3 fields: `first_name`, `last_name`, and `boss_id`. `boss_id` is a reference to another person. The API provides a method `getPerson(id)` which can be accessed through the global object `api`.

#### Task

Write a method `printBoss(id)` that given the ID of an employee, prints a message indicating his/her boss, if it exists in the database. Otherwise, it should print “Not Found”.

#### Example

**Input:** 2

**Output:** *Homer Simpson reports to Marge Simpson*

**Input:** 92

**Output:** *Not Found*

Faculty of Sciences, University of Porto  
Department of Computer Science  
July 2018

**Author:** José Carlos Paiva  
**Email:** [up201200272@fc.up.pt](mailto:up201200272@fc.up.pt)

# PROBLEM E

## SHOOTER STORE

---

### Description

---

#### Introduction

Bob's company is developing a multiplayer First-Person Shooter game where players buy weapons before starting each round. The store has a certain amount of each weapon and it catalogs weapons by type: Pistol, Machine Gun, and Sniper. Each type can have special methods and properties. At the beginning, players have a certain amount of coins and no weapons.

Bob asked you to create classes for the store, player, and weapon, as well as for each type of weapon, while he develops the rest.

#### Task

Your class `Weapon` should have a constructor that receives name, type, cost, damage, rate of fire, and rounds in this order. `Pistol`, `MachineGun`, and `Sniper` are specializations of `Weapon` which have a predefined value for type: 'Pistol', 'MachineGun', and 'Sniper' respectively. For now, these classes will not have additional methods.

The class `Player` should have a constructor that receives the name and the amount of coins, in this order. It should also have a method `getCoins()`, which returns the current amount of coins of the player.

Finally, the class `Store` should have an empty constructor and two methods: `addWeapon(weapon)` and `buy(player, weapon_name)`. The first should add a weapon to the store. Be aware that several weapons can be added with the same name. The method `buy` should make the purchase if possible, removing the coins from the player, adding the weapon to the player, and removing the weapon from the store. If there are no weapons available with that name, it should throw an error with the message "Missing weapon: {{weapon\_name}}." without making any changes. If the player does not have enough coins, it



should throw an error with the message “Not enough coins. Player has {{player.coins}}, weapon costs {{weapon.cost}}.” without making any changes.

### F.2.2 Asura Challenges

The Asura branch has 4 Asura challenges, which consist of different chapters of the same game, the War of Asteroids. These chapters are organized in such a way that the learner builds the Software Agent (SA) progressively, while learning both a new game feature and a new ES6 concept (object destructuring, arrow functions, promises, and classes, respectively). The last chapter expects the learner to create a complete competitive SA to win a final tournament, which is the last level of the Bloom's taxonomy [22]. The statement of each chapter is presented below, as well as the full overview of the game.

Faculty of Sciences, University of Porto  
Department of Computer Science  
July 2018

**Author:** José Carlos Paiva

**Email:** [up201200272@fc.up.pt](mailto:up201200272@fc.up.pt)

# OVERVIEW

## WAR OF ASTEROIDS

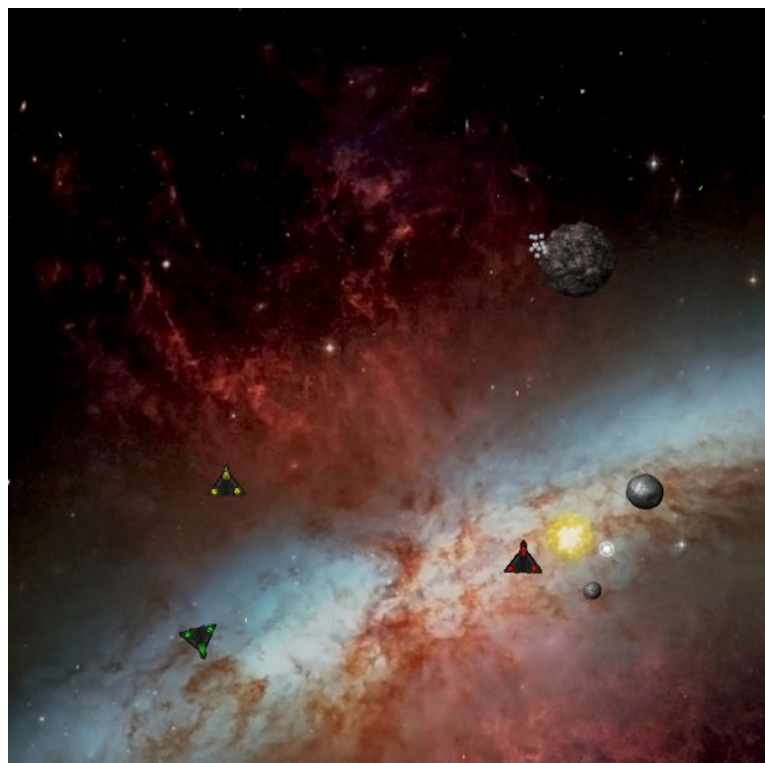
---

### Description

---

#### Introduction: What is the War Of Asteroids?

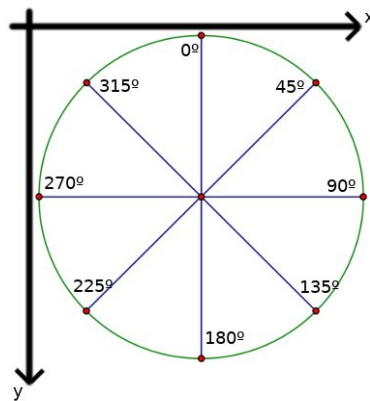
The War Of Asteroids is a game played by up to 4 contestants, each one controlling their own ship flying in the space. Players can thrust, steer left/right, fire, or activate the energy shield of the ship. The goal is to make the highest score among all the players and, of course, stay alive. The game ends after 10000 units of time (frames) or when a player gets alone in the space.



#### Initial State

At the beginning, ships are placed randomly close to one of the four corners of the game world headed to the center. The state of the ship consists of its position (x, y), energy, health, unitary velocity, heading angle, and time of shield (0 if inactive).

The initial state of the ship has a random position, maximum amount of energy (100), maximum amount of health (100), the angle to the center (heading), and 0 time of shield. The game world is a cyclic grid of 600x600, in which the y-coordinate grows down and the x to the right. To better understand the geographic coordinates of the game, please refer to the following image.



## Gameplay

As previously mentioned, players can thrust, steer left/right, fire, or activate the shield of the ship. Each thrust increases the unitary velocity of the ship by 0.5, up to a maximum of 5. The minimum interval between thrusts is 5 frames. Once thrusted, the ship will only lose speed if it collides with asteroids (with the shield activated) or bullets (but it can also gain depending on the trajectory and speed of the collision).

Steering left/right will decrease/increase the heading of the ship by 4 degrees. To go in the direction of the heading, you must thrust after steering.

The ship has two weapons, known as primary and secondary. The primary weapon sends small laser shots which last for 200 frames. They travel at a speed of 2.5 increased by the speed of the ship at the moment of shooting. They have one unit of power, which decreases the health of a ship by 10 or destroys a small asteroid. The recharge time of this weapon is 125.

The secondary weapon throws bombs. Bombs travel at a speed of 1 increased by the speed of the ship at the moment of shooting. They have 4 units of power, which decreases the health of a ship by 40 or destroys any asteroid. However, bombs also decrease the health of actors in a radius of 50 from the explosion (including you). The recharge time of this weapon is 125. Bombs explode after 500 units of time or after they hit something.

The ship has a limited amount of energy which you can use for activating the shield or fire bombs (you spend 50 energy per bomb). Each activation of the shield will last for 20 frames and consumes 0.5 of energy per frame. The shield protects the ship from any outside enemy, but prevents your ship from firing.

Every unit of time the ship recharges its energy by 0.1, if the shield is deactivated.

## Points

The points are awarded according to the following rules:

- 2 per asteroid hit
- 5 per ship hit
- 4 per destroyed asteroid
- 10 per destroyed ship
- 4 when you successfully protect from a bullet
- 2 when you shoot on a protected ship
- 2 when you successfully protect from an asteroid
- 2 per each 5 remaining health points at the end

## The Ship

Ships are equipped with a radar capable of detecting the position of objects around it. It can detect asteroids anywhere in the world, ships within a radius of 250, and bullets/bombs within a radius of 50.

Bullets fired by the primary weapon have a sensor installed to indicate if they have hit a target or not, or if they were not fired.

Enough of theory about the game. Let's see how you actually play the game.

Your ship has two methods:

- `init` which is called when the ship is initialized and allows you to set any initial variables and/or attach handlers to the radar.
- `execute` which is invoked in every frame and allows you to send commands to your ship.

To check the state of the ship you can use the method `state()` of the super class which will provide you with an object containing the current state of the ship, including position (`position.x` and `position.y`), energy (`energy`), health (`health`), unitary velocity (`velocity`), heading angle (`heading`), and time of shield (`shield_counter`).

There are 7 commands that the ship accepts:

- `thrust()` - thrust the ship
- `steerLeft()` - add -4 degrees to the heading of the ship

- `steerRight()` - add 4 degrees to the heading of the ship
- `shield()` - activate the shield
- `firePrimary()` - fire laser (i.e. bullets). This method returns a promise, which resolves with the result, if the bullet was fired, or rejects if the recharge time has not ended or the shield was activated. Result is one of: `NO_HIT`, `HIT_ASTEROID`, `HIT_SHIP`, `HIT_SHIELD`, `DESTROYED_ASTEROID`, `DESTROYED_SHIP`.
- `fireSecondary()` - throw a bomb
- `log(message)` - print debugging messages. message should be string or number. If you need to print objects or arrays, call it like `this.log(JSON.stringify(obj))`;

You can also attach three types of handlers to the radar, for asteroids, bullets, and ships.

- `onAsteroidDetected(handler)`
- `onShipDetected(handler)`
- `onBulletDetected(handler)`

handler is a function that receives a position. Example:

```
this.onAsteroidDetected((asteroid) => {
    const [x, y] = asteroid;
});
```

Faculty of Sciences, University of Porto  
Department of Computer Science  
July 2018

**Author:** José Carlos Paiva  
**Email:** [up201200272@fc.up.pt](mailto:up201200272@fc.up.pt)

# PROBLEM A

## WAR OF ASTEROIDS - Chapter I

---

### Description

---

#### Introduction

Now that you are aware of the rules of the war, I will try to guide you while you become one with your ship.

#### Task

You will go alone with your ship into the space. I want you to understand the state of your war machine first. So, I will send some fake asteroids to the space and you should shoot at the closest asteroid everytime that you fire. Do not shoot randomly or you will not enter in the battlefield. Since your position and heading are unknown, you need to check the state.

