Titor's Equilibrium Project
Final Report

Marino Alge
Gioacchino Noris
Alessandro Rigazzi

June 29, 2007

Contents

1 Introduction **2**

2 Design **4**
2.1 The initial design outlines 4
2.2 The final design 5
2.2.1 Ghosts 5
2.2.2 Hosts 6
2.2.3 Powers 7
2.2.4 Damage system 8
2.2.5 The Arenas 9

3 Development **10**
3.1 Team Description 10
3.2 Scheduling and Milestones 10
3.2.1 Getting started 10
3.2.2 Performance Nightmare 11
3.2.3 The rush to the alpha 11
3.2.4 The last phase 12
3.2.5 Summary 12

4 Playtesting **13**
4.1 Setup 13
4.2 Results 13

5 Conclusion **15**

1 Introduction

Titor's equilibrium is a death match video-game based on physical interaction of rigid bodies developed in the context of "Game Programming Lab" in the summer semester 2007 at ETHZ.

The game allows the players to fight against each other in a last man standing fight. The match entails some ghosts put into an hostile environment that forces them to find and infest some host objects, in which they are safe. Once inside a host, Jedi-like powers and time manipulation become the weapons which will be used to kick the other ghosts out of their hosts, and make them starve in the void.

The atmosphere of the game is a virtual reality, possibly inspired to Tron and Rez. Everything, form the arena environment to the hosts, is composed by simple and basic geometric shapes. The color inside the game have particular meaning.

The hosts, and their associated powers, have different color, in a pool of three (red, green, blue). Like the Rock-Paper-Scissor mechanism, red beats blue, which beats green, which beats red. This component forces the players to develop tactics to be sure that they can switch to the right host.

The game offers unique physics and time related abilities. Hosts and other junk objects will be subject to gravity fields, with charging powers that allow the player to attract or repulse everything. The time can be lowered or even reversed, both globally and locally, allowing the players to protect themselves, or to turn attacks against their dealer.

The game is called "Titor's Equilibrium". The first word refers to John Titor, a person(s?) that in 2000/2001 was quite active in the internet time-related communities, claiming to be a time traveller from the year 2036. He was supposedly sent back to 1975 to retrieve an IBM 5100 computer which he claimed was needed to "debug" various legacy computer programs in 2036. Besides the fun of the thing, we got inspired to the concept of time handling and we planned to put it into the game (slow motions and, if possible, time reversing). The second word refers to the physical concept of equilibrium, which we, as Titor, are going to break :)

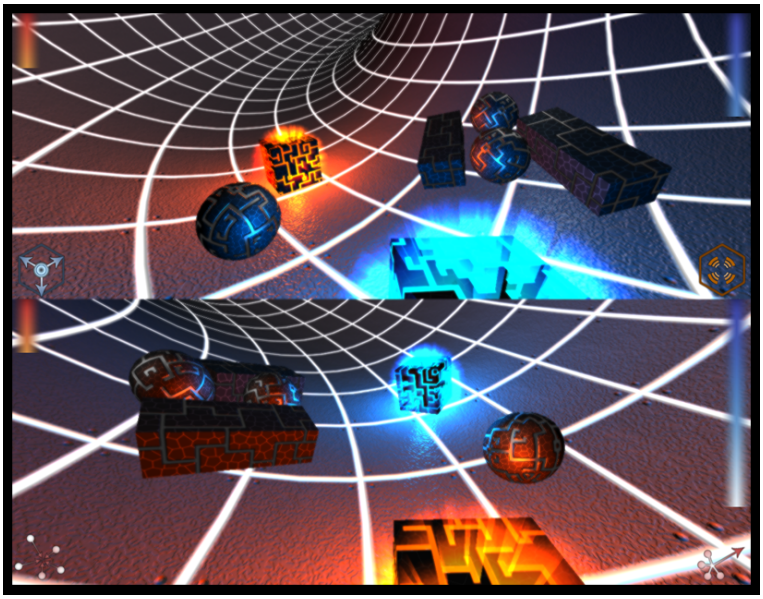

Figure 1: A screenshot of the final version of Titor's equilibrium, representing two infested hosts, a red and a blue one, plus some arena junks

2 Design

2.1 The initial design outlines

We wanted a game which could be feasible by a small team in a small period of time. We wanted a simple concept, in terms of gameplay and graphics. We decided at the very beginning to go for a virtual reality atmosphere, with simple elements and interactions. Since we had already worked as team to a physical simulation of rigid bodies, we decided to port it to a game.

