Experimental Feedback on Prog&Play, a Serious Game for Programming Practice

Mathieu Muratet, Patrice Torguet, Fabienne Viallet, Jean Pierre Jessel

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

Mathieu Muratet[1,3], Patrice Torguet[1,3], Fabienne Viallet[2,3] and Jean-Pierre 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 realtime strategy game. Its goal is to be compatible with different students, teachers and institutions. We propose an iterative evaluation in order to improve it while experiments are running. Through this evaluation, we define a framework that will be tested by third parties in different contexts and we analyse negative points and mistakes in order to improve the project. In every instance, evaluation is beneficial and allows establishing a 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 interested in science. In computer science, for example, according to Crenshaw et al [CCMT08] and Kelleher [Kel06], the number of students is 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 experiences 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 of video games in the 80s, 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 amount to $11.5 billion. This is past the U.S. movie market [The] ($9.8 billion in 2008). Students currently in universities were born in the video games era. And 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 keep students in computer science.

1.1. Related work

Learning programming is the basis of computer science training and an important topic in many scientific trainings. However, programming fundamentals are hard skills to learn, especially for novices. In order to help students, several approaches exist to motivate them. For example, Stevenson and Wagner [SW06] analyse assignments from textbooks and historical usage to look for student's problems and propose a "good programming assignment" in computer science. These exercises could be completed by using block-based graphical languages like StarLogo [KY05], Scratch [MBK*04] or Alice2 [KCC*02]. 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 to programming. Two uses have been experimented: implementing new video games and playing video games. For example, Chen and Cheng [CC07] ask 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] have 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 softwares. Leutenegger and Edgington [LE07] use a "Game First" approach to teach introductory programming. These authors believe that game programming motivates most new programmers. They use 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 allows 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 for demonstrating techniques in lectures; in the classes, students can work on laboratory exercises that test, expand, or modify 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 battle against other tanks programmed by other players. It is designed to help people learn Java programming. The robot battles are running in realtime and on-screen. It is suitable to all kind of programmers from beginners (a simple robot 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 that we know, of a complete video game, which mixes interactivity, storytelling and programming. In this game, the user must colonize 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 living on the Internet. We think that reusing them gives advantages in playing and robustness. In order to build our serious game to practice programming, we used an open source realtime strategy game. Building an efficient tool for a specific teaching course is already an interesting work, but often it could be reused in peripheral 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 quickly introduced in section 2.

The evaluation of this type of tools in real contexts is a complex task in particular due to the large number of uncontrollable parameters. We propose an iterative evaluation detailed in section 3. This evaluation is discussed in the context of an experiment in order to improve it for future experiments.

2. Prog&Play

We consider Prog&Play as an example of a serious game dedicated to programming practice. Currently, serious games exist in several fields such as education, government, health, defence, industry, civil security and science. However, what is a serious game?

2.1. Serious Games

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.*" Thus, we consider as serious games, any video game built to differ from pure entertainment. Serious games use entertainment to pursue different learning objectives: "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 to join the US Army. Serious games must be created according to the needs and expectations of different working sectors and the public, and within the available resources (physical and financial) for their implementation.

2.2. Overview of Prog&Play

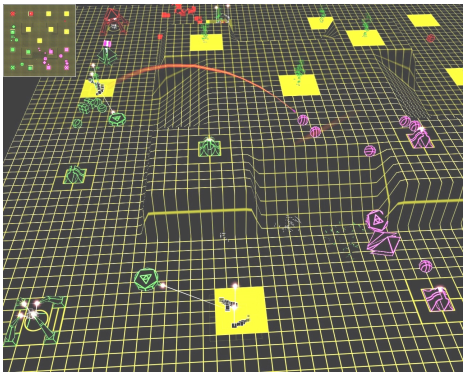

Figure 1: *Kernel Panic.*

Prog&Play is based on an open source realtime strategy (RTS) game called *Kernel Panic* [Ker]. Figure 1 presents a screenshot of *Kernel Panic*. *Kernel Panic* uses computer science metaphors, like bits and pointers, as units. 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 less than ten units; and it uses low-end vectorial graphics, which match the universe. These characteristics emphasize strategy

and tactics in an action-oriented game while always remaining user friendly. In RTSs, 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. This program can load game data like unit features (number, position, 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.