Asteroids will be stopped. Don't crash with them.

In this playground, you can use the method `state()` to access the state of your ship and the commands `thrust()`, `steerLeft()`, `steerRight()`, `shield()`, `firePrimary()`, and `log(message)`.

You can also use a special purpose method `asteroids()` which returns the position of the asteroids sorted by distance to you (Note: this method is only available in this playground). These are all methods of your ship, you should access them with `this`.

Rules are simple. 10000 units of time. 4 asteroids. You die, you lose. You fail more than 20% of your shots, you lose.

Hint: remember that ES6 provides Object Destructuring which can be useful for reading the state. Array Destructuring may also facilitate to get the closest asteroid. Last but not least, query google for `Math.atan2()`, it will help you a lot.

Faculty of Sciences, University of Porto  
Department of Computer Science  
July 2018

**Author:** José Carlos Paiva  
**Email:** [up201200272@fc.up.pt](mailto:up201200272@fc.up.pt)

# PROBLEM B

## WAR OF ASTEROIDS - Chapter II

---

### Description

---

#### Introduction

Have you accomplished your last task? Good. Now, you know how to shoot. But protecting your ship from external attacks is, at least, as important as shooting enemies. You shall know how to use your shield.

#### Task

Asteroids are coming after you. You can't move nor fire. Fortunately, you have your radar and full energy to activate the shield. Your only goal is to SURVIVE during 3000 units of time! Oh, forgot to say... your energy is already limited (see the Overview).

Have fun!

In this playground, you can use the method `state()` to access the state of your ship and the commands `shield()` and `log(message)`. You must attach an handler (arrow function) to the radar for getting updates on asteroid positions and act accordingly. For that, use `onAsteroidDetected(handler)` which updates you by calling the handler every time an asteroid moves anywhere in the space. (Note: the radar also supports similar handlers for ships within a range of 250 and bullets/bombs within a range of 50. This can be useful later.)

**Important notes:** asteroids have radius of 8, 16, 24, or 32. Your ship has 20 or 25 (if shield is active).

Hint: in arrow functions context (`this`) has lexical scope. This means that `this` is the `this` from the scope that defines the function, not the context that calls you.



Faculty of Sciences, University of Porto  
Department of Computer Science  
July 2018

**Author:** José Carlos Paiva  
**Email:** [up201200272@fc.up.pt](mailto:up201200272@fc.up.pt)

# PROBLEM C

## WAR OF ASTEROIDS - Chapter III

---

### Description

---

#### Introduction

One way to gather information about the uncovered area of the world is to send bullets and wait its result. If you store the heading, position, and time of your shot, you can make a number of conclusions from its result.

#### Task

There will be three fake ships outside your radar, you should fire and wait the result or until it makes sense to wait. These ships do not move, fire, or activate shield. Your task is to destroy at least one of them, with a hit ratio of 90%. Shots fired before hitting a ship or after destroying a ship do not count as failures. Only shots missed after hitting a ship and before the same ship is destroyed are considered failures.

In this playground, you can use the method `state()` to access the state of your ship and the commands `steerLeft()`, `steerRight()`, `firePrimary()`, and `log(message)`. The method `firePrimary()` returns a promise, which resolves with the result, if the bullet was fired, or rejects if the recharge time has not ended or the shield was activated. Result is one of: `NO_HIT`, `HIT_ASTEROID`, `HIT_SHIP`, `HIT_SHIELD`, `DESTROYED_ASTEROID`, `DESTROYED_SHIP`.

**Important notes:** the lifetime of your bullets is 200.

Hint: use something like `firePrimary().then(...).catch(...)`. If your ... is an arrow function, you can save values like heading, position, and current time right in const declared variables right before firing, and use them inside.

Faculty of Sciences, University of Porto  
Department of Computer Science  
July 2018

**Author:** José Carlos Paiva

**Email:** [up201200272@fc.up.pt](mailto:up201200272@fc.up.pt)

# PROBLEM D

## WAR OF ASTEROIDS - Chapter IV

---

### Description

---

#### Introduction

War is already here! I believe that you are ready to destroy some folks.

#### Task

You will enter in battles of up to 4 real participants. Your only goal is to win by making the best score. Everything that you need to know is in the Overview. Do not use anything that is not there, unless you know what you are doing.

You can define ES6 classes to represent information from other ships, such as the two previously known positions, etc.

While you prepare your ship, you can select real opponents for friendly battles (see the list on the right). By the end of August, a real competition will take place. The last ship submitted by each participant enters in the competition. This competition will consist of a knockout stage with several rounds. Stay tuned!

**Good Game!**

### F.2.3 Exam Problems

At the end of the course, students of both branches were invited to perform an exam containing 3 **ICPC** problems. The statement of each problem is presented below.

Faculty of Sciences, University of Porto  
Department of Computer Science  
August 2018

**Author:** José Carlos Paiva

**Email:** [up201200272@fc.up.pt](mailto:up201200272@fc.up.pt)

# EXAM - PROBLEM A

## CALLBACKS WITHIN LOOPS

---

### Description

---

#### Task

The following function in ES5 contains a common mistake done by many programmers:

```
function generateCallbackArr(n) {  
    var callbacks = [];  
    for (var i = 0; i < n; i++) {  
        callbacks.push(function() { print(i) })  
    }  
    return callbacks;  
}
```

It prints always n, while we were expecting it to print a sequence 0 ... n-1

Fix it by changing only the necessary to ES6.

Faculty of Sciences, University of Porto  
Department of Computer Science  
August 2018

**Author:** José Carlos Paiva

**Email:** [up201200272@fc.up.pt](mailto:up201200272@fc.up.pt)

# EXAM - PROBLEM B

## BOOK DB

---

### Description

---

#### Task

Consider a database with books and authors of books. Each book has several information, namely “isbn”, “title”, “subtitle”, “authors”, “published”, “publisher”, “pages”, “description”, and “website”. The field “authors” stores the IDs of the authors.

Authors have a “name” and a numerical “rating”.

We have defined an API on top of this database to allow to fetch a book by its ID - using `api.getBook(id)` -, as well as an author - `api.getAuthor(id)`. Both functions return a promise which either resolves with the object or fails with a message “No book with id ‘{{id}}’” or “No author with id ‘{{id}}’”.

Your task is to create a function `getBookAuthors(book_id)` which returns a Promise that resolves with an array of objects of the form

```
{  
  title: {{book title}},  
  author: {{author name}},  
  rating: {{author rating}}  
}
```

containing one entry per book author. These entries must be sorted by decreasing rating. If an error occurs in the api just let it go down in the chain.

Note 1: you can make an array of promises, one for each author of the obtained book and use `Promise.all([array])`

Faculty of Sciences, University of Porto  
Department of Computer Science  
August 2018

**Author:** José Carlos Paiva

**Email:** [up201200272@fc.up.pt](mailto:up201200272@fc.up.pt)

# EXAM - PROBLEM C

## CONSTRUCTION

---

### Description

---

#### Task

Create a class `Employee` and add a constructor with `name` as argument. Define a method `toString` which returns a message “employee name: {{name}}, vacation days: {{remaining vacation days}}”. Define another method `takeVacationDays(days)` which subtracts days from a property initially set to 20. If the property becomes negative, the value should remain unchanged.

Create a subclass `Carpenter` and add a constructor with `name` and `type` as argument. Its `toString` method should return the parent class message suffixed with “, job: carpenter, type: {{type}}”.

Create a subclass `Electrician` and add a constructor with `name`. It should have a method `addLicense(license)` which adds a license to the electrician. Its `toString` method should return the parent class message suffixed with “, job: electrician, licenses: {{space separated licenses}}”.





## Appendix G

# Validation Questionnaires

At the end of the experiments conducted to validate Asura, participants were asked to fill-in online questionnaires in order to obtain more information about their experience. Each of the next sections presents one of those questionnaires.

### G.1 User Acceptance and Efficacy of Asura Builder

The data collected for the validation of the Asura Builder is solely based on questionnaire responses. The questionnaire is composed of five sections: *Actual Usage Data*, *ICPC vs Asura*, *Perceived Usefulness*, *Perceived Ease of Use*, and *Comments*.

The first section aims to obtain the average time spent by authors per problem and type of challenge as well as their biggest hurdle in getting started. Section *ICPC vs Asura* contains a set of linear scale questions to estimate the increase in difficulty of developing an Asura challenge when compared to an ICPC challenge. Section *Perceived Usefulness* and Section *Perceived Ease of Use* follow Davis's model [55] for assessing user acceptance of a system based on perceived usefulness and ease of use. However, in this case, the participants have tested an initial version of Asura Builder in a real scenario of authoring game-based challenges. Finally, Section *Comments* includes three free-text questions to collect authors' opinions about strengths and weaknesses of Asura Builder, and suggestions for improvement.

This survey aims to evaluate the perceived usefulness and ease of use of the framework for building Asura challenges. Please be honest with your answers!

## Actual Usage Data

Example: 8.30 a.m.

Example: 8.30 a.m.

Mark only one oval.

- Mark only one oval.

[illegible]

Mark only one oval.

[illegible]

Mark only one oval.

[illegible]

- Mark only one oval.

Mark only one oval.

Mark only one oval.

## Perceived Usefulness

- Mark only one oval.

Mark only one oval.

Mark only one oval.

Mark only one oval.

[illegible]

**14. Using the system would make it easier to develop challenges \***

Mark only one oval.

[illegible]

15. I would find the Asura Builder useful \*

Mark only one oval.

[illegible]

## Perceived Ease of Use

16. Learning to create Asura challenges would be easy for me \*

Mark only one oval.

[illegible]

17. I would find it easy to get Asura Builder to do what I want it to do \*

Mark only one oval.

[illegible]

18. My interaction with Asura Builder would be clear and understandable \*

Mark only one oval.

[illegible]

19. I would find Asura Builder to be flexible to interact with \*

Mark only one oval.

[illegible]

20. It would be easy for me to become skillful at using Asura Builder \*

Mark only one oval.

[illegible]

**21. I would find Asura Builder easy to use \***

*Mark only one oval.*

	1	2	3	4	5	6	7	
Unlikely	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	Likely

## Comments

**22. Weaknesses**

---

---

---

---

---

**23. Strengths**

---

---

---

---

---

**24. Suggestions**

---

---

---

---

---

## G.2 Usability of Asura

The experiment carried out to validate Asura gathered data of two types: usage data and questionnaire responses. Usage data was automatically build up from the actions executed by students, and aims to evaluate the effectiveness of Asura in increasing practice and improving learning of programming. Questionnaire responses intend to assess the usability of the environment according to three dimensions: Usefulness, Satisfaction, and Ease of Use (and Ease of Learning).

