LEARNING OF THE OBJECT ORIENTED PARADIGM THROUGH INTERACTIVE VIDEO-GAMES DEVELOPMENT

D. Rodríguez-Losada, M. Hernando, P. San Segundo, R. González, C. Platero, L. Dávila, P.L. Castedo, S. López
Universidad Politecnica de Madrid - GSITAE
Madrid, Spain
diego.rlosada@upm.es

Abstract

The Object Orientation Paradigm (OOP) is more than Object Oriented languages. Learning the syntax of a language as C++ or Java is a relatively easy task compared with the understanding of the principles of OO Modeling and Design (OOD), which require a high ability of abstract reasoning. Moreover, it is not enough to teach the artifacts of Computer Aided Software Engineering (CASE) as the Unified Modeling Language (UML) if those principles are not properly understood. We wanted to engage the students in a motivating framework, so both the principles of OOD are properly acquired and put in practice with CASE and programming tools.

We found the ideal framework in the development of videogames. Most students are familiar with (even attracted by) them. The existence of powerful yet simple 3D graphical libraries as OpenGL and the portable Window Manager (GLUT), makes very easy the development of quite attractive 3D applications, so the real effort can be done in the OOP. There are two types of games: those with physical simulations (first person shooters, car race, arcade games) and those with logical or reasoning (chess, 4 on a line, etc). Though the second type is interesting, physical systems can be more easily understood and modeled under OOP, because the software and design objects have an equivalent counterpart in the real world. Furthermore physical systems have the added benefit of involving kinematics, dynamics, math, and other disciplines very interesting for the engineering student.

At Universidad Politecnica de Madrid, we teach Industrial Informatics which covers both the topics of C++ programming and OOP, starting from a basic knowledge of C language. The subject is taught with 3 hours of C++ theory, 2 hours of OOP theory and 2 hours of practice per week, along a semester, plus a compulsory assignment to be accomplished by the students.

We decided to propose the development of a video-game to be carried out along the semester, by groups consisting of two students. Taking into account the required effort, we also decide to restructure the subject around this idea. The practical lessons have been completely modified, leading the student from one practical lesson to the next one in an incremental development of a typical arcade videogame: Pang, in which a player shoots to balls bouncing around the screen. Our focus is the OOP: the application is iteratively built following OOP principles: object creation and destruction, object interaction, relations between classes, GRASP and GOF software patterns, etc. CASE tools and UML are constantly used in this process, in order to support the ideas that motivate our final implementation. The theory and tutorial hours have been also affected: The tutorial time has highly increased, basically overwhelming our capacity. This fact also contributes to the actual understanding of the subject. The theory lessons find the support on the video-games development and practical lessons, making easier for the students to understand the abstract ideas explained in class.

We are convinced about the effectiveness of our approach. The motivation of the students has become clearly visible, both in the tutorial lessons and in the high quality of the developed games. The students' grades have clearly improved, not only because of the contribution of the good grades of the game assignment, but also because the understanding of the concepts and the increase of hours dedicated to practical developments.

Keywords

Video-games, Object Oriented Technologies, Motivation, Software Engineering.

1. INTRODUCTION

The subject “Industrial Informatics” is part of the current degree programme of Industrial Engineering with specialization in industrial electronics, automation and informatics of Escuela Universitaria de Ingeniería Técnica Industrial (E.U.I.T.I.) of Universidad Politécnica de Madrid (U.P.M). This subject is taught during the fourth semester (second year, second semester) of the degree by the Departamento de Electronica, Automatica e Informatica Industrial (ELAI) [1], to which the authors belong. It is a compulsory subject to be followed by all students of this programme, approximately 110 students each course.

Undergraduate students of the above programme earns 10.5 credits with this subject, that are distributed in 7.5 credits of theoretical contents (without computer) corresponding to 5 hours per week and 3 credits to practical lessons, equivalent to 2 hours per week (with computer, one per student). The students are evaluated during these practical lessons, and at the end of the semester, they have to pass a written theoretical-practical exam. They also have to fulfil a practical assignment with a relatively high necessary effort that is also taken into account in the final grade of the subject.