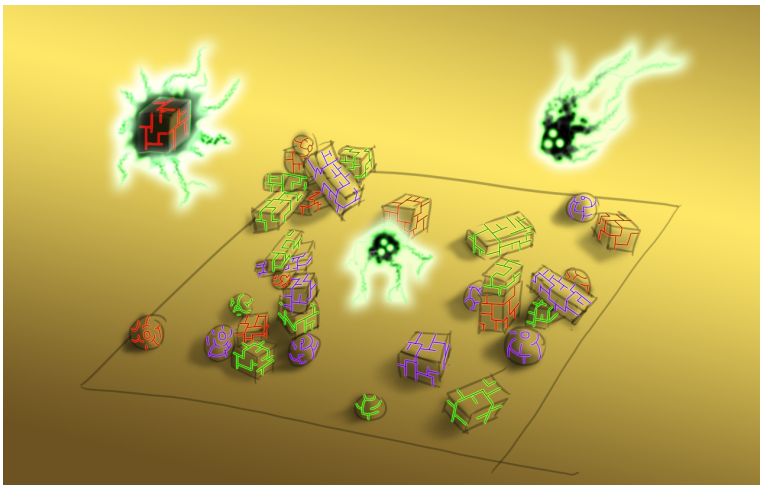

Figure 2: Concept art

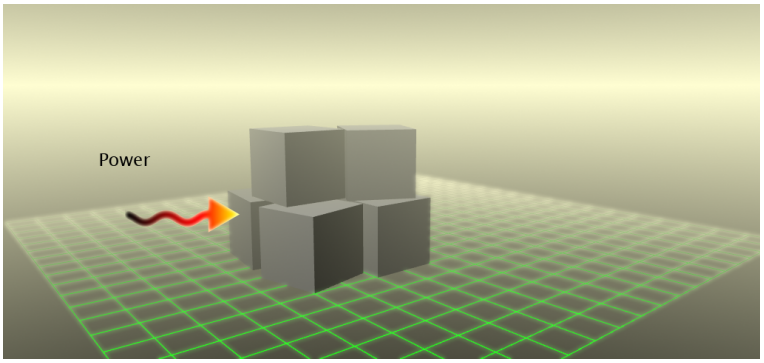

Figure 3: Power Sketch

The process has been a loop with several refinements, for both gameplay and graphics. The initial design entailed multiple goals and kind of powers, as well as visual effect. Some of them were dropped because unrealistic or unfeasible, other