The questionnaire follows Lund’s model [130], including one section per metric (*Usefulness*, *Satisfaction*, *Ease of Use*, and *Ease of Learning*) and an additional section with free-text questions to collect students’ feedback about Asura regarding weaknesses, strengths, and points of improvement.

## Asura - Questionnaire for Students

This survey aims to evaluate the usefulness and effectiveness of the system. Please be honest with your answers!

\*Required

### 1. Mooshak Username \*

## Usefulness

**2. The environment helps me to be more effective. \***

Mark only one oval.

[illegible]

3. The environment helps me to be more productive. \*

Mark only one oval.

[illegible]

**4. The environment is useful. \***

Mark only one oval.

[illegible]

**5. The environment makes learning easier. \***

Mark only one oval.

[illegible]

**6. The environment saves me time while learning. \***

Mark only one oval.

	1	2	3	4	5	6	7
Strongly Disagree	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Strongly Agree							





**14. Using the environment is effortless. \***

Mark only one oval.

[illegible]

Strongly Disagree

Strongly Agree

15. I can use the environment without written instructions. \*

Mark only one oval.

[illegible]

Strongly Disagree

Strongly Agree

16. I did not notice any inconsistencies as I use the environment. \*

Mark only one oval.

[illegible]

Strongly Disagree

Strongly Agree

17. Both occasional and regular users would like the environment. \*

Mark only one oval.

[illegible]

Strongly Disagree

Strongly Agree

18. I can recover from mistakes quickly and easily. \*

Mark only one oval.

[illegible]

Strongly Disagree

Strongly Agree

19. I can use the environment successfully every time. \*

Mark only one oval.

[illegible]

Strongly Disagree

Strongly Agree

## Ease of Learning

20. I learned to use the environment quickly. \*

Mark only one oval.

	1	2	3	4	5	6	7
Strongly Disagree	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Strongly Agree							

Strongly Disagree

Strongly Agree

21. I easily remember how to use the environment. \*

Mark only one oval.

[illegible]

22. It is easy to learn to use the environment. \*

Mark only one oval.

[illegible]

23. I quickly became skillful with the environment. \*

Mark only one oval.

	1	2	3	4	5	6	7	
Strongly Disagree	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	Strongly Agree

## Satisfaction

24. I am satisfied with the environment. \*

Mark only one oval.

	1	2	3	4	5	6	7	
Strongly Disagree	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	Strongly Agree

25. I would recommend the environment to a friend. \*

Mark only one oval.

[illegible]

26. The environment is fun to use. \*

Mark only one oval.

	1	2	3	4	5	6	7
Strongly Disagree	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Strongly Agree							

27. The environment works the way I want it to work. \*

Mark only one oval.

	1	2	3	4	5	6	7
Strongly Disagree	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/> Strongly Agree

28. The environment is wonderful. \*

Mark only one oval.

	1	2	3	4	5	6	7	
Strongly Disagree	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	Strongly Agree

29. I feel I need to have it in my classes. \*

Mark only one oval.

	1	2	3	4	5	6	7	
Strongly Disagree	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	Strongly Agree

30. The environment is pleasant to use. \*

Mark only one oval.

	1	2	3	4	5	6	7	
Strongly Disagree	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	Strongly Agree

Comments

31. Strengths


32. Weaknesses


33. Suggestions




## G.3 Test of Gamer Personality

At the end of the course, participants were also invited to perform a test to identify their gamer personality. This test is based on Bartle's Test of Gamer Psychology [134], which classifies players of multiplayer online games into one of the four categories defined by Bartle's taxonomy of player types [14]: achiever, explorer, socializer, and killer.

The purpose of the results of this test is to check for correlations between the player personality and its behavior during the experiment, analyzed through usage data.

# What type of gamer are you?

This survey aims to identify which type of gamer personality fits best on you, according to Bartle's player types. In fact, the outcome of this questionnaire will contain your percentage of each type of gamer personality: achiever, explorer, socializer or killer. We will use this data anonymized to conduct an analysis which aims to relate gamer psychology with the behavior during the experiment.

Please, be honest with your answers. You will receive an email with your result!

**\*Required**

## 1. E-mail

---

## 2. Mooshak Username \*

---

**Please consider for the next questions that you are a player in a multiplayer online game.**

---

## 3. I would rather win ... \*

*Mark only one oval.*

☐ ... a knowledge contest.

☐ ... a battle.

## 4. I would rather ... \*

*Mark only one oval.*

☐ ... defeat an enemy.

☐ ... explore a new area.

## 5. I would rather ... \*

*Mark only one oval.*

☐ ... become a hero faster than my friends.

☐ ... know more secrets than my friends.

## 6. If I'm alone in an area, I would ... \*

*Mark only one oval.*

☐ ... explore it.

☐ ... look elsewhere for a prey.

## 7. I would rather have ... \*

*Mark only one oval.*

☐ ... a private channel, over which I can communicate with my friends.

☐ ... a premium account and a great amount of world coins.

**8. The most exciting for me is ... \***

*Mark only one oval.*

- ☐ ... a well-roleplayed scenario.
- ☐ ... a deadly battle.

**9. If I could choose what to add to the game, I would add ... \***

*Mark only one oval.*

- ☐ ... people.
- ☐ ... areas to explore.

**10. I would rather be ... \***

*Mark only one oval.*

- ☐ ... popular.
- ☐ ... unknown, but have powerful equipment.

**11. I would prefer to be ... \***

*Mark only one oval.*

- ☐ ... feared by other players.
- ☐ ... loved by other players.

**12. I would rather have ... \***

*Mark only one oval.*

- ☐ ... a spell to damage other players.
- ☐ ... a spell that increases the rate at which I gain experience points.

**13. If I know that another player is planning my demise, I would ... \***

*Mark only one oval.*

- ☐ ... prepare myself in an area that my opponent is unfamiliar with.
- ☐ ... attack him before he attacks me.

**14. If I was being chased by a very powerful non-player character, I would ... \***

*Mark only one oval.*

- ☐ ... ask a friend for help to defeat it.
- ☐ ... run away and hide where I know that it can't reach me.

**15. I would rather ... \***

*Mark only one oval.*

- ☐ ... vanquish my enemies.
- ☐ ... convince my enemies to work with me.

**16. I would rather be known for ... \***

*Mark only one oval.*

- ☐ ... knowledge.
- ☐ ... power.

**17. I would be more prone to brag about ... \***

*Mark only one oval.*

- ☐ ... how many players I've killed.
- ☐ ... the quality of my items.

**18. I would rather be known as ... \***

*Mark only one oval.*

- ☐ ... someone who knows every detail about the game, including its easter eggs.
- ☐ ... the person with the best and most unique equipment in the game.

**19. I feel more engaged ... \***

*Mark only one oval.*

- ☐ ... killing a big monster.
- ☐ ... bragging about it to my friends.

**20. I would rather ... \***

*Mark only one oval.*

- ☐ ... have the highest score on the list.
- ☐ ... beat my best friend one-on-one.

**21. I would rather ... \***

*Mark only one oval.*

- ☐ ... hear what someone has to say.
- ☐ ... show them my full power.

**22. I'm more comfortable ... \***

*Mark only one oval.*

- ☐ ... talking with friends in a tavern.
- ☐ ... out hunting orcs by yourself for experience.

**23. I would rather ... \***

*Mark only one oval.*

- ☐ ... know where I can explore to find things.
- ☐ ... know a way to get things fast.

**24. I prefer to ... \***

*Mark only one oval.*

- ☐ ... get involved in a forum/chat thread.
- ☐ ... get a new item.

**25. If I had one opportunity to explore a new area with another player, I would choose ... \***

*Mark only one oval.*

- ☐ ... a good friend which is a very weak player.
- ☐ ... a player which has many experience in exploring new areas.



**26. I would rather receive as reward ... \***

*Mark only one oval.*

- ☐ ... points of experience.
- ☐ ... a spell that lets me control other players, against their will for a small time.

**27. I would rather have ... \***

*Mark only one oval.*

- ☐ ... two levels of experience.
- ☐ ... a 10% bonus of attack.

**28. I would rather join ... \***

*Mark only one oval.*

- ☐ ... a familiar guild.
- ☐ ... a guild of killers.

**29. When I meet a new player, I think of him as ... \***

*Mark only one oval.*

- ☐ ... someone that can appreciate my knowledge of the game.
- ☐ ... a new prey.

**30. I prefer ... \***

*Mark only one oval.*

- ☐ ... a good roleplaying.
- ☐ ... the uniqueness of features, and game mechanics.

**31. I would prefer to ... \***

*Mark only one oval.*

- ☐ ... win a duel with another player.
- ☐ ... get accepted by a guild.

**32. If someone attacks me, I would ... \***

*Mark only one oval.*

- ☐ ... find out why, and try to convince them not to do it again.
- ☐ ... plot a revenge.

**33. I would enjoy more ... \***

*Mark only one oval.*

- ☐ ... running my own guild.
- ☐ ... making my own maps of the world.

**34. I would rather be known for my ... \***

*Mark only one oval.*

- ☐ ... equipment.
- ☐ ... personality.

**35. I prefer to ... \***

*Mark only one oval.*

- ☐ ... get involved in a good storyline.
- ☐ ... get rewards after completing a mission.

**36. I would rather ... \***

*Mark only one oval.*

- ☐ ... have the most powerful item of a game.
- ☐ ... be the most feared person in the game.

**37. I tend to ... \***

*Mark only one oval.*

- ☐ ... know things no one else does.
- ☐ ... have items no one else does.

**38. If I want to fight a very powerful non-player character, I would ... \***

*Mark only one oval.*

- ☐ ... ask my guild for help to defeat it.
- ☐ ... try every possible attack to find a weakness.

**39. What excites me more when something new appears is ... \***

*Mark only one oval.*

- ☐ ... the opportunity to explore new things and to know more about them.
- ☐ ... the opportunity to be the first collecting new items.

**40. I would rather ... \***

*Mark only one oval.*

- ☐ ... solve a riddle no one else has gotten.
- ☐ ... get to a certain experience level faster than anyone else.

**41. It is worse to be ... \***

*Mark only one oval.*

- ☐ ... without power.
- ☐ ... without guild.

