Experimental Feedback on Prog&Play: A Serious Game for Programming Practice

M Muratet, P Torguet, Fabienne Viallet, Jean-Pierre Jessel

Volume xx (200y), Number z, pp. 1–12

Experimental feedback on Prog&Play: a serious game for programming practice

M. Muratet[1,3], P. Torguet[1,3], F. Viallet[2,3] and J.P. Jessel[1,3]

[1]IRIT/VORTEX
[2]CREFI-T/DiDiST
[3]Université Paul Sabatier, Toulouse, France

Abstract

This paper presents an experimental feedback on a serious game dedicated to strengthening programming skills. This serious game, called Prog&Play, is built on an open source real-time strategy game. Its goal is to be compatible with different students, teachers and institutions. We based its evaluation on an iterative process that allows to implement the game and carry out experimentations in several contexts. Through this assessment, we define a framework which has been tested by third parties and we analyse both positive and negative points in order to improve the project. Evaluation is indeed beneficial and enables you to establish communication about the implemented practices.

Categories and Subject Descriptors (according to ACM CCS): K.3.2 [Computers and Education]: Computer and Information Science Education—Computer science education I.3.0 [Computer Graphics]: General—

1. Introduction

In many countries, students are becoming less and less interested in science. In computer science, for example, according to Crenshaw et al [CCMT08] and Kelleher [Kel06], the number of students has been shrinking. Moreover, "*colleges and universities routinely report that 50% or more of those students who initially choose computer science study soon decide to abandon it*" [ACM05, p. 39]. Our university has been experiencing the same phenomenon with a decrease of 16.6% over the last four years in students studying computer science.

In order to find a solution to this problem, we bet on serious games. Since the first boom in video games in the 1980s, the gaming industry has held an important place in the world market. According to the Entertainment Software Association figures [Ent09], in 2008 the market of U.S. computer and video games amounted to $11.5 billion. This has overwhelmed the U.S. movie market [Num] ($10 billion in 2008). Students currently in universities were born in the video games era. Thus, those games are as much a part of their culture as TV, movies or books. We think that the use of video games technologies in a serious game context to practice programming could be a solution to attract and retain students in computer science. Currently, serious games exist in several fields such as education, government, health, defence, industry, civil security and science. Let us first define a serious game.

For Zyda [Zyd05], a serious game is "*a mental contest, played with a computer in accordance with specific rules, that uses entertainment to further government or corporate training, education, health, public policy, and strategic communication objectives.*" Serious games use entertainment to pursue different learning objectives. For example: "Darfur is dying" [Dar] tries to raise public awareness; "Tactical Language & Culture" [Tac] aims to teach foreign languages and cultures; "America's Army" [Ame] tries to recruit young people for the US Army. Serious games should be created according to the needs and expectations of different sectors, and within the available resources (physical and financial) for their implementation.

1.1. Related work

Learning programming is the basis of computer science training and an important topic in many scientific courses. However, programming fundamentals are hard to learn, especially for novices. Several approaches exist to motivate

submitted to COMPUTER GRAPHICS *Forum* (8/2010). *The definitive version is available at diglib.eg.org and www.blackwell-synergy.com.*

students. For example, Stevenson and Wagner [SW06] analyse assignments from textbooks and historical usage to look for students' problems and propose a "good programming assignment" in computer science. These exercises could be completed by using block-based graphical languages like Scratch [MBK*04], StarLogo [KY05] or Alice2 [KCC*02]. Indeed, these novice-programming environments allow students to forget syntax and directly experiment with programming.

Another approach uses video games in order to hook the player and bring him/her into programming. Two uses have been experimented: implementing new video games and playing video games. For example, Chen and Cheng [CC07] asked students to implement in C++, through a collaborative project, a small-to-medium scale interactive computer game in one semester, using a game framework. Gestwicki and Sun [GS08] based a case study on EEClone. This game is an arcade-style computer game implemented in Java: students analyse various design patterns within EEClone, and from this experiment, learn how to apply design patterns in their own game software. Leutenegger and Edgington [LE07] used a "Game First" approach to teach introductory programming. These authors believe that game programming motivates most new students. They have used 2D game development as a unifying theme.

Another solution is to let students learn when they play a game. Wireless Intelligent agent Simulation Environment (WISE) [CHYH04] combines activities from virtual and physical versions of the Wumpus World game. It enables physically distributed agents to play an interactive game and provides a dynamic learning environment that can enhance a number of computer science courses: it can be used as a medium to demonstrate techniques in lectures; during classes, students can carry out practical work that tests, expands, or modifies the simulator. The Wumpus World game can be played cooperatively or competitively.

Robocode [Roba] is a Java programming game, where the goal is to develop a robot battle tank to fight against other tanks programmed by other players. It is designed to help people learn Java programming. The robot battles are running in real time and on-screen. It is suitable for all kinds of programmers from beginners (simple robot behaviour can be written in just a few minutes) to experts (perfecting an AI - Artificial Intelligence - can take months). Other such games are Marvin's Arena [Mar] using any .NET compatible language, Gun-Tactyx [Gun] using SMALL and Robot Battle [Robb] using a specific script language.

Colobot [Col] is the only example we know of a complete video game which mixes interactivity, storytelling and programming. In this game, the user must colonise a planet using some robots that s/he is able to program in a specific object-oriented programming language similar to C++.

1.2. Overview

Our approach consists in reusing existing games as the basis of a serious game and making it compatible with a maximum of teaching contexts. There are many open source video game projects available on the Internet. We think that reusing them offers advantages in playing and robustness. In order to build our serious game, we used an open source real-time strategy game. Building an efficient tool for a specific teaching course is interesting, and it can often be reused in other contexts. Our approach consists in working on a serious game compatible with different students, teachers and institutions. This serious game is called Prog&Play and is introduced in section 2.

The evaluation of this type of tools in real contexts is a complex task mainly because of the large number of uncontrollable parameters. An iterative evaluation is proposed (section 3) and results are presented in the context of several experimentations.