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. We propose several versions of this API in different programming languages: C (Appendix A gives an example), C++, Java, OCaml, Ada 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 contexts.

In our previous papers [MTJV09], we identified two solutions to map learning objectives into the game: **using a campaign, divided in missions** (equivalent to exercises), to gradually introduce learning topics and enable students to learn how to play and to program AIs (the student motivation is maintained by the campaign story); and **using skirmishes, 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). 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.

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 experiements in a real context. Nevertheless, an evaluation is necessary for several reasons. First, it allows defining a framework that 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 the project. Indeed, evaluation is beneficial and allows establishing a communication about the implemented practices.

In subsection 3.1 we define the methodology of design experiments that serves us as theoretical framework. Then, we present in subsection 3.2 specifications of our evaluation and we discuss it in subsection 3.3 with an analysis of a first experiment. Finally, from this analysis, we propose in subsection 3.4 some improvement for future experiments.

3.1. Theoretical framework

We propose to study if serious games, which can be collaborative learning games, could be useful in order to teach programming, and to attract and keep computer science students. The question is: Is it interesting to use a serious game for teaching programming?

To achieve this goal, we propose 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 intent 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 eventually school administrators. Design contexts are conceptualized as interacting systems and are implemented with a hypothesized learning process and the means of supporting it. Although design experiments are conducted in a limited number of settings, they aim to develop a relatively humble theory that target 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 that constitute the prospective goals or endpoints to student learning. The challenge is to formulate a design that 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. Evaluation criteria

In order to define the evaluation criteria, we recall objectives of our projet: "*determine the efficiency of our serious game for learning programming. This serious game must be usable in various training groups (academic or professional) without modifying educational choices of institutions (programming languages, operating systems...). The serious game is used in addition to classic lessons.*"

From these objectives, we define three evaluation criteria: learning programming; entertainment; and system usability. The first two criteria evaluate the "serious game" concept. Indeed, it is fundamental to check the "serious" side (learning programming) and the "game" side (entertainment) of Prog&Play. The last criterion allows identifying bottlenecks that would disturb students.

We design a temporal schedule of evaluation (see fig-

ure 2). This evaluation is organized in three steps: before, during and after the use of the serious game.

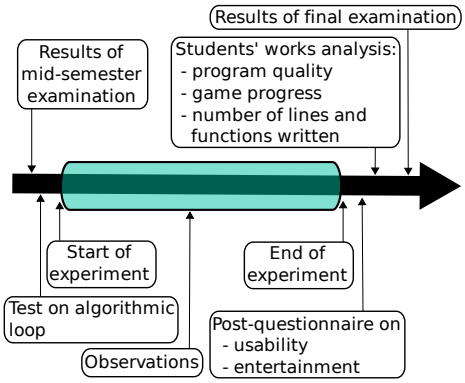

Figure 2: *Temporal schedule of evaluation.*

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: acquired knowledge; program quality and quantity of work achieved. Acquired knowledge will be evaluated with students' results of mid-semester and final examinations. These examinations are designed by external (to the research team) teachers and are the same for all students. Program quality will be evaluated with criterions of Smith and Cordova [SC05] like program correctness, programming style, program documentation and design documentation. Quantity of work achieved will be evaluated using the game progress (number of missions completed) and the number of code lines and functions written.

The usability and the entertainment evaluation are based on a post-questionnaire. We ask students to evaluate the serious game and give their criticisms, remarks and suggestions. Using works of Siang and Rao [SR03], we include questions to evaluate the hierarchy of the players' needs in order to identify which part of our serious game needs improvement.