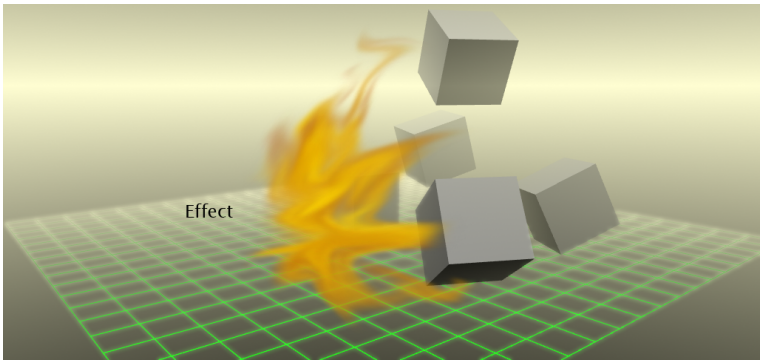

Figure 4: Effect Sketch

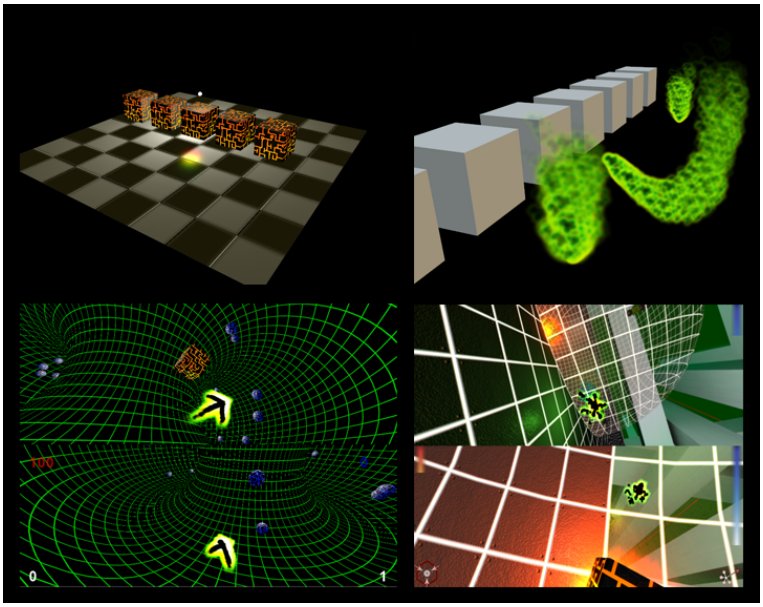

Figure 5: Graphic results at different development stages

because they turned out less brilliant than we guessed. New ideas came during the development. The next section will guide through a complete explanation of the game in its final state.

2.2 The final design

2.2.1 Ghosts

The players will play in the game as ghosts. Ghosts are globe of energy, free to fly around the arena. They have no physical properties; they can fly through objects

and are not affected by gravity, nor by attacks of other players. The only thing a player, as ghost, has to take care of is the amount of energy (shown by the blue bar on the upper left of the screen, as well as the green glow and light emitted by the ghost). This energy will slowly decay in time, and when it reaches zero, the ghost is dead, and the player controlling it looses the match.

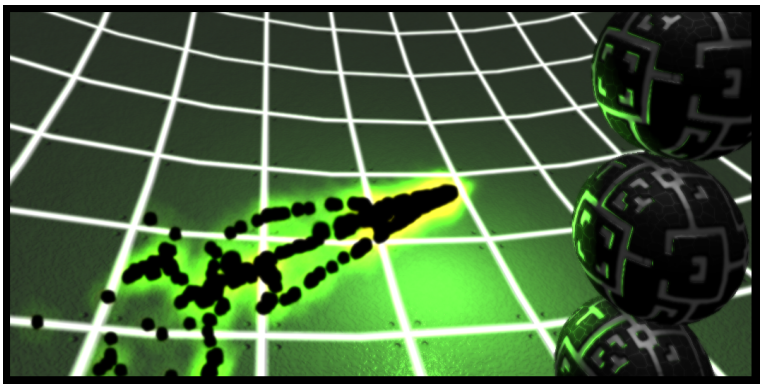

Figure 6: A ghost flying inside the sphere arena

2.2.2 Hosts

The only way to survive, for a ghost is to find a suitable host. Host are recognizable because of their shape - they are cubes - and the appearance - they have colored veins that pulse. A suitable host is one that is not already captured by an other ghost, and has enough health. An already infested host can be recognized by the glowing aura and the emitted light, which represent the fact that there is a ghost inside it. The host health can be spotted by the shining of its veins. Full colored veins mean a healthy host, black veins mean a dead host. Hosts slowly recover their health during time.