The “Industrial Informatics” subject follows the course “Foundations of Informatics” (second semester, first year) which is basically a brief introduction to computer technology and an extended introduction to procedural programming in the C language. Although the latter subject is not a prerequisite, it is strongly encouraged as a basis for the former one. The subject of Industrial Informatics covers the paradigm of Object Orientation (OO), from Object Oriented Analysis and Design and C++ programming to Software Engineering Models as the Unified Process as a framework and CASE tools as UML. All these contents are basic in the actual landscape of Software Engineering, not only in the industrial environment but in management, commerce and many other applications as well.

The area of Informatics of the programme is completed with a third year optional subject (incorrectly named “Software Engineering”) that covers the topics of Operating Systems and concurrent and distributed programming.

Although it is mainly focused in the Informatics area, it must be said that the contents of this subject are closely interconnected with many other subjects of the programme, as Process Control, Electronics, Robotics and Computer Vision, and the overall formation of the students as future engineers should be increased after passing it, with newer abilities of modelling, design, abstract thinking, problem solving, etc. We strongly believe that a good grasp of this subject is extremely important for our graduates’ profiles, which motivated the work presented in this paper.

Section 2 presents some related work and our main goal: how to motivate the students to get sufficiently involved in this subject, so the understanding of the (sometimes difficult) concepts of the OO is improved. Our approach was to propose the development of interactive video-games, obviously with the help of a 3D graphical library as OpenGL, as the compulsory assignment to be fulfilled during the semester. Section 3 describes how we re-structured many aspects of the teaching methodology, specially the practical lessons, to provide the necessary support towards this goal. Some of the results are presented in section 4, and our conclusions and future work is summarized in section 5.

2. MOTIVATION

Computers are a very useful tool for learning, and there exist many interactive systems that help the students to learn many different subjects: History, Geography, Mathematics, Physics, etc. All these systems are characterized by a high interactivity, usually with a carefully designed graphic interface, and an attractive appearance. In fact, interactivity, visual design and appearance are key components of modern technology, from the World Wide Web to mobile phones or even household appliances. People, especially young people like our teenager students, are very used to them.

With all this impressive technology around, engaging a student to develop a typical “console” application could be sometimes possible, for example if an interesting algorithmical challenge is proposed, but can be even frustrating in many other cases. Thus, we had in mind that the proposed assignment should have at least some kind of interactive graphic display.

Consequently we decided that developing an interactive video-game could be very interesting. Most of the students are very used to them; in fact many of them are frequent players of computer games in machines, consoles, mobile phones and PCs. Our first concern was the possible difficulties of using a graphics library, but the existence of a 3D renderer graphics library as OpenGL which is very easy to use (at least for drawing simple 3D scenes, without advances effects as lighting) was finally convincing. Moreover, the GLUT window manager is also quite simple to use, so we adopted it to abstract the students from dealing with the system API. An added benefit of the GLUT window manager is that it is portable, so we could easily change the development environment and operating system if desired without requiring any modification to the code. Furthermore, all the GLUT functionality and many OpenGL functions can be easily encapsulated in a class provided to the students, so they only need to know a few drawing primitives that can be learnt in a couple of hours.

The development of video-games quickly turned out to have many appealing characteristics. Many computer games involve many physics simulations: kinematics, dynamics, collisions, etc. (consider sports, arcade, first person shooters, and so on) that the developer of the game must know, decide how to use them, design the solution and implement it. Many operations involved are matrix and vector operations that have to be implemented (apart of the graphics library, no other one is allowed), thus requiring some algebra knowledge. It is also true that there are many computer games not involving such disciplines. Among them we could cite the logical games as chess, 3 on a line, tetris, etc. Although developing this kind of game is quite interesting, we certainly prefer the physical ones, because the most important benefit of the physical systems is that they are easy to understand and model, and the OO design of the solution is quite intuitive. From our point of view this is the main benefit of our approach: it greatly increases the understanding of the OO paradigm, as the student is able to establish clear connections between the system being modeled, the OO design, the C++ classes and the final output on the screen.

There exist some other experiences related to our approach, where the students are given a similar assignment as in Universidad Complutense [2] or in Universidad de Alcalá de Henares [3], both in Madrid, Spain. Nevertheless, other existing experiences are typically focused on algorithms, or even in the study of the graphics library itself, while we have little interest in it, just as a vehicle for teaching Object Orientation Programming. It is interesting to cite also the recent release of Dark Game Development Kit (GDK) together with the free Visual C++ 2008 Express Edition [4]

3. TEACHING METHODOLOGY

As explained above, the motivating framework is based on the development of a graphical computer game, to be carried out by the students along the semester. While it is possible to develop such game during the practical lessons that would mean that all students should develop the same one. We think it is a better approach that each student can develop his/her favourite game, which they are free to choose. Thus, such development is made an assignment they have to accomplish as homework. Obviously such task cannot be carried out without a large practical and theoretical support, and a knowledge they have to acquire during the lessons. Large parts of the subject have been redesigned to account for such a goal without changing their pedagogical contents. This section revises how the proposed approach has influenced the subject, how the practical lessons have been modified, the consequences on the tutorials and how the theoretical lessons are linked with their practical developments.