2. Prog&Play

We consider Prog&Play as a serious game dedicated to programming practice. Prog&Play is based on an open source real-time strategy (RTS) game called *Kernel Panic* [Ker]. *Kernel Panic* uses computer science metaphors, like bits and pointers, as units (i.e. graphical objects which are controlled by the player). It is a simplified RTS with the following features: there is no resource management except for time and space; all units are free to create; it has a small technology improvement tree with fewer than ten units; and it uses low-end vectorial graphics which match the universe. Owing to these characteristics, differences between two players are about strategies and tactics used (and not about knowledge of units features and relative advantages). Thus, the game is action-oriented while always remaining user friendly.

Kernel Panic takes place inside a computer where players command one of the three available factions: "Systems", "Hackers" and "Networks". Each of them offers units, like Bits, Bytes and Assemblers for the System side, Virus, Bugs and Worms for the Hacker side and Ports, Firewalls and Packets for the Network side. Figure 1 shows the hierarchy of unit development for System factions. The Kernel (which is the main unit of Systems) can build Bits, Bytes, Pointers and Assemblers. The latter can build Sockets (which can solely build Bits) and Terminals.

Starting with *Kernel Panic*, we designed an applicative programming interface that enables students to interact with the game through programming (subsection 2.1). The functional architecture of this library is detailed in subsection 2.2. Following these technical details, we point out two solutions to map learning objectives into the game and we detail the one we used during experimentations in section 2.3.

Table 1: *specification of the Prog&Play API.*

Operators	Descriptions
Open	Opens Prog&Play API
Refresh	Loads the last state of the game
Close	Shuts down and cleans up the Prog&Play API
Changeover	Indicates if the game is over or not
Map Size	Returns the map size
StartPosition	Returns the player starting position on the map
NumSpecialAreas	Returns the number of special areas on the map
SpecialAreaPosition	Returns the position of a specific special area
Resource	Returns the level of a specific resource
NumUnits	Returns the number of visible units being a member of a specific coalition
GetUnitAt	Returns a specific unit which is a member of the indicated coalition
GetUnitCoalition	Returns the coalition of a specific unit
GetUnitType	Returns the type of a specific unit
GetUnitPosition	Returns the position of a specific unit
GetUnitHealth	Returns the health of a specific unit
GetUnitMaxHealth	Returns the maximum health of a specific unit
GetUnitGroup	Returns the group of a specific unit
GetUnitNumPendingCommands	Returns the number of pending commands of a specific unit
GetUnitPendingCommandAt	Returns a specific pending command which is associated to the indicated unit
SetUnitGroup	Defines the group of a specific unit
SetUnitAction	Defines an action for a specific unit

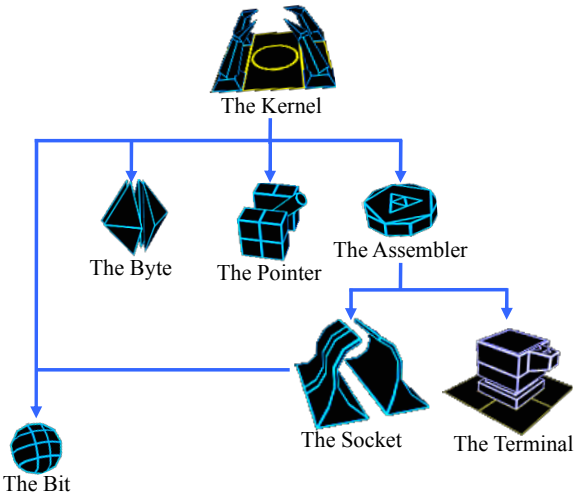


Figure 1: *Units development hierarchy for System factions.*

2.1. Applicative Programming Interface

In RTSes, a player gives orders to his/her units to carry out operations (i.e. moving, building, and so forth). Typically, these instructions are given by clicking on a map with the mouse. We modified the game to allow the player to give these instructions through a program. Thus, students interact with the game using the Prog&Play Applicative Programming Interface (API). This API simplifies programming as much as possible. It hides the game synchronisation complexity and gives access to a sub-set of the game data. Table 1 details its specification. Using this API, programs can load game data like unit features (such as number, position and type), map size, etc. Using these data, the player program can create a set of commands and send it to the game. When the game receives these commands, it executes them, modifying the game state.

The specification detailed in table 1 uses concepts called coalition and special areas. Three coalitions are available in order to structure units: MY_COALITION representing units controlled by the player, ALLY_COALITION representing units controlled by allies of the player and ENEMY_COALITION representing units controlled by enemies of the player. Special areas allow transferring a set of positions through the API. With *Kernel Panic* these special areas are used to specify zones where factory building is allowed.

The Prog&Play API is available in different programming languages: C (Appendix A gives a program example), C++, Java, OCaml, Ada, Scratch and an interpreted language called “Compalgo” (used in a specific course at our university). Thus, the serious game is adaptable to teaching choices and is usable in different teaching approaches (imperative, object-oriented, functional and graphical).

2.2. Functional architecture

From a technical point of view, communication between the student program and the video game is carried out through a shared memory. Figure 2 shows the functional architecture of the Prog&Play API which is an interface to manage this shared memory in order to simplify communication and synchronisation between student programs and *Kernel Panic*. Two interfaces are available: the "Client" is used by the player to code his/her programs and the "Supplier" is integrated into the game engine. On student's program demand, pertinent data of the game are copied into the shared memory. To avoid incoherent situations, the student's program works on this copy. The student's program reads data and writes commands through the "Client" interface. The shared memory is regularly checked by the game engine to carry out pending commands. With this solution, at any time, the player can stop his/her program execution, modify it and execute it again. Thus, the program is able to connect to the shared memory without disturbing the game simulation.

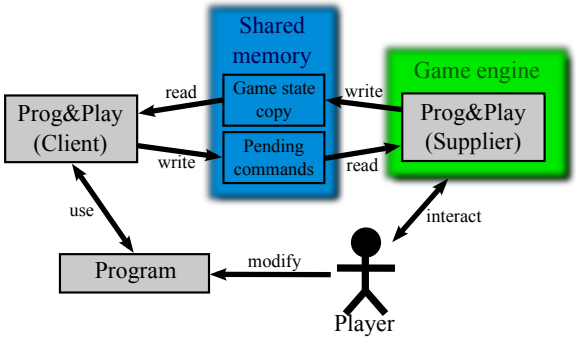


Figure 2: *Functional architecture of the Prog&Play API.*

The "Supplier" is responsible for shared memory management. First, the "Supplier" creates and initialises the shared memory so that it is available to the "Client". After this initialisation, the game enters the simulation loop and for each iteration consults the shared memory to know if an update is required. If this is the case, the "Supplier" computes the update according to the current game state and copies this update into the shared memory. Then, the "Supplier" processes pending commands defined by the "Client". Finally, when the game is over, the "Supplier" frees and cleans up the shared memory.