A ghost can enter a healthy and free host moving in it. At this point several things happens. The ghost disappears, as now it is part of the host. The player is then in control of the host, and must be aware of two characteristics:

- The first one is - again - the **energy**, though now it has a different behavior and importance. The initial value as host, is equal to the one the ghost had when entered the host. It increase during time until it reaches a maximum amount, which change from color to color (red hosts can have 200 energy points, the blue ones 100, and the green ones only 50). If the ghost had more energy than the maximum amount a host can handle - for instance entering in a green host - the energy decays until it reaches that maximum cap. If the

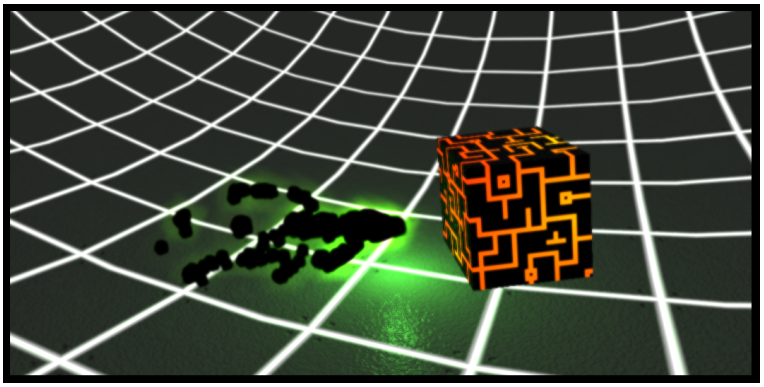

Figure 7: A ghost approaching a Red Host

player leaves the host (intentionally or because of taken damage) the energy is transmitted to the ghost (if it's zero, a small amount is given by default).

- The second characteristic is the **health** of the host. Its initial value is the health of the host when it was empty. Since now the host is occupied, the health does no longer recover, so entering a fully health host is recommended. If the host gets damaged, the health bar decreases, as well as the shining of its veins. If the host gets too much damage and the health reaches zero, the ghost is kicked out, back to its free flying mode. The maximum amount of health is, for all the host 100 health points.

There are other characteristic which will influence the gameplay, but are not shown to the players:

- Hosts carry different masses. Green have 200 mass points, blue 100 and red only 50. See the next section for implications about the mass.
- Host have different acceleration factors. The green ones are the faster, the red ones are in the middle, and the blue ones are the slowest. See the next section for implications about the speed.

2.2.3 Powers

An infested host provides the player with some weapons. The player has the possibility of using two of them at the same time, creating combinations to get in advantage over the adversary. These powers consume energy, and produce various type of effects, such creation of force fields, time manipulation, and variation of the physical properties of the objects. Some powers are instantaneous, and the

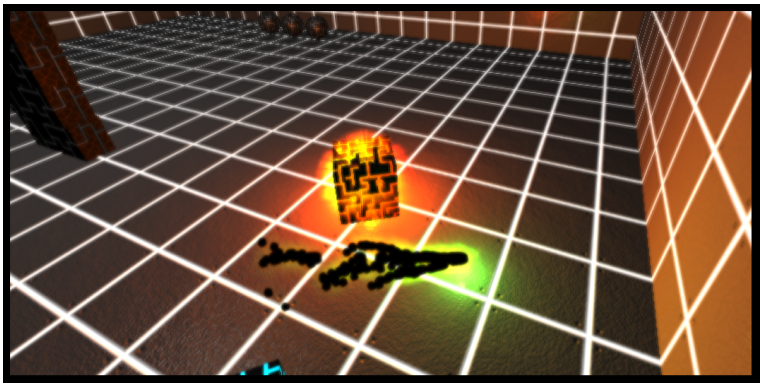

Figure 8: An infested Red Host

analogical pressure tune their intensity. Other require a charge phase, analogical too, and will be effective when the player releases the controller trigger. If the energy reaches zero, all the charge powers are immediately triggered and the instantaneous powers shut down; to cast new powers the player has to wait for the energy to recover.

2.2.4 Damage system

Damage comes from collisions. When a host hits other objects (hosts, or arena junks) a certain amount of health is taken away. There are mainly three factors which influence this amount.