---

Powered by

 Google Forms

# Bibliography

- [1] Wilfried Admiraal, Jantina Huizenga, Sanne Akkerman, and Geert Ten Dam. [The concept of flow in collaborative game-based learning](#). *Computers in Human Behavior*, 27(3): 1185–1194, 2011. doi:10.1016/j.chb.2010.12.013.
- [2] Marco Aedo Lopez, Elizabeth Vidal Duarte, Eveling Castro Gutierrez, and Alfredo Paz Valderrama. [Teaching abstraction, function and reuse in the first class of cs1: A lightbot experience](#). In *Proceedings of the 2016 ACM Conference on Innovation and Technology in Computer Science Education*, ITiCSE '16, pages 256–257. ACM, 2016. ISBN: 978-1-4503-4231-5. doi:10.1145/2899415.2925505.
- [3] June Ahn, Brian S. Butler, Alisha Alam, and Sarah A. Webster. Learner participation and engagement in open online courses: Insights from the Peer 2 Peer University. *Journal of Online Learning and Teaching*, 9(2):160, 2013.
- [4] Mohammad Al-Smadi and Christian Gütl. [Soa-based architecture for a generic and flexible e-assessment system](#). In *Education Engineering (EDUCON), 2010 IEEE*, pages 493–500. IEEE, 2010. doi:10.1109/educn.2010.5492537.
- [5] Paul E. Anderson, Thomas Nash, and Renée McCauley. [Facilitating Programming Success in Data Science Courses Through Gamified Scaffolding and Learn2Mine](#). In *Proceedings of the 2015 ACM Conference on Innovation and Technology in Computer Science Education*, ITiCSE '15, pages 99–104, New York, NY, USA, 2015. ACM. ISBN: 978-1-4503-3440-2. doi:10.1145/2729094.2742597.
- [6] Manuel Araújo and Licínio Roque. Modeling games with petri nets. In *DiGRA Conference*, 2009.
- [7] Ashley Stahl. [Seven Top In-Demand Jobs In 2018](#). Forbes, 2018. [visited June 2018].
- [8] Samir E. Ashoo, Troy Boudreau, and Douglas A. Lane. [CSUS Programming Contest Control System \(PC2\)](#), 2018 [visited September 2018].
- [9] Alexander W. Astin. Student involvement: A developmental theory for higher education. *Journal of college student personnel*, 25(4):297–308, 1984.

- [10] Tapio Auvinen, Lasse Hakulinen, and Lauri Malmi. [Increasing students' awareness of their behavior in online learning environments with visualizations and achievement badges](#). *IEEE Transactions on Learning Technologies*, 8(3):261–273, 2015. doi:10.1109/tlt.2015.2441718.
- [11] Elliott M. Avedon and Brian Sutton-Smith. *The study of games*. John Wiley & Sons, 1971. ISBN: 9780471038399.
- [12] Gabriel Barata, Sandra Gama, Joaquim Jorge, and Daniel Gonçalves. [Studying student differentiation in gamified education: A long-term study](#). *Computers in Human Behavior*, 71(Supplement C):550 – 585, 2017. ISSN: 0747-5632. doi:https://doi.org/10.1016/j.chb.2016.08.049.
- [13] Tiffany Barnes, Heather Richter, Eve Powell, Amanda Chaffin, and Alex Godwin. [Game2learn: Building cs1 learning games for retention](#). In *Proceedings of the 12th Annual SIGCSE Conference on Innovation and Technology in Computer Science Education*, ITiCSE '07, pages 121–125. ACM, 2007. ISBN: 978-1-59593-610-3. doi:10.1145/1268784.1268821.
- [14] Richard Bartle. Hearts, clubs, diamonds, spades: Players who suit MUDs. *Journal of MUD research*, 1(1):19, 1996.
- [15] Sarit Barzilai and Ina Blau. [Scaffolding game-based learning: Impact on learning achievements, perceived learning, and game experiences](#). *Computers & Education*, 70:65–79, 2014. doi:10.1016/j.compedu.2013.08.003.
- [16] Theresa Beaubouef and John Mason. [Why the high attrition rate for computer science students: Some thoughts and observations](#). *SIGCSE Bull.*, 37(2):103–106, June 2005. ISSN: 0097-8418. doi:10.1145/1083431.1083474.
- [17] Stephanie Bell. [Project-based learning for the 21st century: Skills for the future](#). *The Clearing House: A Journal of Educational Strategies, Issues and Ideas*, 83(2):39–43, 2010. doi:10.1080/00098650903505415.
- [18] Jens Bennedsen and Michael E. Caspersen. [Exposing the Programming Process](#), pages 6–16. Springer Berlin Heidelberg, Berlin, Heidelberg, 2008. ISBN: 978-3-540-77934-6. doi:10.1007/978-3-540-77934-6\_2.
- [19] Susan Bergin and Ronan Reilly. [The influence of motivation and comfort-level on learning to program](#). In *Proceedings of the 17th Workshop of the Psychology of Programming Interest Group*, pages 293–304. Psychology of Programming Interest Group, 2005.
- [20] K. Berkling and C. Thomas. [Gamification of a software engineering course and a detailed analysis of the factors that lead to it's failure](#). In *2013 International Conference on Interactive Collaborative Learning (ICL)*, pages 525–530, Sept 2013. doi:10.1109/ICL.2013.6644642.
- [21] Andrija Bernik, Goran Bubas, and Danijel Radosevic. A Pilot Study of the Influence of Gamification on the Effectiveness of an e-Learning Course. In *Central European Conference*

- on Information and Intelligent Systems*, page 73. Faculty of Organization and Informatics Varazdin, 2015.
- [22] TAXONOMY MADE EASY BLOOM'S. *Bloom's taxonomy of educational objectives*. Longman, 1965.
- [23] Rick Blumenthal. Space Invaders: A UML Case Study. Regis University, 2005. class notes.
- [24] Mads T. Bonde, Guido Makransky, Jakob Wandall, Mette V. Larsen, Mikkel Morsing, Hanne Jarmer, and Morten O. A. Sommer. [Improving biotech education through gamified laboratory simulations](#). *Nature biotechnology*, 32(7):694–697, 2014. doi:10.1038/nbt.2955.
- [25] Brenda Brathwaite and Ian Schreiber. *Challenges for Game Designers*. Charles River Media, Inc., 1 edition, 2008. ISBN: 158450580X, 9781584505808.
- [26] Brian Burke. [Gartner Says by 2014, 80 Percent of Current Gamified Applications Will Fail to Meet Business Objectives Primarily Due to Poor Design](#), 2012 [visited October 2017].
- [27] Brian Burke. [Gartner Redefines Gamification](#), 2014 [visited October 2017].
- [28] Cyril Brom and Adam Abonyi. Petri-nets for game plot. In *Proceedings of AISB artificial intelligence and simulation behaviour convention, Bristol*, volume 3, pages 6–13, 2006.
- [29] Cyril Brom, Editá Bromová, Filip Děchtěrenko, Michaela Buchtová, and Martin Pergel. [Personalized messages in a brewery educational simulation: Is the personalization principle less robust than previously thought?](#) *Computers & Education*, 72:339–366, 2014. doi:10.1016/j.compedu.2013.11.013.
- [30] Jere Brophy. On motivating students. *Occasional Paper No. 101*, 1986.
- [31] Jere Brophy. *Motivating students to learn*. Routledge, 2013.
- [32] Patrick Buckley and Elaine Doyle. [Gamification and student motivation](#). *Interactive Learning Environments*, 24(6):1162–1175, 2016. doi:10.1080/10494820.2014.964263.
- [33] Inc Bunchball. Gamification 101: An introduction to the use of game dynamics to influence behavior. *White paper*, page 9, 2010.
- [34] Juan C. Burguillo. [Using game theory and competition-based learning to stimulate student motivation and performance](#). *Computers & Education*, 55(2):566–575, 2010. doi:10.1016/j.compedu.2010.02.018.
- [35] Nergiz Ercil Cagiltay, Erol Ozcelik, and Nese Sahin Ozcelik. [The effect of competition on learning in games](#). *Computers & Education*, 87:35–41, 2015. doi:10.1016/j.compedu.2015.04.001.
- [36] Roger Caillois. *Man, play, and games*. University of Illinois Press, 1961. ISBN: 978-0252070334.

- [37] Tracy Camp, Stu Zweben, Ellen Walker, and Lecia Barker. [Booming enrollments: Good times?](#) In *Proceedings of the 46th ACM Technical Symposium on Computer Science Education*, SIGCSE '15, pages 80–81, New York, NY, USA, 2015. ACM. ISBN: 978-1-4503-2966-8. doi:10.1145/2676723.2677333.
- [38] CareerCast. [The Most In-Demand and Fast-Growing Jobs of 2017](#), 2017 [visited June 2018].
- [39] Oskar Casquero, Javier Portillo, Ramón Ovelar, Manuel Benito, and Jesús Romo. [iPLE Network: an integrated eLearning 2.0 architecture from a university's perspective](#). *Interactive Learning Environments*, 18(3):293–308, 2010. doi:10.1080/10494820.2010.500553.
- [40] Amanda Chaffin, Katelyn Doran, Drew Hicks, and Tiffany Barnes. [Experimental evaluation of teaching recursion in a video game](#). In *Proceedings of the 2009 ACM SIGGRAPH Symposium on Video Games*, Sandbox '09, pages 79–86. ACM, 2009. ISBN: 978-1-60558-514-7. doi:10.1145/1581073.1581086.
- [41] Kuo-En Chang, Lin-Jung Wu, Sheng-En Weng, and Yao-Ting Sung. [Embedding game-based problem-solving phase into problem-posing system for mathematics learning](#). *Computers & Education*, 58(2):775–786, 2012. doi:10.1016/j.compedu.2011.10.002.
- [42] Li-Jie Chang, Jie-Chi Yang, and Fu-Yun Yu. [Development and evaluation of multiple competitive activities in a synchronous quiz game system](#). *Innovations in Education and Teaching International*, 40(1):16–26, 2003. doi:10.1080/1355800032000038840.
- [43] Brenda Cheang, Andy Kurnia, Andrew Lim, and Wee-Chong Oon. [On automated grading of programming assignments in an academic institution](#). *Computers & Education*, 41(2): 121 – 131, 2003. ISSN: 0360-1315. doi:https://doi.org/10.1016/S0360-1315(03)00030-7.
- [44] Katheryn R. Christy and Jesse Fox. [Leaderboards in a virtual classroom: A test of stereotype threat and social comparison explanations for women's math performance](#). *Computers & Education*, 78(Supplement C):66 – 77, 2014. ISSN: 0360-1315. doi:10.1016/j.compedu.2014.05.005.
- [45] J McGrath Cohoon, Margaret Gonsoulin, and James Layman. [Mentoring computer science undergraduates](#). *WIT Transactions on Information and Communication Technologies*, 31, 2004. doi:10.2495/CI040211.
- [46] Miguel Ángel Conde, Francisco J García, María J Casany, and Marc Aliet. [Applying web services to define open learning environments](#). In *Database and expert systems applications (dexa), 2010 workshop on*, pages 79–83. IEEE, 2010. doi:10.1109/dexa.2010.36.
- [47] Raul Ferrer Conill and Michael Karlsson. [The gamification of journalism](#). In *Emerging Research and Trends in Gamification*, pages 356–383. IGI Global, 2016. doi:10.4018/978-1-4666-8651-9.ch015.