In addition to these indicators, we plan to observe teacher and student activities during experiment. For teachers, we study the duration of teaching dedicated to the game and the one dedicated to the knowledge. For students, we observe his/her activities related to gaming and programming.

Finally, we design a test based on the algorithmic loop manipulation in order to determine the profile of students. These data will allow us to compare results of several experiments on students' skill levels.

3.3. Implementation and analysis

This experiment 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.

For this experiment, we structured lessons in two phases. During the first phase, **students play the game** in a multiplayer session without programming. They familiarize themselves with the game universe and units. The second phase is a presentation of the Prog&Play API. Students learn to use it through **missions solving** after a presentation of the API by the teacher. After these two phases, students should be able to do small-scale programming compatible with *Kernel Panic*.

In order to select these students, we designed a new questionnaire to evaluate their motivation to play video games and learn programming. We selected in priority students who are 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 practicing video games. We inspired ourselves from the "motivated strategies for learning" questionnaire [PMB93].

During this experiment, we filmed teacher activities for one session. We identified activities dedicated to game usability, teaching contents and other tasks. The "game usability" activity concerns explanations on game's control (How to launch the game? How to move the camera?) and on game's interaction with students' programs (How to select or command units? How to check if programs are correct?). The "teaching contents" activity concerns explanations on programming obstacles like variables (types, assignments, records), functions (call, argument passing), control structure (conditional, iterative). The "other tasks" activity concerns administrative tasks (students welcome, roll call), explanations on operating system usability, etc. The time spent by the teacher on each activity is detailed in table 1 (Note that some teacher speeches are counted in several activities, which explains why the sum of time spent is higher than the duration of a session).

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

Table 1: *Time spent by the teacher on each activity during one session*

3.3.1. Learning programming

The first analysis is based on some algorithmic loop manipulation. We have built this test from Ginat's work [Gin04]. This knowledge is a central point of programming. We ask students eight progressive questions. They have to answer in twenty minutes, without documentation and on a paper sheet. This test was proposed after the gaming phase and its rating was not taken into account for obtaining a degree.

Students solutions have been analysed with criteria from Smith and Cordova [SC05]. We keep from these criteria "Correct Output", "Coding style" and "Neatness/Clarity". For the "Correct Output" we check if student programs produce expected output with no error, many errors, no output at all or completely incorrect output. For the "Coding style" we check if identifiers are descriptive and follow naming conventions and if program constructs are simple and elegant. For the "Neatness/Clarity" we check indentation, comments and code readability. We add to these criteria an analysis of students' difficulties on variables, loops and subroutines.

In order to evaluate pertinence of this test, we compare students' results to this test with those they got to their mid-semester algorithmic examination. The latter examination is about basic algorithmic concepts like variables, types, assignments, functions, parameter passing, records and conditional and interative control structure. Comparison is shown in figure 3 and gives for each of the fifteen students their grades (ranging from 0 to 20) for the Prog&Play preliminary evaluation and the mid-semester examination. Please note that both evaluations are carried out by students before experiment. The objective of this comparison is to check if our evaluation reflects students skill level. This seems to be the case with a mean difference of 2.06 points between these two evaluations for the fifteen students. Future experiments will take place with different institutions, students and teachers. We need an identical comparison basis for every experiment. We plan to use this algorithmic loop manipulation test as this basis. This will be performed by comparing the results of future experiments and taking account of students' skill levels evaluated before each of these experiments.

A second analysis concerns the processing of data produced by students during experiment. We count the number of missions completed by each student and for each mission the number of compiling and executions necessary for perfecting their solution. With these data we can define a difficulty ratio for each mission shown in figure 4. Missions and teaching content are more detailed in a previous paper [MTJV09].