- One is the **mass** of the objects involved in the collision. An object with higher mass will take less damage. The hosts have different mass points: red 0.5, green 1.0, green 2.0. For instance a green host will likely damage a red host without getting much affected on its own. Moreover, there are powers which change the masses of the objects, and will therefore produce a different outcome of the damage dealt by a collision.

- The **relative speed** of the object involved in the collision, in terms of magnitude and direction will affect the damage dealt by the collision. A high speed front crash will deal a lot more damage than the slow friction of two object sliding on each other on the side.

- The last thing is the **absolute speed** of the object. If you are going faster, you take less damage. This encourages the players to build some speed before crashing against the opponents, and prevent unexpected collision with the arena junks to deal too much damage while riding fast.

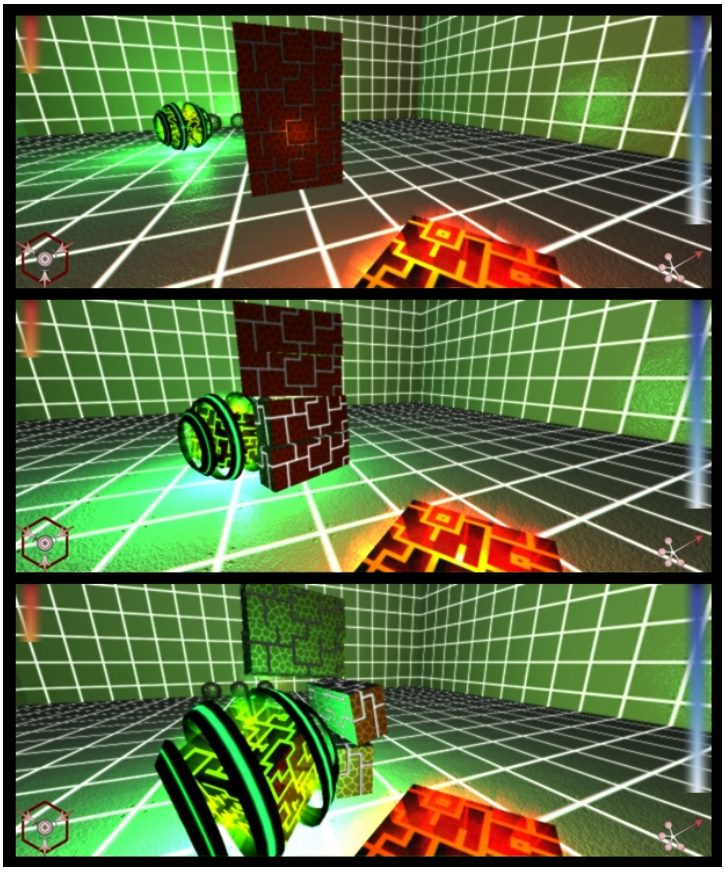

Figure 9: The green hosts can morph into a sphere, gaining various attributes

2.2.5 The Arenas

There are four types of arena in the game: Sphere, Cube, Cylinder and Torus. Each one has a unique gravity field, as well as hosts and junks configuration.

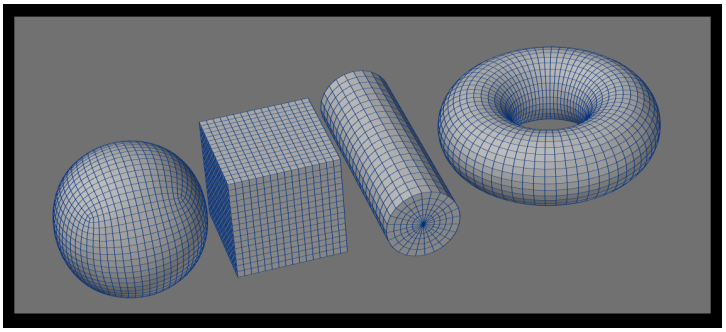

Figure 10: Shapes of the arena of the game

3 Development

3.1 Team Description

As team, we had complementary skills, which allowed us to easily split the work between us, defining the competence regions, and therefore avoiding problems at the decision level. Each of us had the last word in a specific field, and the cooperation went smooth through all the development process.