- [48] Thomas M. Connolly, Elizabeth A. Boyle, Ewan MacArthur, Thomas Hainey, and James M. Boyle. [A systematic literature review of empirical evidence on computer games and serious games](#). *Computers & Education*, 59(2):661–686, 2012. ISSN: 0360-1315. doi:<https://doi.org/10.1016/j.compedu.2012.03.004>.
- [49] Chris Crawford. *The art of computer game design*. Osborne/McGraw-Hill Berkeley, CA, 1984.
- [50] Geoffrey T. Crisp. [Assessment in next generation learning spaces](#). In *The future of learning and teaching in next generation learning spaces*, pages 85–100. Emerald Group Publishing Limited, 2014. doi:10.1108/s1479-362820140000012009.
- [51] Mihaly Csikszentmihalyi. [Toward a psychology of optimal experience](#). In *Flow and the foundations of positive psychology*, pages 209–226. Springer, 2014. doi:10.1007/978-94-017-9088-8\_14.
- [52] Mihaly Csikszentmihalyi, Kevin Rathunde, and Samuel Whalen. *Talented teenagers: The roots of success and failure*. Cambridge University Press, 1997. ISBN: 9780521574631.
- [53] Declan Dagger, Alexander O’Connor, Seamus Lawless, Eddie Walsh, and Vincent P Wade. [Service-oriented e-learning platforms: From monolithic systems to flexible services](#). *IEEE Internet Computing*, 11(3), 2007. doi:10.1109/mic.2007.70.
- [54] V. Dagiene and J. Skupiene. [Learning by competitions: olympiads in informatics as a tool for training high-grade skills in programming](#). In *ITRE 2004. 2nd International Conference Information Technology: Research and Education*, pages 79–83, June 2004. doi:10.1109/ITRE.2004.1393650.
- [55] Fred D. Davis. [Perceived usefulness, perceived ease of use, and user acceptance of information technology](#). *MIS Quarterly*, 13(3):319–340, 1989. ISSN: 02767783. doi:10.2307/249008.
- [56] Sagarmay Deb. [Information technology, its impact on society and its future](#). *Advances in Computing*, 4(1):25–29, 2014. doi:10.5923/j.ac.20140401.07.
- [57] Sebastian Deterding. [Pawnd. Gamification and Its Discontents](#). <https://www.slideshare.net/dings/pawnd-gamification-and-its-discontents>, 2010 [visited October 2017].
- [58] Sebastian Deterding, Dan Dixon, Rilla Khaled, and Lennart Nacke. [From game design elements to gamefulness: defining gamification](#). In *Proceedings of the 15th international academic MindTrek conference: Envisioning future media environments*, pages 9–15. ACM, 2011. doi:10.1145/2181037.2181040.
- [59] Jens Dietrich, Johannes Tandler, Li Sui, and Manfred Meyer. [The primegame revolutions: A cloud-based collaborative environment for teaching introductory programming](#). In *Proceedings of the ASWEC 2015 24th Australasian Software Engineering Conference*, ASWEC ’15 Vol. II, pages 8–12, New York, NY, USA, 2015. ACM. ISBN: 978-1-4503-3796-0. doi:10.1145/2811681.2811683.



- [60] Marcus Dithmer, Jack Ord Rasmussen, Erik Grönvall, Helle Spindler, John Hansen, Gitte Nielsen, Stine Bæk Sørensen, and Birthe Dinesen. [“The Heart Game”: Using Gamification as Part of a Telerehabilitation Program for Heart Patients](#). *Games for health journal*, 5(1): 27–33, 2016. doi:10.1089/g4h.2015.0001.
- [61] Joris Dormans. Machinations: Elemental feedback structures for game design. In *Proceedings of the GAMEON-NA Conference*, pages 33–40, 2009.
- [62] Michael Eagle and Tiffany Barnes. [Wu’s castle: teaching arrays and loops in a game](#). In *Proceedings of the 13th annual conference on Innovation and technology in computer science education*, ITiCSE ’08, pages 245–249. ACM, 2008. ISBN: 978-1-60558-078-4. doi:http://doi.acm.org/10.1145/1384271.1384337.
- [63] Anna Eckerdal, Robert McCartney, Jan Erik Moström, Mark Ratcliffe, and Carol Zander. [Can graduating students design software systems?](#) In *Proceedings of the 37th SIGCSE Technical Symposium on Computer Science Education*, SIGCSE ’06, pages 403–407, New York, NY, USA, 2006. ACM. ISBN: 1-59593-259-3. doi:10.1145/1121341.1121468.
- [64] Stephen H. Edwards, Jürgen Börstler, Lillian N. Cassel, Mark S. Hall, and Joseph Hollingsworth. [Developing a Common Format for Sharing Programming Assignments](#). *SIGCSE Bull.*, 40(4):167–182, November 2008. ISSN: 0097-8418. doi:10.1145/1473195.1473240.
- [65] Jaap Eldering, Thijs Kinkhorst, and Peter van de Warken. [DOMjudge-programming contest jury system](#), 2011 [visited November 2017].
- [66] Christian Elverdam and Espen Aarseth. [Game Classification and Game Design: Construction Through Critical Analysis](#). *Games and Culture*, 2(1):3–22, 2007. doi:10.1177/1555412006286892.
- [67] Entertainment Software Association. [Essential Facts About the Computer and Video Game Industry](#). Technical report, Entertainment Software Association, 2017 [visited December 2017].
- [68] Usef Faghihi, Albert Brautigam, Kris Jorgenson, David Martin, Angela Brown, Elizabeth Measures, and Sioui Maldonado-Bouchard. [How gamification applies for educational purpose specially with college algebra](#). *Procedia Computer Science*, 41(Supplement C):182 – 187, 2014. ISSN: 1877-0509. 5th Annual International Conference on Biologically Inspired Cognitive Architectures, 2014 BICA. doi:https://doi.org/10.1016/j.procs.2014.11.102.
- [69] Christopher J. Ferguson and Adolfo Garza. [Call of \(civic\) duty: Action games and civic behavior in a large sample of youth](#). *Computers in Human Behavior*, 27(2):770–775, 2011. ISSN: 0747-5632. Web 2.0 in Travel and Tourism: Empowering and Changing the Role of Travelers. doi:https://doi.org/10.1016/j.chb.2010.10.026.



- [70] Jared Z. Ferrell, Jacqueline E. Carpenter, E. Daly Vaughn, Nikki M. Dudley, and Scott A. Goodman. [Gamification of human resource processes](#). In *Emerging research and trends in gamification*, pages 108–139. IGI Global, 2016. doi:10.4018/978-1-4666-8651-9.ch006.
- [71] Per Fors and Thomas Taro Lennerfors. Gamification for sustainability. In *The Business of Gamification: A Critical Analysis*, page 163. Taylor & Francis, 2016. ISBN: 9781317581451.
- [72] De Grove Frederik, Mechant Peter, and Van Looy Jan. [Uncharted waters?: exploring experts’ opinions on the opportunities and limitations of serious games for foreign language learning](#). In *Proceedings of the 3rd International Conference on Fun and Games*, pages 107–115. ACM, 2010. doi:10.1145/1823818.1823830.
- [73] Jordan Frith. [Turning life into a game: Foursquare, gamification, and personal mobility](#). *Mobile Media & Communication*, 1(2):248–262, 2013. doi:10.1177/2050157912474811.
- [74] Daniel D. Garcia, Jennifer Campbell, John DeNero, Mary Lou Dorf, and Stuart Reges. [Cs10k teachers by 2017?: Try cs1k+ students now! coping with the largest cs1 courses in history](#). In *Proceedings of the 47th ACM Technical Symposium on Computing Science Education, SIGCSE ’16*, pages 396–397, New York, NY, USA, 2016. ACM. ISBN: 978-1-4503-3685-7. doi:10.1145/2839509.2844660.
- [75] Ginés García-Mateos and José Luis Fernández-Alemán. [A course on algorithms and data structures using on-line judging](#). In *Proceedings of the 14th Annual ACM SIGCSE Conference on Innovation and Technology in Computer Science Education, ITiCSE ’09*, pages 45–49. ACM, 2009. ISBN: 978-1-60558-381-5. doi:10.1145/1562877.1562897.
- [76] James Paul Gee. [What video games have to teach us about learning and literacy](#). *Comput. Entertain.*, 1(1):20–20, October 2003. ISSN: 1544-3574. doi:10.1145/950566.950595.
- [77] Katerina Georgouli and Pedro Guerreiro. [Incorporating an automatic judge into blended learning programming activities](#). *Advances in Web-Based Learning–ICWL 2010*, pages 81–90, 2010. doi:10.1007/978-3-642-17407-0\_9.
- [78] Stefan Göbel, Sandro Hardy, Viktor Wendel, Florian Mehm, and Ralf Steinmetz. [Serious games for health: personalized exergames](#). In *Proceedings of the 18th ACM international conference on Multimedia*, pages 1663–1666. ACM, 2010. doi:10.1145/1873951.1874316.
- [79] Agoritsa Gogoulou, Evangelia Gouli, Maria Grigoriadou, Maria Samarakou, and Dionisia Chinou. A web-based educational setting supporting individualized learning, collaborative learning and assessment. *Journal of Educational Technology & Society*, 10(4), 2007. ISSN: 11763647.
- [80] Miguel Ángel Conde González, Francisco José García Peñalvo, María José Casany Guerrero, and Marc Alíer Forment. [Adapting LMS architecture to the SOA: an Architectural Approach](#). In *Internet and Web Applications and Services, 2009. ICIW’09. Fourth International Conference on*, pages 322–327. IEEE, 2009. doi:10.1109/iciw.2009.54.