3.1 Assignment: design and develop a video-game

Although individual assignments are occasionally allowed, the assignments has to be carried out by pairs of students, in order to promote collaborative working but also to take advantage of the advanced students that can share their knowledge with their classmates. Students with a lower programming profile can easily find help in their pairs to learn the subject without requiring the professor tutorials, thus avoiding a total overflow of the tutorials capacity.

As described above, computer games with physical objects, interactions, and animated displays are preferred. Nevertheless other kinds of games are also allowed, and initiative and new ideas are always encouraged.

The development of such software is not an easy task. The students find the primary support and developing guidelines in the practical lessons, that have been redesigned to teach them the necessary abilities. As developing those kinds of games is not an easy task, the student constantly needs to revise the theoretical lessons, the C++ syntax and the OO concepts. Basically, it is avoided the development of a game by many repetitive tasks, but instead putting in practice many different aspects of OO and C++ programming is required: objects creation-destruction, code reuse, code encapsulation, interaction between software objects, inheritance and polymorphism, etc.

Many things are taken into consideration to evaluate the developed software. A good OO design is fundamental, but also good C++ practices are important as code style, adequate comments or folder structure. Adding visual effects as textures or sounds and music usually have little interest from the subject pedagogical goals, but are interesting from the motivational point of view, so they are also taken into account. More interesting is the development of physical simulations, behavioural or game-logic development, improved user interfaces and other things that can be easily integrated in the project software architecture.

At the end of the semester, the students have to deliver the source code, to show the game functioning, but also all the documentation: use cases, analysis and design UML diagrams, etc.

3.2 Used tools

Many software tools are used in this subject. The chosen Operating System is Microsoft Windows (mainly 2000 and XP) for practical reasons: that is the system that all students have already installed in their homes. It is true that we could impose (and we are sometimes tempted to) free software (as Linux and gcc) as development environment, but this would produce a huge amount of work that is not the primary objective of the subject. Such approach (using Linux + GNU) is done in “Software Engineering” subject of 6th semester.

We use Microsoft Visual Studio (C++) 6.0. Although it is almost deprecated in the Information Technologies and Informatics area, it is also true that it is still largely used in industrial environments. OpenGL and GLU libraries come along with the Visual Studio installation, and only the GLUT library is required. This library is quite small, only consisting of a single header file, a static library (.lib) and a dynamic one (.dll) files that can be easily copied to the development folder and managed as part of the project without requiring any modifications on the computers.

We could have used other programming language as Java, that is even more adequate for OO, and much simpler in many aspects than C++, so we could focus on the OO methodologies. Nevertheless, C++ is more common in industry as Java requirement of a Virtual Machine to execute the bytecode has traditionally implied less efficiency. Also it is important to remember than other subjects of the programme use the C language.

We have traditionally used Rational Rose as the main CASE and diagrammatic tool for UML modelling. Nevertheless, this tool has a few disadvantages for the students, it is not free at is quite heavy, both in size, required resources and time to install. Since the last course, we are beginning to use and recommend BoUML [5], which is a lightweight application, totally free, easy to install and use. The documentation can be written in any word processor.

Several resources are available on the subject web page [1], from document templates for the software artifacts to code examples, tutorials of OpenGL and classes material.

3.3 Practical lessons

The practical lessons have been fully redesigned with a simple goal: at the end of the semester the students will conclude with a complete computer game developed at class. A typical game has been chosen because of its popularity, but also because of its many interesting features: the Pang. In the Pang game, the user move in a window shooting to several balls that are bouncing around the screen.

When a bullet hits a ball, the ball is split into two smaller ones. If a ball hits the player, the player loses points until it dies. Figure 1 shows a snapshot of the proposed application.