In contrary to what we expected, missions' difficulty is not progressive. There is an important difficulty difference between the first three missions and the fourth mission. For future experiments, we plan to split mission 4 into two new missions in order to propose a more progressive challenge. The lessening of difficulty between missions 1 and 3 is credited to the appropriation of the API. Finally, we will update mission 5 to propose a better challenge for ending the campaign. Indeed several students who reached this mission have been frustrated by its simplicity.

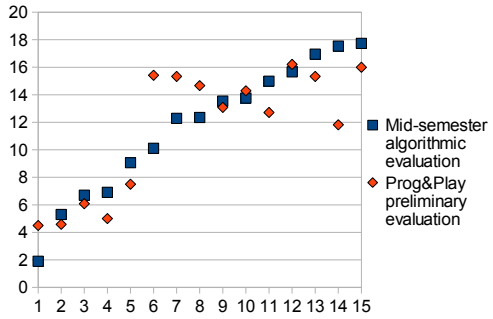

Figure 3: *Comparison of students' results at the Prog&Play preliminary evaluation and at the mid-semester algorithmic examinations.*

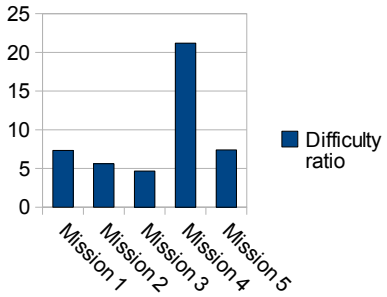

Figure 4: *Difficulty ratio for each mission.*

To evaluate game impact on students' acquired knowledge, we plan to compute the evolution of students' results between mid-semester and final examinations and to compare this evolution with students who did not use Prog&Play. Owing to the small population of this first experiment, we cannot compare these two populations statistically. New experiments will increase our sample groups in order to carry out quantitative analyses and robust statistics.

3.3.2. Entertainment

Some questions dealing with the entertainment post-questionnaire are presented in figure 5.

The average and median answer to the first question are, respectively, 5.85 and 6 which 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 appropriate the universe of the game and makes students immersion into the story easier.

The second question was crucial for us. Introducing algorithmic concepts into the game do not reduce entertainment.

- Q1: Do you appreciate the campaign story (missions)? (1="not at all", 7="a lot")
- Q2: Do you think that using programming in Kernel Panic increases entertainment?
  - ☐ Reduces entertainment
  - ☐ Do not change entertainment
  - ☐ Increases entertainment
  - ☐ I don't know

Figure 5: *Relevant questions for entertainment post-questionnaire.*

This requirement is due to our vision of serious games that have to be above all entertaining. 100% of students found that using programming in the game increases entertainment. We attribute this success to the original approach of Prog&Play, which enables to use programs to interact with a RTS. In addition to this novelty, students appreciate using their programming knowledge in a real context. The second element taking part in this success is the campaign story, which has a double objective. Firstly, it **motivates** the player making him/her actor/actress of the story development. Secondly, it **introduces gradually** the pedagogical contents of the serious game. At the end of the story, students master the programming interface and are pleased to start the last session, which consists in programming their own AIs. Johnson et al. [JVM05] argue that story maintains user interest, encourages player to progress all the time and links together actions and future objectives. Challenges are introduced with the story and must match player experience. Greitzer et al. [GKH07] highlight the difficulty to build a suitable scenario, neither too easy nor too difficult, in order to submit a challenge without discouraging the player.

We found out during this first experiments that the game is fun and rewarding: fun because all students found that programming increases entertainment; and 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.