- [81] Neil Andrew Gordon. Issues in retention and attainment in computer science. *York: Higher Education Academy*, 2016.
- [82] C. S. Green and D. Bavelier. [Learning, attentional control, and action video games](#). *Current Biology*, 22(6):R197–R206, 2012. ISSN: 0960-9822. doi:<https://doi.org/10.1016/j.cub.2012.02.012>.
- [83] Pedro Guerreiro and Katerina Georgouli. Enhancing elementary programming courses using e-learning with a competitive attitude. *International Journal of Internet Education*, 10, 01 2008.
- [84] Lassi Haaranen, Petri Ihantola, Lasse Hakulinen, and Ari Korhonen. [How \(not\) to introduce badges to online exercises](#). In *Proceedings of the 45th ACM Technical Symposium on Computer Science Education, SIGCSE '14*, pages 33–38, New York, NY, USA, 2014. ACM. ISBN: 978-1-4503-2605-6. doi:10.1145/2538862.2538921.
- [85] Lasse Hakulinen, Tapio Auvinen, and Ari Korhonen. [The Effect of Achievement Badges on Students' Behavior: An Empirical Study in a University-Level Computer Science Course](#). *International Journal of Emerging Technologies in Learning*, 10(1), 2015. doi:10.3991/ijet.v10i1.4221.
- [86] Juho Hamari and Jonna Koivisto. [Measuring flow in gamification: Dispositional flow scale-2](#). *Computers in Human Behavior*, 40:133–143, 2014. doi:10.1016/j.chb.2014.07.048.
- [87] Juho Hamari, David J. Shernoff, Elizabeth Rowe, Brianno Coller, Jodi Asbell-Clarke, and Teon Edwards. [Challenging games help students learn: An empirical study on engagement, flow and immersion in game-based learning](#). *Computers in Human Behavior*, 54:170–179, 2016. doi:10.1016/j.chb.2015.07.045.
- [88] K. W. Han, E. Lee, and Y. Lee. [The impact of a peer-learning agent based on pair programming in a programming course](#). *IEEE Transactions on Education*, 53(2):318–327, May 2010. ISSN: 0018-9359. doi:10.1109/TE.2009.2019121.
- [89] Michael D. Hanus and Jesse Fox. [Assessing the effects of gamification in the classroom: A longitudinal study on intrinsic motivation, social comparison, satisfaction, effort, and academic performance](#). *Computers & Education*, 80:152–161, 2015. doi:10.1016/j.compedu.2014.08.019.
- [90] Ken Hartness. [Robocode: Using Games to Teach Artificial Intelligence](#). *J. Comput. Sci. Coll.*, 19(4):287–291, April 2004. ISSN: 1937-4771.
- [91] Tatsuhito Hasegawa, Makoto Koshino, and Hiromi Ban. [An English vocabulary learning support system for the learner's sustainable motivation](#). *SpringerPlus*, 4(1):99, 2015. doi:10.1186/s40064-015-0792-2.
- [92] Rachelle S. Heller, Cheryl Beil, Kim Dam, and Belinda Haerum. [Student and faculty perceptions of engagement in engineering](#). *Journal of Engineering Education*, 99(3):253–261, 2010. doi:10.1002/j.2168-9830.2010.tb01060.x.

- [93] P. Herzig, K. Jugel, C. Momm, M. Ameling, and A. Schill. [Gaml - a modeling language for gamification](#). In *2013 IEEE/ACM 6th International Conference on Utility and Cloud Computing*, pages 494–499, Dec 2013. doi:10.1109/UCC.2013.96.
- [94] Caitlin Holman, Stephen J. Aguilar, Adam Levick, Jeff Stern, Benjamin Plummer, and Barry Fishman. [Planning for Success: How Students Use a Grade Prediction Tool to Win Their Classes](#). In *Proceedings of the Fifth International Conference on Learning Analytics And Knowledge, LAK '15*, pages 260–264, New York, NY, USA, 2015. ACM. ISBN: 978-1-4503-3417-4. doi:10.1145/2723576.2723632.
- [95] Johan Huizinga. *Homo ludens: A study of the play-element in our culture*. Routledge & Kegan Paul, 1949. ISBN: 7100-0578-4.
- [96] Cheng-Yu Hung, Jerry Chih-Yuan Sun, and Pao-Ta Yu. [The benefits of a challenge: student motivation and flow experience in tablet-PC-game-based learning](#). *Interactive Learning Environments*, 23(2):172–190, 2015. doi:10.1080/10494820.2014.997248.
- [97] Robin Hunicke, Marc LeBlanc, and Robert Zubek. MDA: A formal approach to game design and game research. In *Proceedings of the AAAI Workshop on Challenges in Game AI*, volume 4, page 1722, 2004.
- [98] Robert Hutton. [The gamification of finance](#). *TOPIA: Canadian Journal of Cultural Studies*, 30–31, 2014. doi:10.3138/topia.30-31.207.
- [99] Wu-Yuin Hwang, Chin-Yu Wang, Gwo-Jen Hwang, Yueh-Min Huang, and Susan Huang. [A web-based programming learning environment to support cognitive development](#). *Interacting with Computers*, 20(6):524–534, 2008. doi:10.1016/j.intcom.2008.07.002.
- [100] María-Blanca Ibáñez, Angela Di-Serio, and Carlos Delgado-Kloos. [Gamification for engaging computer science students in learning activities: A case study](#). *IEEE Transactions on Learning Technologies*, 7(3):291–301, 2014. doi:10.1109/tlt.2014.2329293.
- [101] R. Ibrahim, J. Semarak, K. Lumpur, and A. Jaafar. [Using educational games in learning introductory programming: A pilot study on students' perceptions](#). In *2010 International Symposium on Information Technology*, volume 1, pages 1–5, June 2010. doi:10.1109/ITSIM.2010.5561414.
- [102] Petri Ihanola. [Automated assessment of programming assignments: visual feedback, assignment mobility, and assessment of students' testing skills](#). PhD thesis, Aalto University, 2011.
- [103] Linda A. Jackson, Edward A. Witt, Alexander Ivan Games, Hiram E. Fitzgerald, Alexander von Eye, and Yong Zhao. [Information technology use and creativity: Findings from the children and technology project](#). *Computers in Human Behavior*, 28(2):370–376, 2012. ISSN: 0747-5632. doi:https://doi.org/10.1016/j.chb.2011.10.006.

- [104] Jincheul Jang, Jason J. Y. Park, and Y. Yi Mun. [Gamification of online learning](#). In *International Conference on Artificial Intelligence in Education*, pages 646–649. Springer, 2015. doi:10.1007/978-3-319-19773-9\_82.
- [105] Tony Jenkins. On the difficulty of learning to program. In *Proceedings of the 3rd Annual Conference of the LTSN Centre for Information and Computer Sciences*, volume 4, pages 53–58, 2002.
- [106] L. Johnson, S. Adams Becker, V. Estrada, A. Freeman, P. Kampylis, R. Vuorikari, and Y. Punie. [Games and gamification](#). In *Horizon Report Europe: 2014 Schools Edition*, pages 42–43. Luxembourg: Publications Office of the European Union, & Austin, Texas: The New Media Consortium, 2014. ISBN: 978-92-79-38477-6. doi:10.2791/83704.
- [107] Korina Katsaliaki. [Serious games for sustainable development](#). *Journal of Management Education*, 37(6):889–894, 2013. doi:10.1177/1052562913509219.
- [108] Caitlin Kelleher and Randy F. Pausch. [Using storytelling to motivate programming](#). *Commun. ACM*, 50:58–64, 2007. doi:10.1145/1272516.1272540.
- [109] Kristian Kiili, Sara de Freitas, Sylvester Arnab, and Timo Lainema. [The design principles for flow experience in educational games](#). *Procedia Computer Science*, 15:78–91, 2012. doi:10.1016/j.procs.2012.10.060.
- [110] Alfie Kohn. *No contest: The case against competition*. Houghton Mifflin Harcourt, 1992.
- [111] Alfie Kohn. *Why incentive plans cannot work*, 1993.
- [112] Michael Kölling. [The greenfoot programming environment](#). *Trans. Comput. Educ.*, 10(4): 14:1–14:21, November 2010. ISSN: 1946-6226. doi:10.1145/1868358.1868361.
- [113] K. Kori, M. Pedaste, E. Tõnisson, T. Palts, H. Altin, R. Rantsus, R. Sell, K. Murtazin, and T. Rüütman. [First-year dropout in ict studies](#). In *2015 IEEE Global Engineering Education Conference (EDUCON)*, pages 437–445, March 2015. doi:10.1109/EDUCON.2015.7096008.
- [114] Theodora Koulouri, Stanislao Lauria, and Robert D. Macredie. [Teaching introductory programming: A quantitative evaluation of different approaches](#). *Trans. Comput. Educ.*, 14(4):26:1–26:28, December 2014. ISSN: 1946-6226. doi:10.1145/2662412.
- [115] Ivan Kuo. [Nike+: Building community and competitive advantage with gamification](#), 2015 [visited October 2017].
- [116] Elias Kyewski and Nicole C. Krämer. [To gamify or not to gamify? an experimental field study of the influence of badges on motivation, activity, and performance in an online learning course](#). *Computers & Education*, 118:25 – 37, 2018. ISSN: 0360-1315. doi:https://doi.org/10.1016/j.compedu.2017.11.006.
- [117] Angelo Kyrilov and David C. Noelle. [Binary instant feedback on programming exercises can reduce student engagement and promote cheating](#). In *Proceedings of the 15th Koli*

- Calling Conference on Computing Education Research*, Koli Calling '15, pages 122–126, New York, NY, USA, 2015. ACM. ISBN: 978-1-4503-4020-5. doi:10.1145/2828959.2828968.
- [118] Richard N. Landers and Amy K. Landers. [An empirical test of the theory of gamified learning: The effect of leaderboards on time-on-task and academic performance](#). *Simulation & Gaming*, 45(6):769–785, 2014. doi:10.1177/1046878114563662.
- [119] Reed W Larson and Maryse H Richards. [Boredom in the middle school years: Blaming schools versus blaming students](#). *American journal of education*, 99(4):418–443, 1991. doi:10.1086/443992.
- [120] Kris M. Y. Law, Victor C. S. Lee, and Y. T. Yu. [Learning motivation in e-learning facilitated computer programming courses](#). *Computers & Education*, 55(1):218 – 228, 2010. ISSN: 0360-1315. doi:https://doi.org/10.1016/j.compedu.2010.01.007.
- [121] Ramon Lawrence. [Teaching data structures using competitive games](#). *IEEE Transactions on Education*, 47(4):459–466, 2004. doi:10.1109/te.2004.825053.
- [122] Nicole Lazzaro. Why we play games: Four keys to more emotion without story. In *Game Developers Conference*, 2004.
- [123] José Paulo Leal and Ricardo Queirós. [Using the Learning Tools Interoperability Framework for LMS Integration in Service Oriented Architectures](#). *Technology Enhanced Learning TECH-EDUCATION'11*, 2011. doi:10400.22/4724.
- [124] José Paulo Leal and Fernando Silva. [Mooshak: A Web-based multi-site programming contest system](#). *Software: Practice and Experience*, 33(6):567–581, 2003. doi:10.1002/spe.522.
- [125] José Paulo Leal and Fernando Silva. Using Mooshak as a Competitive Learning Tool. In *The 2008 Competitive Learning Symposium*, 2008. ISBN: 978-84-937580-3-5.
- [126] Joey J. Lee, Eduard Matamoros, Rafael Kern, Jenna Marks, Christian de Luna, and William Jordan-Cooley. [Greenify: fostering sustainable communities via gamification](#). In *CHI'13 Extended Abstracts on Human Factors in Computing Systems*, pages 1497–1502. ACM, 2013. doi:10.1145/2468356.2468623.
- [127] Michael J. Lee and Andrew Ko. [Gidget](#), 2018 [visited July 2018].
- [128] Peter L. Liu. [Using open-source robocode as a java programming assignment](#). *SIGCSE Bull.*, 40(4):63–67, November 2008. ISSN: 0097-8418. doi:10.1145/1473195.1473222.
- [129] Chris Loftus, Lynda Thomas, and Carol Zander. [Can graduating students design: Revisited](#). In *Proceedings of the 42Nd ACM Technical Symposium on Computer Science Education, SIGCSE '11*, pages 105–110, New York, NY, USA, 2011. ACM. ISBN: 978-1-4503-0500-6. doi:10.1145/1953163.1953199.
- [130] Arnold M. Lund. Measuring usability with the use questionnaire. *Usability interface*, 8(2): 3–6, 2001.