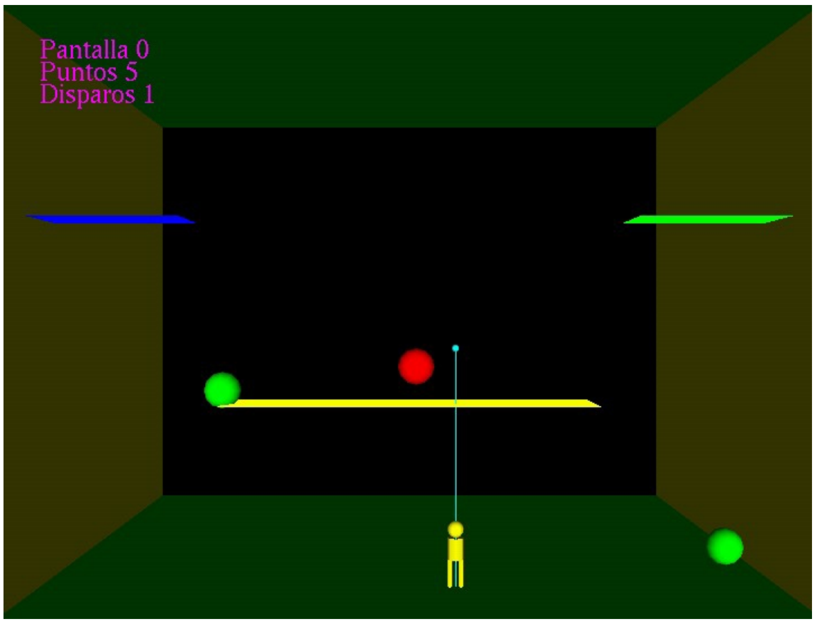

Figure 1. Pang Computer Game

As it has been previously exposed, the subject has been designed so that the acquired theoretical knowledge could be taken to the practice progressively. With this purpose ten practical lessons have been designed. The theoretical concepts are exposed and illustrated incrementally while a set of tools for the development of the game are provided. The practical lessons, as well as a brief description of its educational aims, are described next:

1. C programming: Using the Debugger.

   Introduction to MSVS6.0 and the debugging tools.
2. Introduction to OpenGL and GLUT window manager

   Introduction to OpenGL. Use of the Glu and Glut libraries. Structure of the application. Handling system events with callbacks.
3. C++ classes

   Introduction to the power of OOP, classes, objects, encapsulation. The first class: Sphere.
4. Object Oriented Analysis and Design with UML.

   Relation between design and implementation. Classes and Sequence diagrams. Several new objects are included into the application: the shots, the player, the bonus, etc.
5. Interactions

   Use of Indirection GRASP pattern to implement physical interactions between objects. Development of class Vector2D, operator overloading.
6. Creating and destroying objects.

   The importance of object creation, destruction. New, delete. Development of a dynamic containment class that can handle a variable size vector of Spheres.
7. Discovering inheritance by specialization and generalization

   Reutilization of code. Base classes for graphical objects. New child classes.
8. Polymorphism and inheritance.

   Specialized Classes with common interfaces. Polymorphic copy. Implementation of a state machine. End of the game.
9. Design with Interfaces.

   Learning to design polymorphic solutions. Abstract Base Classes, virtual pure methods.
10. Final practical examination

The last practical lesson is a final examination, and entails carrying out a simple object oriented program. The final qualification of the laboratory is obtained taking into account the notes of the different practical lessons and especially the final examination. This value constitutes the 25 % of the final mark of the subject. To pass the practical part of the subject it is compulsory to pass the practical examination independently of the qualifications obtained during the course.

The practical lessons consist of 2 hours sessions in a classroom of 22 PCs. The computer used by the professor is connected to a video projector, and therefore the students can easily follow the explanations. During the last two years, an average of 84 students took the course, distributed for the practical lessons in 7 groups of 12 students each.

At the beginning and during the first 15 minutes, the professor corrects the exercises proposed in the previous session. These exercises consist on small variations that the students have to include in the version of the program at the end of the last practical lesson. Original contributions (they are given freedom) and solutions are positively evaluated, and in any case, a fast inspection of the code is made to verify that the POO philosophy has been followed.

After the correction, the teacher explains in a few minutes the main concepts used along the lesson. After that, step by step, the points of the practical lesson script are followed, with solutions that must be programmed by the students applying what it has been explained.

Every practical lesson ends with a more evolved version of the program that must be used as base for the accomplishment of the exercises proposed for the next practical lesson. That final version is the program where the following practice starts.