- Marino Alge is a very talented programmer, with skills both in high level design of software as well as low level high performance optimizations. His role was to lead the programming, designing the class structure, as well as coding everything related to the physics.

- Gioacchino Noris is a hobby artist with knowledge in modeling and texturing. He did some graphics design in the past, writing documents as this one, and has some programming skills. His task was to handle everything related to graphics, from the visual concept, to the asset creation, passing through the graphical engine coding. He also managed the documentation of the game, and the creation of the final movie.

- Alessandro Rigazzi is a computational scientist with a lot of console gaming experience, and good programming skills. His task was to design the gameplay and implement it in the code. He also handled the public presentations.

The team was very active since the early beginning. Everyone had to combine creativity and engineering skills to solve the tasks. The work implied a lot of communication and organization of the work, and thanks to the initial setting of the team, everything went smooth.

3.2 Scheduling and Milestones

3.2.1 Getting started

We started our game development with a specific milestone in mind: having the physical engine written in C# and working on the XBOX360. We considered this as the first step for starting coding anything about the gameplay. The graphics was actually independant. Thanks to the simplicity of XNA, after a couple of hours we had everything we need to visualize objects, so that the physical test could be run, and more ambitious plan for the graphics could be done separately. Here the team worked the following way:

- Marino Alge ported the C++ code of our Physical Engine into C#, a task that went smooth in the sense of coding, but had some major implication on the design of the game, because of the poor performance of XNA and the XBOX, compared to the C++ engine running on PC.
- Gioacchino Noris started the graphics test about particles emitters, decided the early structure of the graphical engine, and wrote the visual design of the game.
- Alessandro Rigazzi wrote the gameplay of the game, considering a lot of possible goals, powers and fun related aspect of the game. His work started with an idea in mind, which had to be reconsidered when the porting of the physics was done.

3.2.2 Performance Nightmare

Once the physics were ported, we immediately figured out that the number of object simulated would have been our bottleneck trough out all the development process. The initial tests in the graphics domain also help to identify the floating operation as something not really optimized running on the XBOX360 with the current XNA compiler. The particles simulation showed a 25% performance drop coming to XBOX360 (the implementation, at that time, made extensive use of the CPU, instead of using the GPU to update the positions of the particles, as suggested by the XNA board). The Physical engine already damped to a 50% of performance passing from optimized c++ to C#, had an other collapse when testing things on the XBOX360 went below 20% in terms of number of object that could be simulated smoothly. Since we had to drastically decrease the number of objects, the game design, in terms of gameplay had to be changed, considering the density of rigid bodies present in the arena. In the graphics part, the bad news didn't have much influence, and the implementation for the engine structure and the following tests about shaders could proceed flawlessly.

3.2.3 The rush to the alpha

This phase was basically to iterate over and over the game logic code and transform a big messy god class into a structured and usable code. Again, the graphics could be developed independently of the rest. The end of this phase coincided with the alpha presentation of the game to the class. We had to hurry up a bit to get it working, but we succeeded.

3.2.4 The last phase

After a good structure was there the game took the fly. It was easy to add new things and tune the existing ones. Everything relevant to the game had three representation in terms of code, one in each of the three domain we were responsible of (for instance the gameplay powers had the equivalent as gravity field for the physics, and as post processing effects for the graphics). Even the big picture of the game code, in terms of software design, had a three pole symmetry with possibly analogue structure in the three domains. We consider this a good implementation, having found the right balance between high level and low level programming.

3.2.5 Summary

The initial idea of the game had to be changed due to unexpected (more or less) performance problems. The bottleneck was represented by the way the CPUs of the XBOX360 performed; the XNA compiler seemed to produce slow code, not optimized for the console architecture, something not so unexpected since XNA is a new technology. Therefore we reduced everything that could consume CPUs computation time, as the particles and the rigid bodies numbers. An other issue that was actually solved to good is the garbage collection, which was heavily reduced by porting everything inside the physical engine from class to struct. The graphics was a separate domain and could be developed in a regular flow. It used to proceed ahead the rest of the development and motivate the team to go on.