- [131] M. Lykke, M. Coto, S. Mora, N. Vandel, and C. Jantzen. [Motivating programming students by problem based learning and lego robots](#). In *2014 IEEE Global Engineering Education Conference (EDUCON)*, pages 544–555, April 2014. doi:10.1109/EDUCON.2014.6826146.
- [132] Andrzej Marczewski. [Gamification Design vs Game Design](#), 2014 [visited July 2018].
- [133] Roozbeh Matloobi, Michael Blumenstein, and Steve Green. Extensions to generic automated marking environment: Game-2. In *Interactive Computer Aided Learning Conference*, volume 1, pages 1069–1076, 2009.
- [134] Matthew Barr. [The Bartle Test of Gamer Psychology](#), 2018 [visited July 2018].
- [135] Michael McCracken, Vicki Almstrum, Danny Diaz, Mark Guzdial, Dianne Hagan, Yifat Ben-David Kolikant, Cary Laxer, Lynda Thomas, Ian Utting, and Tadeusz Wilusz. [A multi-national, multi-institutional study of assessment of programming skills of first-year cs students](#). In *Working Group Reports from ITiCSE on Innovation and Technology in Computer Science Education*, ITiCSE-WGR '01, pages 125–180, New York, NY, USA, 2001. ACM. doi:10.1145/572133.572137.
- [136] W. J. McKeachie. [The rewards of teaching](#). *New Directions for Teaching and Learning*, 10: 7–13, 1982. doi:10.1002/tl.37219821003.
- [137] Jeanne Meister. [Future of work: Using gamification for human resources](#), 2015 [visited December 2017].
- [138] Elisa D. Mekler, Florian Brühlmann, Klaus Opwis, and Alexandre N Tuch. [Do points, levels and leaderboards harm intrinsic motivation?: an empirical analysis of common gamification elements](#). In *Proceedings of the First International Conference on gameful design, research, and applications*, pages 66–73. ACM, 2013. doi:10.1145/2583008.2583017.
- [139] Aaron S. Miller, Joseph A. Cafazzo, and Emily Seto. [A game plan: Gamification design principles in mHealth applications for chronic disease management](#). *Health informatics journal*, 22(2):184–193, 2016. doi:10.1177/1460458214537511.
- [140] Cheryl L. Morris and Gabriel M. Silberman. Programming contests in academic environments. In *fi*, pages F1F7–7. IEEE, 2003.
- [141] Robert Moser. [A fantasy adventure game as a learning environment: Why learning to program is so difficult and what can be done about it](#). In *Proceedings of the 2Nd Conference on Integrating Technology into Computer Science Education*, ITiCSE '97, pages 114–116, New York, NY, USA, 1997. ACM. ISBN: 0-89791-923-8. doi:10.1145/268819.268853.
- [142] T. Murata. [Petri nets: Properties, analysis and applications](#). *Proceedings of the IEEE*, 77(4):541–580, April 1989. ISSN: 0018-9219. doi:10.1109/5.24143.
- [143] Mathieu Muratet, Patrice Torquet, Fabienne Viallet, and Jean-Pierre Jessel. [Experimental feedback on prog&play: A serious game for programming practice](#). *Computer Graphics Forum*, 30:61–73, 2010. doi:10.1111/j.1467-8659.2010.01829.x.



- [144] Fiona Fui-Hoon Nah, Qing Zeng, Venkata Rajasekhar Telaprolu, Abhishek Padmanabhuni Ayyappa, and Brenda Eschenbrenner. *Gamification of Education: A Review of Literature*, pages 401–409. Springer International Publishing, Cham, 2014. ISBN: 978-3-319-07293-7. doi:10.1007/978-3-319-07293-7\_39.
- [145] Jeanne Nakamura and Mihaly Csikszentmihalyi. *Flow theory and research*. *Handbook of positive psychology*, pages 195–206, 2009. doi:10.1093/oxfordhb/9780195187243.013.0018.
- [146] Jeanne Nakamura and Mihaly Csikszentmihalyi. *The concept of flow*. In *Flow and the foundations of positive psychology*, pages 239–263. Springer, 2014. doi:10.1007/978-94-017-9088-8\_16.
- [147] Mark J Nelson. *Soviet and american precursors to the gamification of work*. In *Proceeding of the 16th International Academic MindTrek Conference*, pages 23–26. ACM, 2012. doi:10.1145/2393132.2393138.
- [148] Christa R Nevin, Andrew O Westfall, J Martin Rodriguez, Donald M Dempsey, Andrea Cherrington, Brita Roy, Mukesh Patel, and James H Willig. *Gamification as a tool for enhancing graduate medical education*. *Postgraduate Medical Journal*, 90(1070):685–693, 2014. ISSN: 0032-5473. doi:10.1136/postgradmedj-2013-132486.
- [149] David Nieborg. America’s Army: More than a game. Transforming Knowledge into Action through Gaming and Simulation. Ed. & Thomas Eberle Willy Christian Kriz. München: SAGSAGA, CD-ROM, 2004.
- [150] Nielsen. *Games 360 U.S. Report*, 2017 [visited November 2017].
- [151] Esko Nuutila, Seppo Törmä, and Lauri Malmi. *PBL and Computer Programming – The Seven Steps Method with Adaptations*. *Computer Science Education*, 15(2):123–142, 2005. doi:10.1080/08993400500150788.
- [152] Amanda Oddie, Paul Hazlewood, Stewart Blakeway, and Alma Whitfield. *Introductory problem solving and programming: Robotics versus traditional approaches*. *Innovation in Teaching and Learning in Information and Computer Sciences*, 9(2):1–11, 2010. doi:10.11120/ital.2010.09020011.
- [153] Florin Oprescu, Christian Jones, and Mary Katsikitis. *I PLAY AT WORK – ten principles for transforming work processes through gamification*. *Frontiers in psychology*, 5, 2014. doi:10.3389/fpsyg.2014.00014.
- [154] Igor Ostrovsky. *RoboZZle*, 2009 [visited July 2018].
- [155] José Carlos Paiva, José Paulo Leal, and Ricardo Queirós. *Odin: A service for gamification of learning activities*. In *International Symposium on Languages, Applications and Technologies*, pages 194–204. Springer, 2015. doi:10.1007/978-3-319-27653-3\_19.

- [156] José Carlos Paiva, José Paulo Leal, and Ricardo Alexandre Queirós. [Enki: A pedagogical services aggregator for learning programming languages](#). In *Proceedings of the 2016 ACM Conference on Innovation and Technology in Computer Science Education*, pages 332–337. ACM, 2016. doi:10.1145/2899415.2899441.
- [157] I. Paliokas, C. Arapidis, and M. Mpimpitsos. [Playlogo 3d: A 3d interactive video game for early programming education: Let logo be a game](#). In *2011 Third International Conference on Games and Virtual Worlds for Serious Applications*, pages 24–31, May 2011. doi:10.1109/VS-GAMES.2011.10.
- [158] John F. Pane and Brad A. Myers. Usability issues in the design of novice programming systems. Technical report, Carnegie-Mellon University Pittsburgh PA Department of Computer Science, 1996.
- [159] Marina Papastergiou. [Digital game-based learning in high school computer science education: Impact on educational effectiveness and student motivation](#). *Computers & Education*, 52(1):1–12, 2009. ISSN: 0360-1315. doi:https://doi.org/10.1016/j.compedu.2008.06.004.
- [160] Mads Kock Pedersen, Anette Svenningsen, Niels Bonderup Dohn, Andreas Lieberoth, and Jacob Sherson. [DiffGame: Game-based Mathematics Learning for Physics](#). *Procedia - Social and Behavioral Sciences*, 228(Supplement C):316 – 322, 2016. ISSN: 1877-0428. 2nd International Conference on Higher Education Advances, HEAd’16, 21-23 June 2016, València, Spain. doi:https://doi.org/10.1016/j.sbspro.2016.07.047.
- [161] Xu Pengcheng, Ying Fuchen, and Xie Di. [PKU JudgeOnline](#), 2013 [visited November 2017].
- [162] Bernadette Perry. [Gamifying French Language Learning: A Case Study Examining a Quest-based, Augmented Reality Mobile Learning-tool](#). *Procedia - Social and Behavioral Sciences*, 174(Supplement C):2308 – 2315, 2015. ISSN: 1877-0428. International Conference on New Horizons in Education, INTE 2014, 25-27 June 2014, Paris, France. doi:https://doi.org/10.1016/j.sbspro.2015.01.892.
- [163] Robin K. Pettit, Lise McCoy, Marjorie Kinney, and Frederic N. Schwartz. [Student perceptions of gamified audience response system interactions in large group lectures and via lecture capture technology](#). *BMC medical education*, 15(1):92, 2015. doi:10.1186/s12909-015-0373-7.
- [164] Andrew M. Phelps, Kevin J. Bierre, and David M. Parks. [Muppets: Multi-user programming pedagogy for enhancing traditional study](#). In *Proceedings of the 4th Conference on Information Technology Curriculum*, CITC4 ’03, pages 100–105. ACM, 2003. ISBN: 1-58113-770-2. doi:10.1145/947121.947143.
- [165] Paul R. Pintrich. [The role of motivation in promoting and sustaining self-regulated learning](#). *International Journal of Educational Research*, 31(6):459 – 470, 1999. ISSN: 0883-0355. doi:https://doi.org/10.1016/S0883-0355(99)00015-4.