3.4 Tutorials

In the tutorial action, the majority of the time is dedicated to outline the final work of the course. Following the Unified Process and considering the proposed program of each group, the first three weeks are used to formalize it writing the documents of Vision and Scope, Cases of Use, Glossary and Complementary Specifications. For that purpose a few templates are given to them to make the carrying out of the mentioned reports easier.

In the following month the analysis of the program is being carried out using the artifact of the Domain Model. Also the students are asked for the knowledge of the graphical libraries used during the practical lessons.

Since the second month, the design artifacts, the class diagrams and the sequence diagrams, are requested. At this point the tutorial action makes synergy with the software patterns explained both in the classes of theory as in the practical lessons.

At the beginning it is only requested to them the use of the patterns of Low Coupling, Information Expert and High Cohesion with the intention of defining the software packages and making easier the organization of the production of software. More ahead, the students are motivated to develop a single factory of objects and to design using Protected Variations.

3.5 Theory

The theoretical lessons are divided into two main areas: C++ [6] and OO [7]. While it is clear how the methodology fits the practical lessons and the course assignment, the remaining question was whether the theoretical lessons could benefit from the approach. Teaching C++ syntax and programming details can be more easily done with code fragments and examples, but as described above, teaching the OO paradigm is much more difficult.

The OO programme covers many aspects of software engineering, as Use Cases, the Unified Process, and explains many different artifacts, generally using UML as diagrammatic tool. But the main effort has to be done on the philosophical aspects of the OO, so the students can get a good grasp on the concepts, in order to create good software OO designs and software engineered solutions. Large parts of the programme is thus dedicated to perform OO analysis to generate Domain Models, but mainly in OO design through the use of software design patterns as GRASP and GoF [8].

To explain all these things, we found it very convenient to establish connections with the practical lessons, and computer games in general. Very clear and illustrative examples can be done following this approach. Moreover, the student can hardly assimilate and understand all the theoretical concepts involved in the practical lessons: they are usually too busy coding and debugging, and rely on the guidance of the teacher for architectural issues. The theoretical lessons can clarify, justify and explain the architecture, the used design patterns and solutions, and propose new examples with different games, apart of the Pang developed in the practical lessons.

As an example, this exercise was proposed as a part of the theoretical exam of September 2007:
A game has to be developed in which men fight against dragons. Men use (throw) knifes and dragons use (spit) fire balls. There are mobile walls in the middle of the screen, with the men located on one side (right) and the dragons on the other one. The number of players is variable.

The following list of two levels is then defined:
1. Game of Men vs. Dragons

    1.1. Men move on a restricted area at the right of the screen

    1.2. Dragons move on a restricted area at the left of the screen

    1.3. Men throw knifes that stick to walls or kill a dragon

    1.4. Dragons spit balls of fire that kill men, but cannot pass walls.

    1.5. Both dragons and men disappear when they die.

Then, the Domain Model has to be done. The students have to apply the required techniques (like linguistic tests and lists of conceptual categories) to identify concepts, and their associations, and then translating all of them to a UML diagram. A reference solution (not necessarily the only correct one) is shown in next figure:

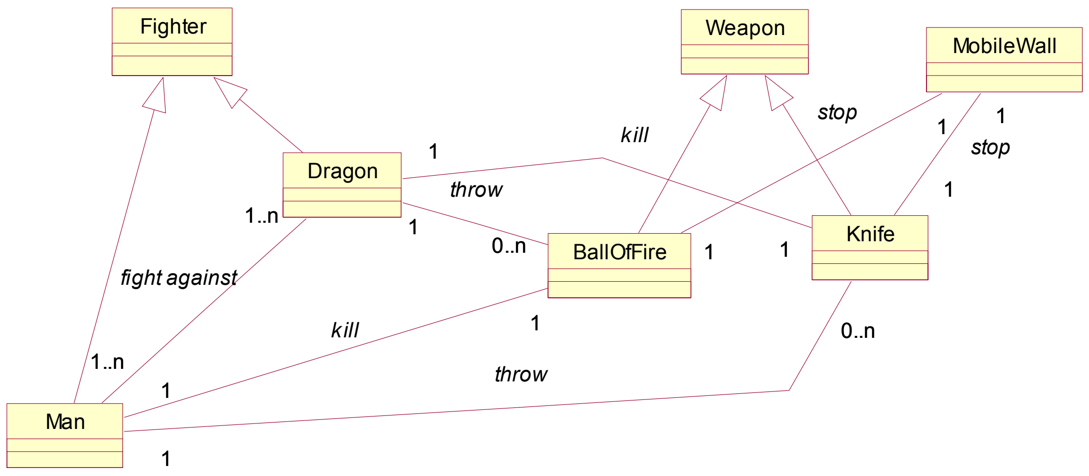

Figure 2. Domain Model in UML. Theoretical exam, September 2007.