On the "Client" side, calling API's operators can be carried out after a first call to the *Open* and *Refresh* operators. If there is a problem, API's operators return an error code or an exception depending on the language used. When the "Client" asks for a refresh, this call is blocking until the "Supplier" has detected and carried out the update.

2.3. Mapping learning objectives into the game

In our previous paper [MTJV09], we identified two solutions to map learning objectives into the game. First a campaign can be used, divided in missions (equivalent to exercises), to gradually introduce learning topics and enable students to learn how to play and program AIs (student motivation is maintained by the campaign story). Second skirmishes can be used as a project approach where students program their own AIs in order to use them in a multiplayer session (the student motivation is maintained by competition between players).

Currently, we have only tested the first solution (i.e. the campaign). As the original *Kernel Panic* was only a multiplayer game and did not provide campaigns, we had to build one dedicated to our educational objectives. We have taken advantage of the *Kernel Panic* universe and offered students the following scenario: "*For a certain number of years, a secret war has been rife inside computers. Steady attacks have been led against innocent victims. Today is your turn. Your aggressor captured your mouse controller. You must recover it. Your only solution: programming*". To achieve this objective, five missions were created. During the first, the player controls only one Bit and has to move it to a specific position in order to retrieve a lost Byte. In the second mission, the player controls two units (one Bit and one Byte) and has to place them in two different positions in order to find more units. Thus, in this mission, the conditional control structure is introduced to give a target position to each unit according to their types (Bit or Byte). In the third mission, the player controls a weak army which is made up of seven Bits and three Bytes. In order to strengthen his/her army, the player has to move it to meet up with an Assembler. To do this, the iterative control structure is introduced to loop through each unit and move them towards the right position. In mission four, the player has to use the Assembler capability to repair all the army. This is the most complex mission and it requires overlapping iterative and conditional control structures. Finally, during the last mission, the player launches an attack against opponents in order to retrieve the mouse controller.

Thus, the serious game presented in this paper is a combination of three entities (see figure 3): the first one is a standard video game (*Kernel Panic*); the second one, the Prog&Play API, provides the programming facilities; the third one, the game mode (campaign or skirmish), transforms the programming game into a pedagogical artefact, i.e. a serious game. As each entity can be changed as long as it interfaces with the two others, new versions of the serious game can be built. For example we can replace *Kernel Panic* with another game, create new campaigns adapted to object-oriented programming teaching or work with different programming languages.

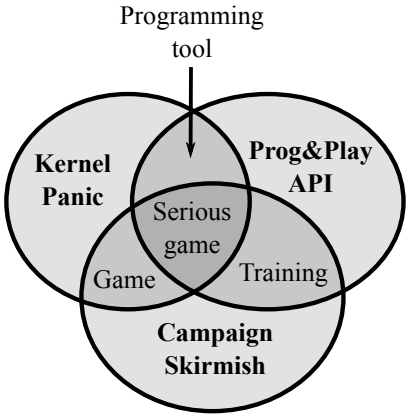


Figure 3: *Serious game composition.*

3. Evaluation

The conception of an evaluation to assess the positive or negative impacts of our serious game on students is a complex task. This is due to the large number of uncontrollable parameters inherent to experiments in a real context. Nevertheless, an evaluation is necessary for several reasons. First, it allows defining a framework which will be tested by third parties in different contexts. Second, results of evaluation show negative points and mistakes and will be analysed in order to improve pedagogical aspects of our serious game. Indeed, evaluation is beneficial and allows communication about the implemented practices.

In subsection 3.1 we define the methodology of design experiments that serves as theoretical framework. Then, we present in subsection 3.2 an iterative implementation of this framework. Finally, we define evaluation criteria and we give experimentation results in subsection 3.3.

3.1. Theoretical framework

We propose to study whether serious games could be useful to teach programming, to attract and retain computer science students. The question is: Is it interesting to use a serious game to teach programming?

To achieve this goal, we propose to use the methodology of design experiments [CCD*03]: "*prototypically, design experiments entail both "engineering" particular forms of learning and systematically studying those forms of learning within the context defined by the means of supporting them. This designed context is subject to test and revision, and the successive iterations that result play a role similar to that of systematic variation in experiment*". The aim of this methodology in educational research is to investigate the possibilities for educational improvement by bringing about new forms of learning in order to study them. Because designs are typically test-beds for innovation, the nature of the methodology is highly interventionist, involving a research team, one or more teachers, at least one student and possibly school administrators. Design contexts are conceptualised as interacting systems and are implemented with a hypothesised learning process and the means of supporting it. Design experiments are conducted in a limited number of settings and aim to develop a relatively humble theory which targets a domain specific learning process. To prepare a design experiment, the research team has to define a theoretical intent and specify disciplinary ideas and forms of teaching which constitute the prospective goals or endpoints for student learning. The challenge is to formulate a design which embodies testable conjectures about both significant shifts in student learning and the specific means of supporting those shifts. In our experiments, the theory we attempt to develop is the process of learning programming through serious games.

3.2. Iterative evaluation

The design experiment is based on an iterative process which allows the serious game and its evaluation to evolve. Each iteration is made up of three stages: preparing for a design experiment, conducting a design experiment and conducting retrospective analysis. In the first stage, the theoretical intent has first to be clarified (what is the point of this iteration? What will be tested?). After this we need to select a course to conduct experimentation corresponding to the objectives of this iteration. Then, according to the conjectured starting points (the environment where Prog&Play will be used), elements of trajectory (how the missions will be fulfilled), and prospective endpoints (the improvement of students' programming skills or motivation), an appropriate design experiment has to be formulated either by us, the teachers or both. Finally, in order to adequately document the learning ecology, we have to choose the evaluation criteria, associated indicators and the way to support them. The second stage carries out the experimentation and collects a maximum amount of data to ensure that retrospective analyses will result in rigorous, empirically grounded claims and assertions. Finally, the third stage analyses experimentation data in order to verify the theoretical intent and optionally proposes new experiments. Three iterations have so far been carried out with Prog&Play.