- [166] Martinha Piteira and Carlos Costa. [Learning computer programming: Study of difficulties in learning programming](#). In *Proceedings of the 2013 International Conference on Information Systems and Design of Communication*, ISDOC '13, pages 75–80, New York, NY, USA, 2013. ACM. ISBN: 978-1-4503-2299-7. doi:10.1145/2503859.2503871.
- [167] Ricardo Queirós, Paulo José Leal, and José Campos. [Sequencing educational resources with Seqins](#). *Computer Science and Information Systems*, 11(4):1479–1497, 2014. doi:10.2298/csis131005074q.
- [168] Ricardo Queirós, José Paulo Leal, and José Carlos Paiva. [Integrating Rich Learning Applications in LMS](#). In Yanyan Li, Maiga Chang, Milos Kravcik, Elvira Popescu, Ronghuai Huang, Kinshuk, and Nian-Shing Chen, editors, *State-of-the-Art and Future Directions of Smart Learning*, pages 381–386. Springer Singapore, Singapore, 2016. ISBN: 978-981-287-868-7. doi:10.1007/978-981-287-868-7\_46.
- [169] Byron Reeves and J. Leighton Read. *Total engagement: How games and virtual worlds are changing the way people work and businesses compete*. Harvard Business Press, 2009.
- [170] Miguel A. Revilla, Shahriar Manzoor, and Rujia Liu. Competitive learning in informatics: The UVa online judge experience. *Olympiads in Informatics*, 2:131–148, 2008.
- [171] Pedro Manuel Pinto Ribeiro, Hugo Simões, and Michel Ferreira. Teaching artificial intelligence and logic programming in a competitive environment. *Informatics in Education*, 8(1):85–100, 2009.
- [172] Ganit Richter, Daphne R. Raban, and Sheizaf Rafaeli. [Studying gamification: the effect of rewards and incentives on motivation](#). In *Gamification in education and business*, pages 21–46. Springer, 2015. doi:10.1007/978-3-319-10208-5\_2.
- [173] Anthony Robins, Janet Rountree, and Nathan Rountree. [Learning and teaching programming: A review and discussion](#). *Computer science education*, 13(2):137–172, 2003. doi:10.1076/csed.13.2.137.14200.
- [174] Karen Robson, Kirk Plangger, Jan H. Kietzmann, Ian McCarthy, and Leyland Pitt. [Is it all a game? Understanding the principles of gamification](#). *Business Horizons*, 58(4): 411–420, 2015. doi:10.1016/j.bushor.2015.03.006.
- [175] Manuel Rubio-Sánchez, Päivi Kinnunen, Cristóbal Pareja-Flores, and Ángel Velázquez-Iturbide. [Student perception and usage of an automated programming assessment tool](#). *Computers in Human Behavior*, 31:453 – 460, 2014. ISSN: 0747-5632. doi:https://doi.org/10.1016/j.chb.2013.04.001.
- [176] Katie Salen and Eric Zimmerman. *Rules of play: Game design fundamentals*. MIT press, 2004. ISBN: 0262240459.
- [177] Sepandar Sepehr and Milena Head. [Competition As an Element of Gamification for Learning: An Exploratory Longitudinal Investigation](#). In *Proceedings of the First International*

- Conference on Gameful Design, Research, and Applications*, Gamification '13, pages 2–9, New York, NY, USA, 2013. ACM. ISBN: 978-1-4503-2815-9. doi:10.1145/2583008.2583009.
- [178] Valerie J. Shute, Matthew Ventura, Malcolm Bauer, and Diego Zapata-Rivera. Melding the power of serious games and embedded assessment to monitor and foster learning. *Serious games: Mechanisms and effects*, 2:295–321, 2009.
- [179] Miguel Sicart. Defining game mechanics. *Game Studies*, 8(2), 2008.
- [180] Martin Sillaots. [Achieving flow through gamification: a study on re-designing research methods courses](#). In *European Conference on Games Based Learning*, volume 2, page 538. Academic Conferences International Limited, 2014. doi:978-1-910309-57-5.
- [181] Jorge Simões, Rebeca Díaz Redondo, and Ana Fernández Vilas. [A social gamification framework for a K-6 learning platform](#). *Computers in Human Behavior*, 29(2):345–353, 2013. doi:10.1016/j.chb.2012.06.007.
- [182] Catherine Sotirakou and Constantinos Mourlas. [A Gamified News Application for Mobile Devices: An Approach that Turns Digital News Readers into Players of a Social Network](#). In *International Conference on Games and Learning Alliance*, pages 480–493. Springer, 2015. doi:10.1007/978-3-319-40216-1\_53.
- [183] Tarja Susi, Mikael Johannesson, and Per Backlund. [Serious games: An overview](#). Technical report, University of Skövde, Sweden, 2007.
- [184] Penelope Sweetser and Peta Wyeth. [GameFlow: A Model for Evaluating Player Enjoyment in Games](#). *Comput. Entertain.*, 3(3):3–3, July 2005. ISSN: 1544-3574. doi:10.1145/1077246.1077253.
- [185] M. J. Taylor, D. Gresty, and M. Baskett. [Computer game-flow design](#). *Comput. Entertain.*, 4(1), January 2006. ISSN: 1544-3574. doi:10.1145/1111293.1111300.
- [186] Alexandru Topîrceanu. [Gamified learning: A role-playing approach to increase student in-class motivation](#). *Procedia Computer Science*, 112(Supplement C):41 – 50, 2017. ISSN: 1877-0509. Knowledge-Based and Intelligent Information & Engineering Systems: Proceedings of the 21st International Conference, KES-20176-8 September 2017, Marseille, France. doi:https://doi.org/10.1016/j.procs.2017.08.017.
- [187] Andrew Trotman and Chris Handley. [Programming contest strategy](#). *Computers & Education*, 50(3):821–837, 2008. doi:10.1016/j.compedu.2006.08.008.
- [188] Nghi Truong, Paul Roe, and Peter Bancroft. [Static analysis of students' java programs](#). In *Proceedings of the Sixth Australasian Conference on Computing Education - Volume 30, ACE '04*, pages 317–325, Darlinghurst, Australia, Australia, 2004. Australian Computer Society, Inc.

- [189] Hakan Tüzün, Meryem Yılmaz-Soylu, Türkan Karakuş, Yavuz İnal, and Gonca Kızılkaya. [The effects of computer games on primary school students' achievement and motivation in geography learning](#). *Computers & Education*, 52(1):68–77, 2009. doi:10.1016/j.compedu.2008.06.008.
- [190] Virginia M. C. Tze, Lia M. Daniels, and Robert M. Klassen. [Evaluating the relationship between boredom and academic outcomes: a meta-analysis](#). *Educational Psychology Review*, 28(1):119–144, 2016. doi:10.1007/s10648-015-9301-y.
- [191] Andika Y. Utomo, Afifa Amriani, Alham F. Aji, Fatin R. N. Wahidah, and Kasiyah M. Junus. [Gamified E-learning model based on community of inquiry](#). In *Advanced Computer Science and Information Systems (ICACSIS), 2014 International Conference on*, pages 474–480. IEEE, 2014. doi:10.1109/icacsis.2014.7065830.
- [192] Sylke Vandercruysse, Mieke Vandewaetere, Frederik Cornillie, and Geraldine Clarebout. [Competition and students' perceptions in a game-based language learning environment](#). *Educational Technology Research and Development*, 61(6):927–950, 2013. doi:10.1007/s11423-013-9314-5.
- [193] Maarten Vansteenkiste and Edward L. Deci. [Competitively contingent rewards and intrinsic motivation: Can losers remain motivated?](#) *Motivation and emotion*, 27(4):273–299, 2003. ISSN: 1573-6644. doi:10.1023/A:1026259005264.
- [194] Matthew Ventura, Valerie Shute, and Yoon Jeon Kim. [Video gameplay, personality and academic performance](#). *Computers & Education*, 58(4):1260–1266, 2012. ISSN: 0360-1315. doi:https://doi.org/10.1016/j.compedu.2011.11.022.
- [195] Matthew Ventura, Valerie Shute, and Weinan Zhao. [The relationship between video game use and a performance-based measure of persistence](#). *Computers & Education*, 60(1):52–58, 2013. ISSN: 0360-1315. doi:https://doi.org/10.1016/j.compedu.2012.07.003.
- [196] Tom Verhoeff. Programming Task Packages: Peach Exchange. *Olympiads in Informatics*, page 192, 2008. ISSN: 1822-7732.
- [197] Tiantian Wang, Xiaohong Su, Peijun Ma, Yuying Wang, and Kuanquan Wang. [Ability-training-oriented automated assessment in introductory programming course](#). *Computers & Education*, 56(1):220 – 226, 2011. ISSN: 0360-1315. Serious Games. doi:https://doi.org/10.1016/j.compedu.2010.08.003.
- [198] X. Henry Wang and Bill Yang. [Why Competition may Discourage Students from Learning? A Behavioral Economic Analysis](#). *Education Economics*, 11(2):117–128, 2003. doi:10.1080/09645290210131656.
- [199] Feifei Xu, Jessika Weber, and Dimitrios Buhalis. [Gamification in tourism](#). In *Information and Communication Technologies in Tourism 2014*, pages 525–537. Springer, 2013. doi:10.1007/978-3-319-03973-2\_38.

- 
- [200] Ibrahim Yildirim. [The effects of gamification-based teaching practices on student achievement and students' attitudes toward lessons.](#) *The Internet and Higher Education*, 33(Supplement C):86 – 92, 2017. ISSN: 1096-7516. doi:<https://doi.org/10.1016/j.iheduc.2017.02.002>.
- [201] Fu-Yun Yu. [Competition within Computer-Assisted Cooperative Learning Environments: Cognitive, Affective, and Social Outcomes.](#) *Journal of Educational Computing Research*, 24(2):99–117, 2001. doi:10.2190/3U7R-DCD5-F6T1-QKRJ.
- [202] Bill Zhou. [Coderally](#), 2013 [visited November 2017].
- [203] MengChu Zhou and Richard Zurawski. [Introduction to Petri Nets in Flexible and Agile Automation](#), pages 1–42. Springer US, Boston, MA, 1995. ISBN: 978-1-4615-2231-7. doi:10.1007/978-1-4615-2231-7\_1.
- [204] Gabe Zichermann and Christopher Cunningham. *Gamification by design: Implementing game mechanics in web and mobile apps*. O'Reilly Media, Inc., 2011. ISBN: 9781449397678.