3.3.3. System usability

First, we note that Prog&Play is functional because no critical bugs were revealed during experiment. Nevertheless, reliability is not sufficient and we evaluate the hierarchy of the players' needs [SR03]. This hierarchy is divided into seven levels where lower levels are to be fulfilled before moving to the higher levels in the pyramid. The seven levels by priority order are as follows: **Rules need**, players are seeking for information to understand the basic rules of the game; **Safety need**, players need helping information; **Belongingness need**, players need to know it is possible to win; **Esteem need**, players need to be in possession and have full control over the game; **Need to know and understand**, players need to understand and know more about the game such as different strategies, hidden items, etc.; **Aesthetic need**, players need good graphics and visual effects, appropriate music, sound effects, etc; **Self actualisation need** players want to be able to do anything as long as it conforms to the game rules. Figure 6 shows students satisfaction for each level of this hierarchy in our serious game.

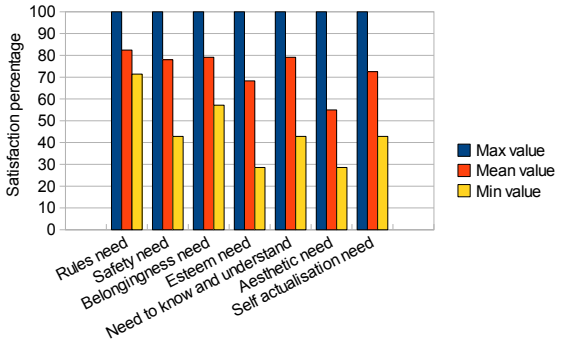

Figure 6: *Students satisfaction for the hierarchy of the players' needs with min and max value.*

The lower satisfaction concerns the aesthetic need with a greater proportion of satisfaction between 28.57% and 55% than between 55% and 100%. The low-end vectorial graphics of *Kernel Panic* seem disturbing for our students. Nevertheless, this simplified RTS allows a quick learning curve which takes part in the positive satisfaction of lower levels. Therefore we still think that *Kernel Panic* is not a bad choice for supporting Prog&Play.

We notice that the second level (Safety need) has been resolved, in a large part, by teachers. In the filmed session example (see table 1), the teacher spent twenty two minutes and thirteen seconds explaining how to use the game. Currently Prog&Play is a serious game for programming practice; help information about programming, learning contents and game usability are not included into the game and must be added by teachers according to their teaching choices.

3.4. Report and evolutions

As we have defined in subsection 3.1, a design experiment "*is subject to test and revision*". After the analysis of this first experiment (first iteration), we propose a revision of our evaluation in order to prepare for future experiments.

Our questionnaire on motivation, designed for students' selection, has been reviewed to be integrated into the evaluation. This integration enables to evaluate influence of Prog&Play on students' motivation. We center this questionnaire on Viau [Via97] works. Indeed, as we have detailed before, we want to use a serious game to motivate students. However, what is motivation and how can we evaluate its evolution? For Viau, the performance to achieve an activity is the observable result of learning and is a consequence of

motivation (i.e. performance is an indicator of motivation). Acting on motivation is a means to indirectly allow the students' performance to evolve. In Viau's model, determinants and indicators characterize the motivation. Determinants of motivation concern: the student's perception of the value of an activity; the perception of his/her ability to achieve it; and the perception of the controllability of its unfolding and of its consequences. Indicators of motivation are characterized by: the choice to start an activity; the determination and the cognitive commitment to achieve it; and the student's performance. Operating on one of these motivational components can affect all the others and indirectly improve students' performances. In order to evaluate the evolution of motivation during the experiments, we built two questionnaires. The first one is handed out to students at the beginning of the first lab session and the second one is handed out to students at the end of the last lab session. Using these, we will be able to answer the following questions: Is the students' motivation to persevere with computer science modified through the use of our serious game? Which feature of our serious game is the most important for motivation?

We think that Prog&Play could be interesting not only for programming practice but also for recruiting students in computer science. Therefore we plan to analyse students' orientation choice after the use of Prog&Play.

Studying the usability of the serious game only for students is not sufficient. We also need to check its usability for teachers (Did they succeed in inserting the game in their lessons?) and for system engineers (Did they manage to install the game into their environment?). For future experiments, we plan to integrate this into our evaluation.