3.2.1. First iteration

The first iteration's objective is to observe student behaviour in relation to the serious game to determine its motivational potential and influence on teacher activity. To carry out this first iteration, we have chosen to test the serious game in a familiar environment during some practical work. The experimentation took place with first-year students learning computer science at an institute of technology in Toulouse (IUT A). Among 196 students, we chose 15 students from 40 volunteers. Students were novices: at the time of the experiment, they did not know any programming language, except

"Compalgo", an algorithmic language developed by teachers of IUT A. During five sessions of an hour and a half each, students used Compalgo in the Windows environment.

In order to select these students, we designed a questionnaire to evaluate their motivation to play video games and learn programming. We gave priority to students who were not motivated by programming but very motivated by video games. This questionnaire is based on the goals of Viau [Via97]; the value and success expectation model of Bandura [Ban97]; and the causal model of Pintrich and Schunk [PS96]. Each model suggests specific indicators which we adapted to students' motivation in learning programming and practising video games. We drew inspiration from the "motivated strategies for learning" questionnaire [PMB93].

For this experiment, we structured lessons in two phases. In the first, students play the game in a multiplayer session without programming. Thus, they become familiar with the game universe and units. The second phase is a presentation of the Prog&Play API which students learn to use through mission solving. After these two phases, students should be able to achieve small-scale programming compatible with *Kernel Panic*.

3.2.2. Second iteration

The second iteration's objective is to study the implementation of our serious game on a large scale in order to evaluate its influence on students' results and teachers' assessment. As a matter of fact, teachers involved in this experiment were not members of our research team and had no previous experience of serious games. To carry out this second iteration, we introduced Prog&Play in the first semester of the computer science core curriculum (bachelor's degree - Paul Sabatier University). In this course, computer programming is dealt with a functional approach. Students apply basic knowledge of this programming paradigm during six practical work sessions of two hours each using the "OCaml" programming language on Linux.

The class is made up of more than three hundred students structured in practical work groups of about fifteen. Students' distribution into those groups is achieved randomly. About twenty teachers work in this course. To evaluate the influence of the serious game on students' results, half the groups were reference groups which followed the standard teaching whereas the other half used the serious game.

The teaching team's required condition to validate the game's integration into the curriculum concerns the ability of the serious game to preserve basic knowledge processed during standard teaching. Table 2 shows teaching objectives of standard practical work (PW) sessions.

PW sessions one, three and four were modified. After validation by the teaching team, modifications are as follows:

Table 2: *Teaching objectives of the practical work sessions (PW).*

PW	Objectives
1	Working environment presentation: operating system (Linux - Mandriva 2008), window manager (KDE), OCaml interactive loop and word processor (Kate).
2	Arithmetical functions: pattern matching, n-tuple parameter, closure and recursion; introduction to dichotomy.
3	Use of the graphical library: definition of a library, trigonometrical arithmetic, optimisation and recursion.
4	Lists handling: list management and construction; pattern matching and recursion with lists; introduction to predicates; sorting.
5 and 6	Data structures management: reuse of previous functions; projection, doublet removal, selection, update and sorting; user data type manipulation.

- **PW 1**: in addition to the working environment presentation, this first session presents the game universe. An appendix to the teaching documentation describes the process of creating a multiplayer game. Students are invited to test the game by themselves before the third session.
- **PW 3**: the third session is centred on library use. The teaching documentation has been completely rewritten to present the Prog&Play API and the campaign. In order to match this with the original PW, a new mission (called 1a) has been added to the campaign between the first and the second missions. Now, at the end of the first mission, the lost Byte is not present at the indicated position. Thus, in this new mission, additional information is supplied to find the Byte. The player controls only one Bit (the same as in the first mission) and knows the Byte's direction (expressed in degrees, 0 being north) and relative distance. The player has to compute the Byte position according to the above information. During this session, students have to complete missions 1, 1a, 2 and 3. Thus, students learn how to use the library with mission 1, how to carry out a trigonometrical exercise on mission 1a and how to handle pattern matching with mission 2 and recursion in mission 3.
- **PW 4**: finally, the teaching aid of the fourth session has also been rewritten. Here, students work on the fourth mission. They start by creating functions to manage lists of units and use them to solve the mission simply. With sorting algorithms, students code a second, more efficient, solution and glimpse the optimisation principles which was removed from PW 3.

Owing to the number of teachers taking part in this experiment, a teacher training session took place in order to present

the serious game, the teaching aids modifications and solutions to the new exercises.

3.2.3. Third iteration

Prog&Play can be seen as an artefact. We have built a Website where teachers can have access to the serious game and adapt it to their own pedagogy by themselves. The third iteration aims to test the usability of this artefact by third parties.

The first experimentation has been carried out in collaboration with Jérémie Guiochet, teacher at the department of electrical engineering, electronics and industrial informatics at an institute of technology in Toulouse (IUT A). He chose to use the game with students in difficulty. The teacher expected this could motivate them to practice programming. Fifteen students used the C++ language to complete the campaign over nineteen hours.

The second experimentation has been carried out in collaboration with Elisabeth Delozanne and Françoise Le Calvez from Pierre and Marie Curie university (Paris VI). They have deployed the game in a transversal teaching unit based on information and communication technologies in education. In this teaching unit, teachers propose a set of projects, one of which is based on our serious game. Among volunteers, eight students who have followed introductory courses in imperative programming have been retained. Teachers supervised two sessions of two hours in order to introduce the game and programming concepts not studied during lectures (library use and data structures). Following this presentation, the first three missions have been completed using a pure algorithmic language and then coded again using the C language. Then, students had two months to complete the campaign and prepare a presentation of their achievements.

3.3. Evaluation criteria and results

In order to define the evaluation criteria, we summarise objectives revealed from iterations: determine the motivational potential of the serious game; evaluate its influence on students' results and retrieve teachers' views.

From these objectives, we define four evaluation criteria: enhancement of programming skills; system usability; entertainment; and teachers' assessment. The first and third criteria evaluate the "serious game" concept. Indeed, it is fundamental to check the "serious" side (enhancement of programming skills) and the "game" side (entertainment) of Prog&Play. The second criterion allows identifying bottlenecks which could hinder students (installation problems, hardware requirements, software bugs, etc.). The last criterion allows obtaining a qualitative evaluation of the serious game by teachers.