4 Playtesting

4.1 Setup

We had one session of playtesting, done with a beta version of the game. We invited six people with different backgrounds about games played, hobbies and interest. Except two of them, the other had a console at home, and knew how to handle a XBOX360 controller. The playtesting session was organized in one afternoon, in a comfortable environment.

The tests were basically a series of matches where the players could play around 10 minutes in a row, cycling in a tournament-like topology, where everybody could play at least against two others. When two were playing, the other four watched. The players didn't receive any hint about the game, but discussed among themselves while playing and watching. This was very useful for us, as well looking the way the players handled the game.

After the play session, we asked them some direct questions about the game, and collected the feedbacks.

4.2 Results

The first thing which was clear is that you need to have the control over the XBOX360 controller in order to play. Especially one invited player had real problems on doing what she wanted to, from the orientation of the camera, the movements around the arena, and the use of the powers. Experienced players, on the other hand, had no such problem and could master the controls in about five minutes of play, where they got in touch of the powers, and tested the effects. Some powers - those related to an immediate visible effect - where very clear from the beginning, while other - for instance the mass shield - took some time to be understood; a process that started with some random but lucky use of them.

The positive comments where that the game, after you understand the way to play it is fun. It has good ideas and pushes the competition. Moreover the learning curve had an high slope, and the players learned to do what they wanted very quickly. One comment which was given more time was about the original concept of the game.

The critiques were also important. One aspect was that the camera movements were, in certain situation not satisfying. Too slow sometimes, and too chaotic in certain area of the map, where the gravity switched direction. An other point was that the host of the player occupied too much screen space. Moreover the controller force feedback vibrated too much. We also received hints on how to tune the gameplay. The green hosts seemed to be weaker than the red and the blue, and these two were not so different, and there were suggestion on what to

do to improve the situation.

Overall the game was appreciated, mainly because it was fun and quite original. The playtesting session helped us in confirm some impressions, as well as getting new ones.

5 Conclusion

We consider this Game Programming Class a success in every aspect. Since it was the first year it was given, we didn't expect much help from the assistants, and in fact about two weeks after the beginning we had far more experience than them on using XNA. Though, they were very engaged and tried to fulfill our needs the best way they could. A major successful aspect of the lab was the communication. Inside the group, between the groups, and with the professor and the assistant, a lot information was exchanged, and people helped each other with good results. The professor staff immediately catched the students comments and suggestions, adapting the class to the real needs we had. This was really appreciated from our part, and helped us focussing on the project. With the lab going on, the students knew each other and shared experiences, suggesting and discussing solutions, and keeping a very cooperative and positive mood. Even in the end when there was a bit of competition, this mood didn't change at all, and we mutually recognized the good efforts of the others.

We are really satisfied by the results of the lab. Given one semester of development we achieved a very good result, and were pleased to see that other groups did the same. We are happy about how we worked as team, and about the fact that we could meet all the deadline and almost all the goals we wanted to. The key element was probably a mix of being friends, the will of creating a game, and having different skills which touched almost all the aspect of the game creation. Besides some small issue, the development was really fluent and everybody was satisfied by the work of the others.

The most difficult task we had to face was probably the performances problems of the setting C# plus XBOX360. But adapting the game design and implementing some good ideas, this could be solved letting us develop a good concept for the game, even if different from what we thought initially. The XNA framework is still young but has helped us especially for everything related to the graphics. Actually one important reason that the graphics development could proceed so quickly and smoothly was exactly the XNA framework. We had active contact with the XNA developers board, which helped us in more situations, explaining the way to proceed to use the XNA framework at best.

As suggestion for the organization of the coming classes we mention the fact that the students should know from the very beginning what kind of game XNA is suited to create. A second point is that the teams should have a various range of skill spread among the team members; this had for sure a lot of influence on our success. Last but not least, that the class should hire some students of this year to help the next ones sharing the big amount of experience done in the course of this lab.

Creating a game is simply amazing...