With these revisions, we designed a new temporal schedule of evaluation (see figure 7).

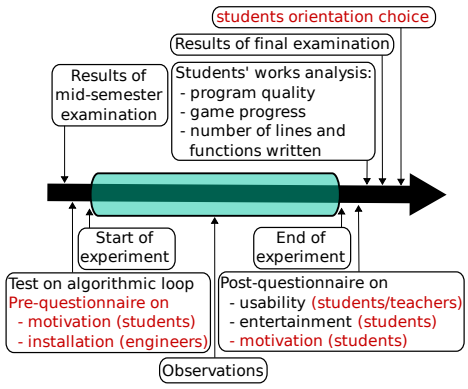

Figure 7: *Temporal schedule of evaluation for future experiments.*

4. Conclusion

This paper describes Prog&Play, a serious game to encourage students to persevere in computer science. We present an evaluation and we discuss it in the context of a first experiment. We are aware of the small amount of data of this experiment and we are careful from hasty conclusions. Currently, we are not in position to validate or to invalidate interest of Prog&Play to learn programming. We wait for new experiments to increase our sample groups in order to carry out quantitative analyses and robust statistics. Nevertheless, first results on entertainment and usability of the serious game are encouraging and incite ourselves to persevere in this way.

A possible evolution of the serious game is to combine it with a block-based graphical language like StarLogo. It will be interesting to study the impact of these two technologies on students' grades. We made Prog&Play flexible towards programming languages; we plan to do the same with video games. Indeed, with the quick evolution of video game standards, its integration into new RTSes is essential to continue interesting 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.php`. These pages are currently available in French only but will soon be translated into English. If you are interested in our serious game, do not hesitate to get in touch with us!

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 10 December 2009. 2

[Ban97] BANDURA A.: *Self-efficacy: The exercise of control*. New York: Worth Publishers, 1997. 4

[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. 1, 4

[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. 3

[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 10 December 2009. 2

[Dar] DARFUR IS DYING: http://www.darfurisdying.com/. accessed 10 December 2009. 2

[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

[Gin04] GINAT D.: On novice loop boundaries and range conceptions. *Computer Science Education 14*, 3 (2004), 165–181. 5

[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. 6

[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. 1, 4

[Gun] GUN-TACTYX: http://apocalyx.sourceforge.net/guntactyx/. accessed 10 December 2009. 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. 6

[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). 1

[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 10 December 2009. 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. 1

[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, 4

[Mar] MARVIN'S ARENA: http://www.marvinsarena.com/. accessed 10 December 2009. 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. 1

[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. 3, 5

[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. 4

[PS96] PINTRICH P. R., SCHUNK D. H.: *Motivation in Education: theory, research and applications*. Englewood Cliffs : Prentice Hall, 1996. 4

[Roba] ROBOCODE: http://robocode.sourceforge.net/. accessed 10 December 2009. 2

[Robb] ROBOT BATTLE: http://www.robotbattle.com/. accessed 10 December 2009. 2

[SC05] SMITH L., CORDOVA J.: Weighted primary trait analysis for computer program evaluation. *J. Comput. Small Coll. 20*, 6 (2005), 14–19. 4, 5

[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. 4, 6

[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. 1

[Tac] TACTICAL LANGUAGE AND CULTURE: http://www.tacticallanguage.com/. accessed 10 December 2009. 2

[The] THE NUMBERS: http://www.the-numbers.com/market/2008.php. accessed 10 December 2009. 1

[Via97] VIAU R.: *La motivation en contexte scolaire*. Bruxelles : De Boeck, 1997. (in French). 4, 6

[Zyd05] ZYDA M.: From visual simulation to virtual reality to games. *IEEE Computer 38*, 9 (2005), 25–32. 2

Appendix A: Exercice example

We present in figure 8 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 exercice.

```
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(ME, 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 8: *A solution written in* C