Before presenting each criterion and associated results, we synthesise all the means we have used during experimentations to collect data (see figure 4). The latter operation is organised in three stages: before, during and after the use of the serious game. For example, we massively used questionnaires instead of interviews in order to make data processing easier and reuse them for several iterations. Please note that we did not use all these solutions for each iteration.

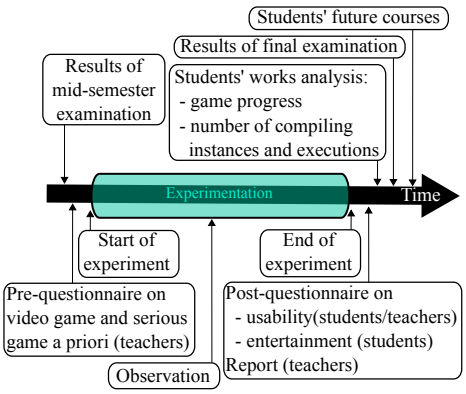


Figure 4: *Synthesis of means used to collect data during experimentations.*

3.3.1. Criterion 1: enhancement of programming skills

The learning evaluation is based on works of Chen and Cheng [CC07], Gestwicki and Sun [GS08] and Leutenegger and Edgington [LE07]. We define three indicators: quantity of work achieved, acquired knowledge and students' future courses. Quantity of work achieved is evaluated using the game progress (number of missions completed). Acquired knowledge is evaluated with students' results of mid-semester and final examinations. These examinations are designed by teachers external to the research team and are the same for all students.

During the first iteration, we analyse the first indicator (quantity of work achieved) by processing data produced by students during experiments. We count the number of missions completed by each student and for each mission the number of compiling instances and executions necessary to perfect their solution. These data have been collected thanks to a software spy integrated into the development environment used during sessions. With these data, we can define a difficulty ratio (noted "d_m") for each mission "m" shown in figure 5. This difficulty ratio is computed as indicated in formula 1, where "$nbStudents_m$" is the number of students who have completed mission "m", "$nbComp_{im}$" and "$nbExec_{im}$" are respectively the number of compiling instances and the number of executions required for the student "i" to complete mission "m".

$$d_m = \frac{\sum_{i=1}^{nbStudents_m} nbComp_{im} + nbExec_{im}}{nbStudents_m} \quad (1)$$

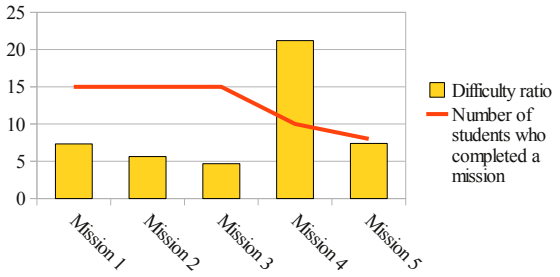


Figure 5: *Difficulty ratio for each mission.*

Contrary to what we expected, the difficulty caused by the various missions is not progressive. There is an important difference in difficulty between the first three missions and the fourth. For future experiments, we plan to split mission 4 into two new missions in order to propose a more progressive challenge. The players getting familiar with the API seems to lessen the difficulty between missions 1 and 3. Finally, we will update mission 5 to offer a better challenge to end the campaign. Indeed several students who reached this mission have been frustrated by its simplicity. Johnson et al. [JVM05] argue that story maintains user interest, encourages the player to progress constantly and links actions and future objectives. Challenges are introduced with the story and must match players' experience. Greitzer et al. [GKH07] highlight the difficulty to build a suitable scenario which is neither too easy nor too difficult in order toubmit a challenge without discouraging the player.

The other indicators (acquired knowledge and students' future courses) have been evaluated during the experimentation of the second iteration. Concerning the "acquired knowledge" indicator, we have collected students' results of mid-semester and final examinations. Figure 6 shows the average marks for reference and serious game groups. Please note that we did not have those marks when we started experimentation.

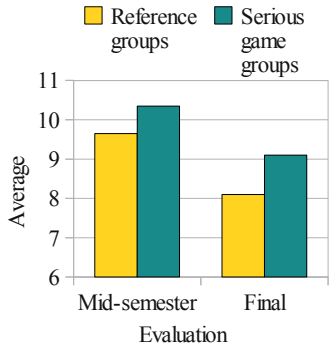


Figure 6: *Average marks for each examination by groups (20 being the top mark).*

First, we notice a decline in marks between mid-semester and final examinations whatever group we take. However, this decline is less important for the serious game groups (1.25 points) than reference groups (1.55 points). We cannot link this result only to the serious game because experimentations have been carried out with a large number of uncontrollable parameters. Nevertheless, we consider that the serious game is partly responsible for this positive result.

Concerning the "students' future courses" indicator, we used students' choices for their second semester. Indeed, after the first semester, nine major courses are available, one of which is dedicated to computer science. Among students from the serious game groups 55% have chosen this major course compared to 49% in the reference groups.

Thus, these results show that even if missions' difficulty is less progressive than it was expected, the campaign supplies a sufficient difficulty to encourage work and stimulate students' interest. From a statistical point of view, we note that the serious game has contributed to reduce students' failure compared to reference groups and to recruit more students for computer science for the second semester.

3.3.2. Criterion 2: system usability

System usability is based on a post-questionnaire. We ask students to evaluate the serious game. Using works of Siang and Rao [SR03], we define questions to evaluate the hierarchy of the players' needs in order to identify which part of our serious game needs improvement. This hierarchy is divided into seven levels and lower levels are to be fulfilled before moving on to the higher levels in the pyramid. The seven levels by priority order are as follows: **Rules need**, players seek information to understand the basic rules of the game; **Safety need**, players need guidance about the game software; **Belongingness need**, players need to become familiar with the game in order to feel capable to achieve objectives; **Esteem need**, players need to be motivated by the game (feedbacks, scores, competition, etc.); **Need to know and understand**, players need to understand and know more information about the game (e.g. different strategies or hidden items) in order to reuse it during play time; **Aesthetic need**, players need good graphics and visual effects, appropriate music, sound effects, etc; **Self actualisation need** players need to be able to transfer their creativity and imagination into the game as long as it conforms to the game rules.