After that, the Design Classes Diagram is to be carried out in UML. Large parts of the theoretical lessons of the OOA/D are focused on GRASP and GoF design patterns. This exercise is a good example where many patterns can be found. A Factory can be used to create objects, and the factory can be instantiated as a Singleton. The Pure Fabrication and Polymorphism appear in a natural way, while their uses clearly provide Protected Variations. The reference solution is shown in next figure. Although other possible solutions are possible, the problem statement should produce in practice a solution close to the given one.

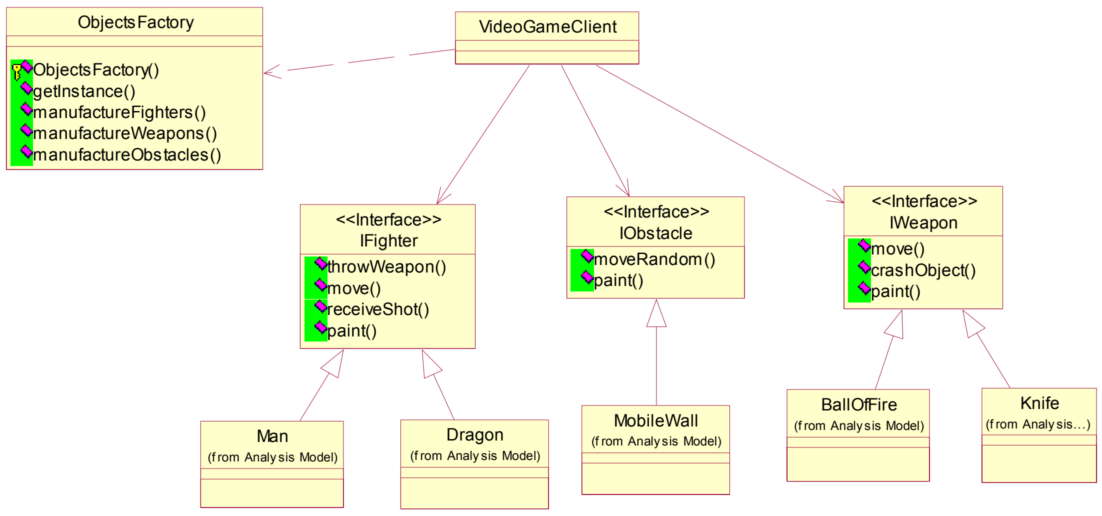

Figure 3. Design Classes Diagram in UML. Theoretical exam, September 2007.

Finally, the source code (only headers files with definitions) is required, to show how the design is transformed into the architectural skeleton of the application.

4. RESULTS

There are tasks for students which are fully dictated by the teacher and other tasks which the pupils can propose. Amongst other topics videogames stand out (see figure below) such as: typical platform games where the player jumps over platforms killing the enemies and picking up objects on the way; sports games such as golf, car or motor racing; more recreational ones such as pinball or even the so called 'first person shooter' action games.