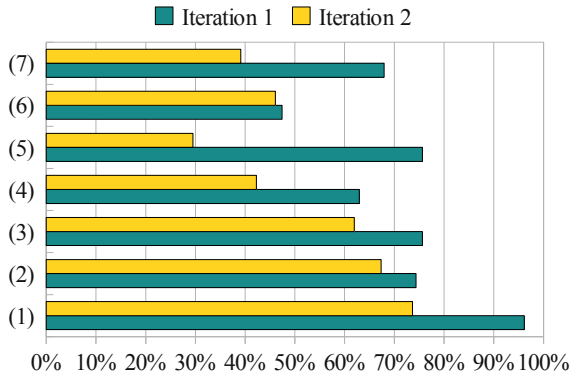
Additionally, we ask students to express their criticisms, remarks and suggestions about the serious game as well as their point of view about the interest of a serious game to learn computer programming. Questions are presented in figure 7.

During the first two iterations, we evaluate the hierarchy of players' needs. Figure 8 shows students' satisfaction for each level of this hierarchy.

Concerning the first iteration, the lowest satisfaction level

- Q1: In your opinion, is it interesting to use a video game to learn how to program? (1="not at all", 7="essential")
- Q2: Do you think that practical work sessions based on a video game are adapted to this course? (1="not at all", 7="perfectly adapted")

Figure 7: *Opinion-related questions in the post-questionnaire.*



(7) Self actualisation need
(6) Aesthetic need
(5) Need to know and understand
(4) Esteem need
(3) Belongingness need
(2) Safety need
(1) Rules need

Figure 8: *Students' satisfaction for the hierarchy of the players' needs.*

concerns the aesthetic need with an average value of 47%. The low-end vectorial graphics of *Kernel Panic* seem disturbing to our students. Nevertheless, this simplified RTS allows a quick learning curve which takes part in the positive satisfaction of lower needs. Therefore, we still think that *Kernel Panic* is not a bad choice to support Prog&Play.

For the second iteration, we observe lower satisfaction rates although the serious game is the same except for the new mission (1a). In order to explain differences with the first iteration, we analyse remarks and criticisms expressed by students. The main criticisms are levelled at the teaching aids that are considered too descriptive and over-prescriptive. We suggest that this feeling has counterbalanced the game understanding. Thus, this experiment highlights that results obtained depend not only on the serious game but also on the implementation, supervision, teaching aids, etc.

Nevertheless, whatever iteration, students have appreciated the initiative and are interested in this kind of pedagogy.

The average and median answers to the two additional questions are presented in table 3 for the first two iterations.

Table 3: *Average and median answers of questions presented figure 7 for the two first iterations (highest possible value being 7).*

	1st iteration		2nd iteration	
	Q1	**Q2**	**Q1**	**Q2**
Average	6.17	6	5.05	5.18
Median	6.83	7	5	5

In conclusion about this criterion, the game is functional because no critical bugs were revealed during experimentations. The only issue is that some teachers have encountered difficulties to configure their operating system to use the serious game correctly. The analysis of the hierarchy of players' needs shows that students' satisfaction heavily depends on implementation. This confirms the importance of supervision for our serious game.

3.3.3. Criterion 3: entertainment

This criterion has been evaluated with a questionnaire distributed to students at the end of experiments. Questions are presented in figure 9.

- Q3: Do you appreciate the campaign story (missions)? (1="not at all", 7="a lot")
- Q4: Do you think that using programming in Kernel Panic increases entertainment?
  - ☐ Reduces entertainment
  - ☐ Do not change entertainment
  - ☐ Enhances entertainment
  - ☐ I don't know

Figure 9: *Questions for the entertainment post-questionnaire.*

The fourth question (Q4, figure 9) was crucial for us. Introducing algorithmic concepts into the game should not make it any less entertaining. This requirement is due to our vision of serious games which have to be, above all, entertaining.

Concerning the first iteration, the average and median answers to the third question (Q3, figure 9) are, respectively, 5.85 and 6. These results show that students liked playing the campaign. We think that missions' interest is partly due to the first phase of the experimentation. The multiplayer session allows students to become familiar with the universe of the game and makes students' immersion into the story easier.

For the fourth question (Q4, figure 9) 100% of students found that using programming in the game enhances entertainment. We found out during this first experiment that the game is fun and rewarding: it is fun because all students

found that programming enhances entertainment; and it is rewarding because even though we initially planned, for the final session, to let students play a normal multiplayer game, more than half of the students preferred to continue programming.

Concerning the second iteration, results are slightly lower. The average and median answers to the third question (Q3, figure 9) are, respectively, 4.14 and 4. This is the same for the answers to the fourth question (Q4, figure 9). 15% of students found that the introduction of programming into the game reduces entertainment, 22% indicate there is no change, 42% identify an enhancement and 21% do not give their opinions. We attribute these results to the same reason identified in criterion 2 analysis: the teaching aids were perceived as too descriptive and over-prescriptive, they have counterbalanced the entertainment.

Thus, the campaign is entertaining and it is a real success from this point of view. Moreover, students appreciate using their programming knowledge in a real context.

3.3.4. Criterion 4: teachers' assessment

In order to observe teachers' activities during experiments, we propose to film sessions. Particular emphasis has been laid on activities dedicated to game usability and teaching contents. The activity called "game usability" concerns explanations on the game's control (how to launch the game? How to move the camera?) and on the game's interaction with students' programs (how to select or command units? How to check if programs are correct?). The activity called "teaching contents" deals with explanations on programming obstacles like variables (types, assignments, records), functions (call, argument passing) and control structure (conditional, iterative). The activity called "other tasks" concerns administrative tasks (e.g. welcoming students, roll call), explanations on operating system usability, etc.

Furthermore, we submit a questionnaire to teachers about their perception of video games and serious games before experimentation. Then, each teacher has to file a report, which enables us to collect their points of view about the past experimentation. On this occasion, we ask them if they encountered difficulties with the serious game and what they would advise regarding a possible renewal of the experiment.

The observation of the teacher's activities has been carried out during the second practical work session of the first iteration. The time spent by the teacher on each activity is detailed in table 4 (please note that some activities are counted in several categories, which explains why the amount of time spent is higher than the duration of a session).

As can be seen in the table, the longest activity deals with "teaching contents". This distribution is appropriate,

Activity	Time spent
Game usability	22m 13s
Teaching contents	1h 3m 42s
Other tasks	41m 27s

Table 4: *Time spent by the teacher on each activity during one session*

because, the game's explanations should not excessively reduce teaching activities. Moreover, the time spent on game-related activities diminishes when students become expert in game handling. Nevertheless, "game usability" is an important activity mainly for the first sessions. Indeed, the second level of the hierarchy of players' needs (safety need) has been resolved, in a large part, by teachers. Currently Prog&Play is a serious game for programming practice; information about programming, learning contents and game usability are not included in the game and must be added through teachers' speeches or teaching aids.

Concerning the second iteration, we take advantage of the important number of teachers by asking them about their perception of video games and serious games before the experimentation. Three women and twelve men answered our questions. Nine people said they played video games. Teachers have mainly pointed out three qualities of such games: they encourage thinking through strategic developments, they are entertaining and allow immersion into virtual worlds. To a lesser extent, competition with multiplayer sessions, storytelling and creativity are identified as intrinsic qualities of video games. The main drawback is addiction. Some teachers find video games too complex and associate them with a waste of time. Concerning serious games, seven teachers who have a positive perception think that they can motivate students to practice computer science activities. Moreover, they consider serious games as concrete tools close to students' interests. The two teachers who have a negative perception fear that serious games misrepresent computer science as only made of video games, give an advantage to boys, are a waste of time (i.e. explaining the game instead of the teaching content) and that playing distracts students from basic knowledge acquisition. The six teachers who had no opinion said they would consider results before expressing their convictions.

Again from the second iteration, teachers' reports are mixed. On the performance side, they thought that sessions based on the serious game have been counter-productive for several students. They do not attribute this consequence to the serious game but to the teaching aids. Like students, they felt this documentation was too descriptive. On the plus side, most teachers have noticed a positive influence of the serious game on students.

Concerning the third iteration, teachers, who have carried out experimentations by themselves, give positive reports.

First, Mr Guiochet said that the serious game campaign was well adapted to his students (they needed basic programming practice and had a lot of time for this course). Indeed, students have shown their interest through some personal work which was not required. Second, Mrs Delozanne and Mrs Le Calvez have been impressed by the wide range of strategies planned and the quality of presentations prepared by students. They report that students have appreciated the projects because they have programmed something concrete. Mrs Delozanne and Mrs Le Calvez added that their sessions preparation time was quite long due to their inexperience of video games. In both instances, teachers plan to use our serious game again next year.

In conclusion about teachers' assessment we can say that globally they are positive about the influence of the serious game on students' work. Indeed, in the second iteration the negative evaluations are mainly attributed to the teaching aids and not to Prog&Play. This is very encouraging for us.

4. Conclusion and future work

This paper describes Prog&Play, a serious game aiming to encourage students to persevere in computer science. Through a theoretical framework, we designed an iterative evaluation based on several experiments. During the first iteration, we observed the serious game used by students in a real context. On this occasion, we identified that Prog&Play is really appreciated by students. Following this first iteration, the portability of the game was questioned. Thus, a second iteration was designed to study the game's implementation on a large scale. Despite poor teaching aids, the serious game has helped reduce students' failure compared to reference groups. Then, the last iteration was meant to let teachers implement the game by themselves in their courses. Reports of these teachers are positive since they plan to continue using Prog&Play.

Thus, the serious game is functional and motivates students. We attribute this success to the original approach of Prog&Play which allows the use of programs to interact with a RTS. The second element taking part in this success is the campaign story which has a twofold objective. Firstly, it motivates the player by making him/her protagonist in the story development. Secondly, it gradually introduces the pedagogical contents of the serious game. At the end of the story, students master the programming interface and are able to program their own AIs.

For pedagogical reasons, we made the Prog&Play API available in different programming languages. Specifically, Scratch compatibility opens up new research vistas. Indeed, it will be interesting to study the impact of these two technologies (block-based graphical languages and serious games) on students' motivation. Another future project consists in adapting the serious game to other RTSes. Indeed, with the quick evolution of video games engines, its integration into new RTSes is essential to continue entertaining students. It would also be interesting to evaluate this approach with another video game genre and to compare it with our RTS based serious game.

The Prog&Play system with Kernel Panic and compatible interfaces are downloadable at `http://www.irit.fr/~Mathieu.Muratet/progAndPlay_en.php`. If you are interested in our serious game, please do not hesitate to get in touch with us!

Acknowledgments

This work would not have been possible without the collaboration of several learning institutions. The authors thank the following people and institutions: Christian Percebois and Max Chevalier (department of computer science) and Jérémie Guiochet and André Lozes (department of electrical engineering, electronics and industrial informatics) at IUT A in Toulouse; Mathias Paulin, Véronique Gaildrat and all Paul Sabatier university teachers who took part in our experimentation; and Elisabeth Delozanne and Françoise Le Calvez at Pierre and Marie Curie university in Paris.

References

[ACM05] ACM/IEEE-CURRICULUM 2005 TASK FORCE (Ed.): *Computing Curricula 2005, The Overview Report*. IEEE Computer Society Press. and ACM Press., New York, 2005. 1

[Ame] AMERICA'S ARMY: http://www.americasarmy.com/. accessed 26 August 2010. 1

[Ban97] BANDURA A.: *Self-efficacy: The exercise of control*. New York: Worth Publishers, 1997. 6

[CC07] CHEN W.-K., CHENG Y. C.: Teaching object-oriented programming laboratory with computer game programming. *Education, IEEE Transactions on 50*, 3 (Aug. 2007), 197–203. 2, 7

[CCD*03] COBB P., CONFREY J., DISESSA A., LEHRER R., SCHAUBLE L.: Design experiments in educational research. *Educational Researcher 32*, 1 (Jan. 2003), 9–13. 5

[CCMT08] CRENSHAW T. L., CHAMBERS E. W., METCALF H., THAKKAR U.: A case study of retention practices at the university of illinois at urbana-champaign. *39th ACM Technical Symposium on Computer Science Education 40*, 1 (2008), 412–416. 1

[CHYH04] COOK D. J., HUBER M., YERRABALLI R., HOLDER L. B.: Enhancing computer science education with a wireless intelligent simulation environment. *Journal of Computing in Higher Education 16*, 1 (2004), 106–127. 2