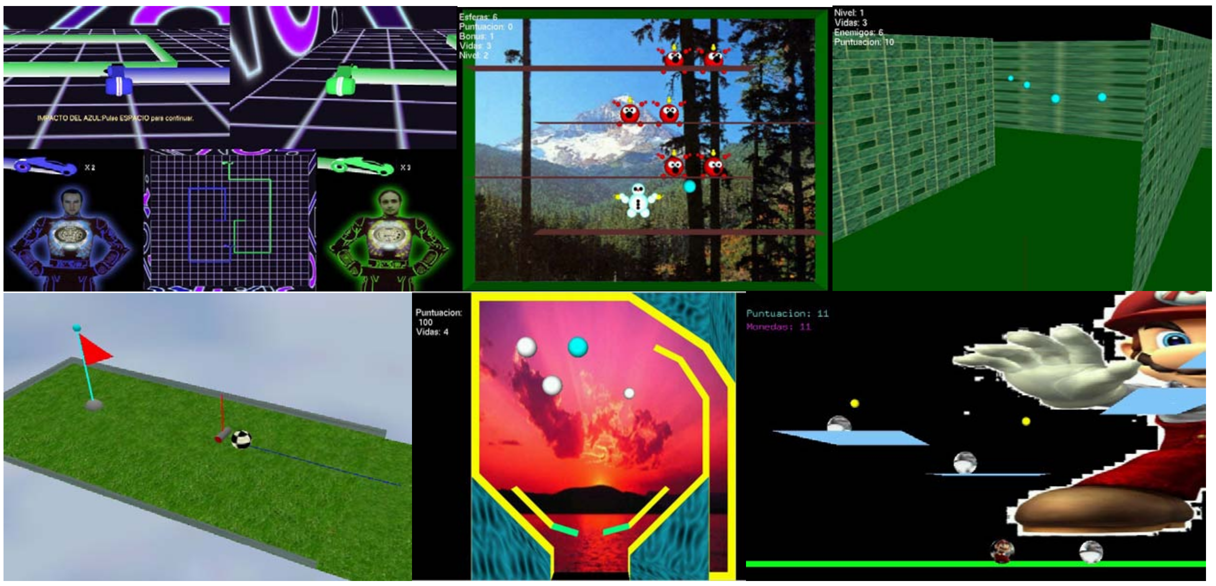

Figure 4. Some games developed by students of course 06/07

The proposed methodology has been very successful amongst our students, and actually very motivating in many cases. We note that many of them continue to develop their videogame after having passed the subject. In some cases their own families get involved and act as "testers" of their software products.

We would also like to point out that the actual time our students dedicate to developing their software increases notably due to their motivation. The teachers of the subject estimate that this increase is from 25% up to a 100% in some cases, compared to the same pupil working in a different project. In the long run this translates into an increase in knowledge and a better assimilation of the subject, which clearly benefits the pupil and gives rise to overall higher marks amongst the students.

5. CONCLUSIONS AND FUTURE WORK

We conclude from our experience that the development of graphical computer games is definitely a good educational approach, and specially suited to address the difficulties inherent to the Object Orientation Paradigm. The development of such programs is completely affordable in terms of time, and the motivation plays a crucial role in learning both the C++ syntax and the use of OO to engineer software solutions. In fact, the final results, as shown in the previous section, usually surpass our expectations. There is absolute unanimity in the success of this approach, which motivates us to keep working in this or similar directions. Some of our new ideas are mentioned here.

A web page will be used to host common resources, tutorials, software, as those related to camera movement, lights and textures in OpenGL, physic simulations of explosions, collisions, or the use of music and sounds. Furthermore, we pretend to set up the web site so the students can upload their games, and make them publicly available, including a “top ten” ranking based on the number of downloads and public surveys.

The same methodology has been recently adopted in the “Software Engineering” subject. This 07/08 course has been the first one in which the students had to develop a distributed multiplayer computer game. Although the development environment was Linux, their projects could be largely reused as a based for the new assignments, thanks to the GLUT portability, so the students could focus on developing protocols and communicating by sockets, and dealing with concurrent solutions. Although we considered the use of the GDK distributed with Visual 2008 Express, we are still reluctant to use non portable solutions.

References

[1] Department of Electronics, Automation and Industrial Informatics (ELAI) of Universidad Politecnica Madrid (UPM), Web page: www.elai.upm.es

[2] Universidad Complutense Madrid. Game development at class. http://www.fdi.ucm.es/Guia_Docente/ver_prog_asig.asp?Titulo=450&Asignatura=525&fdicurso=2007-2008

[3] Universidad Alcala de Henares. Game development at class. http://www.etsii.uah.es/Estudios/pdf/guia-docenteII06-07.pdf

[4] Microsoft. http://www.microsoft.com/

[5] Bruno Pages. BoUML. http://bouml.free.fr/

[6] Hernando, M. Programación C++. Servicio de Publicaciones EUITI-UPM, 2005.

[7] Platero C., Apuntes de Informática Industrial (Análisis y Diseño Orientado a Objetos), Servicio de Publicaciones EUITI-UPM, 2006.

[8] Larman, C., UML y Patrones. Una introducción al análisis y diseño orientado a objetos y al proceso unificado, Segunda edición, Pearson Educación 2002.