[Col] COLOBOT: http://www.ceebot.com/colobot/index-e.php. accessed 26 August 2010. 2

[Dar] DARFUR IS DYING: http://www.darfurisdying.com/. accessed 26 August 2010. 1

[Ent09] ENTERTAINMENT SOFTWARE ASSOCIATION: Essential facts about the computer and video game industry. http://www.theesa.com/facts/pdfs/ESA_EF_2009.pdf, 2009. 1

[GKH07] GREITZER F. L., KUCHAR O. A., HUSTON K.: Cognitive science implications for enhancing training effectiveness in a serious gaming context. *J. Educ. Resour. Comput. 7*, 3 (2007), 2. 8

[GS08] GESTWICKI P., SUN F.-S.: Teaching design patterns through computer game development. *ACM Journal on Educational Resources in Computing 8*, 1 (2008), 1–22. 2, 7

[Gun] GUN-TACTYX: http://apocalyx.sourceforge.net/guntactyx/. accessed 26 August 2010. 2

[JVM05] JOHNSON W. L., VILHJALMSSON H., MARSELLA S.: Serious games for language learning: How much game, how much ai? In *Proceeding of the 2005 conference on Artificial Intelligence in Education* (Amsterdam, The Netherlands, The Netherlands, 2005), IOS Press, pp. 306–313. 8

[KCC*02] KELLEHER C., COSGROVE D., CULYBA D., FORLINES C., PRATT J., PAUSCH R.: Alice2: Programming without syntax errors. In *15th annual symposium on the User Interface Software and Technology* (Paris, France, Oct. 2002). 2

[Kel06] KELLEHER C.: Alice and the sims: the story from the alice side of the fence. In *The Annual Serious Games Summit DC Washington* (DC, USA, Oct. 30–31, 2006). 1

[Ker] KERNEL PANIC: http://springrts.com/wiki/Kernel_Panic. accessed 26 August 2010. 2

[KY05] KLOPFER E., YOON S.: Developing games and simulations for today and tomorrow's tech savvy youth. *TechTrends: Linking Research and Practice to Improve Learning 49*, 3 (2005), 33–41. 2

[LE07] LEUTENEGGER S., EDGINGTON J.: A games first approach to teaching introductory programming. *SIGCSE '07: Proceedings of the 38th SIGCSE technical symposium on Computer science education 39*, 1 (Mar. 2007), 115–118. 2, 7

[Mar] MARVIN'S ARENA: http://www.marvinsarena.com/. accessed 26 August 2010. 2

[MBK*04] MALONEY J., BURD L., KAFAI Y., RUSK N., SILVERMAN B., RESNICK M.: Scratch: A sneak preview. In *2nd International Conference on Creating Connecting, and Collaborating through Computing* (Keihanna-Plaza, Kyoto, Japan, Jan. 2004), pp. 104–109. 2

[MTJV09] MURATET M., TORGUET P., JESSEL J.-P., VIALLET F.: Towards a serious game to help students learn computer programming. *Int. J. Comput. Games Technol. 2009* (2009), 1–12. 4

[Num] The Numbers: http://www.the-numbers.com/market/. accessed 26 August 2010. 1

[PMB93] PINTRICH P., MARX R. W., BOYLE R. A.: Beyond cold conceptual change: the role of motivational beliefs and classroom contextual factors in the process of contextual change. *Educational Research 630*, 2 (1993), 167–199. 6

[PS96] PINTRICH P. R., SCHUNK D. H.: *Motivation in Education: theory, research and applications*. Englewood Cliffs : Prentice Hall, 1996. 6

[Roba] ROBOCODE: http://robocode.sourceforge.net/. accessed 26 August 2010. 2

[Robb] ROBOT BATTLE: http://www.robotbattle.com/. accessed 26 August 2010. 2

[SR03] SIANG A., RAO R. K.: Theories of learning: a computer game perspective. In *Multimedia Software Engineering, 2003. Proceedings. Fifth International Symposium on* (Dec. 2003), pp. 239–245. 8

[SW06] STEVENSON D. E., WAGNER P. J.: Developing real-world programming assignments for cs1. In *ITICSE '06: Proceedings of the 11th annual SIGCSE conference on Innovation and technology in computer science education* (Bologna, Italy, June 2006), pp. 158–162. 2

[Tac] TACTICAL LANGUAGE AND CULTURE: http://www.tacticallanguage.com/. accessed 26 August 2010. 1

[Via97] VIAU R.: *La motivation en contexte scolaire*. Bruxelles : De Boeck, 1997. (in French). 6

[Zyd05] ZYDA M.: From visual simulation to virtual reality to games. *IEEE Computer 38*, 9 (2005), 25–32. 1

Appendix A: Example solution

We present in figure 10 a solution (in C) to the following scenario: "*You need to find an allied unit near to the position of your unit. Tracks indicate that it has moved away at a distance of 1060 units and an orientation of 209 degrees.*" In this context, we have used the native Prog&Play API. In "C", a unit is an abstract type that can be consulted with a set of functions. For example, the function *PP_Unit_GetPosition(u)* allows getting the position of a unit. With the assistance of teachers, students have to find in the documentation which function is useful to complete this exercise.

```
01 - #include "PP_Client.h"
02 - #include "constantList_KP3.1.h"
03 - #include <math.h>
04 -
05 - int main (){
06 -  PP_Unit u;
07 -  PP_Pos unitPos, targetPos;
08 -  double radianAngle;
09 -
10 -  PP_Open(); /* Open the game API */
11 -  PP_Refresh(); /* Refresh game state */
12 -  u = PP_GetUnitAt(MY_COALITION, 0); /* Get my first unit */
13 -  /* Compute target position */
14 -  unitPos = PP_Unit_GetPosition(u); /* Get unit's position */
15 -  radianAngle = 3.14159265*(209)/180;
16 -  targetPos.x = unitPos.x + cos(radianAngle) * 1060;
17 -  targetPos.y = unitPos.y - sin(radianAngle) * 1060;
18 -  /* Order the unit to move to the position */
19 -  PP_Unit_ActionOnPosition(u, MOVE, targetPos);
20 -  PP_Close(); /* Close the game API */
21 - }
```

Figure 10: *A solution written in* C