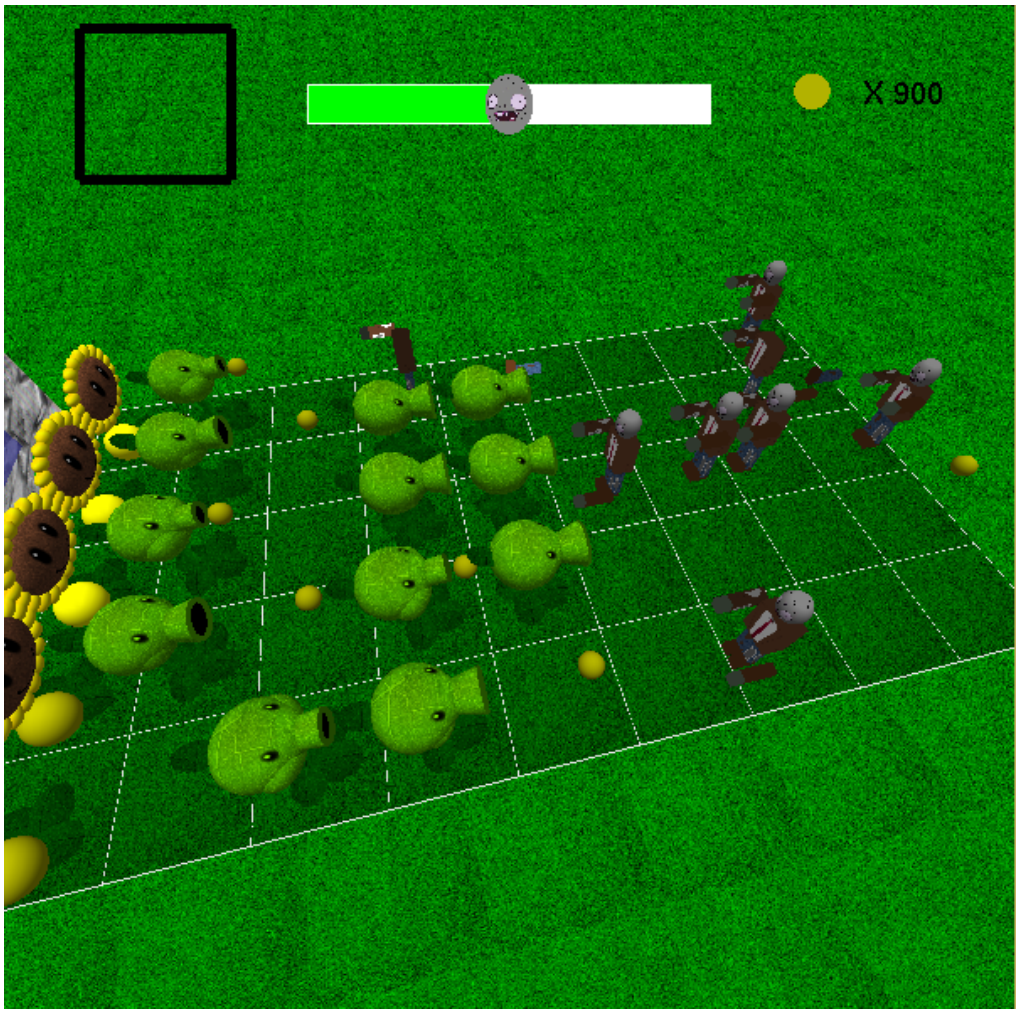
Lucas Brunet 7655819
Yin Cai 7655827

Computer Graphics
COMP 371

Project Report: Plant vs Zombies 3D



Concordia University
In fulfillment of COMP371, Fall 2014

Introduction

This report conclude our project for the Computer Graphics course(COMP371) for the fall Semester 2014 at Concordia University.

The objective of our project was to build a computer game that is substantially inspired by the game called “Plants vs Zombie” (developed and published by PopCap). This “tower defense” game have as objective to defend your house against a zombies army by mean of living pea shooting plants that you will place in strategic position in order to kill these zombies. To create a plant, you need to have a certain amount of “suns” which are product by a special kind of plant, the sunflower.

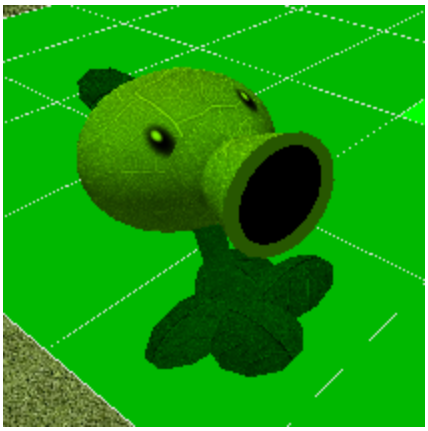
We will first expand on the game feature what will be concluded by a user manual. Then We will show our design before presenting further the implementation and what has been changed with respect to the original design. A discussion of the improvement made since the demo, the improvement that might be done in the future and what we learned during this project will follow before we conclude.

I. The Game

1. Game feature:

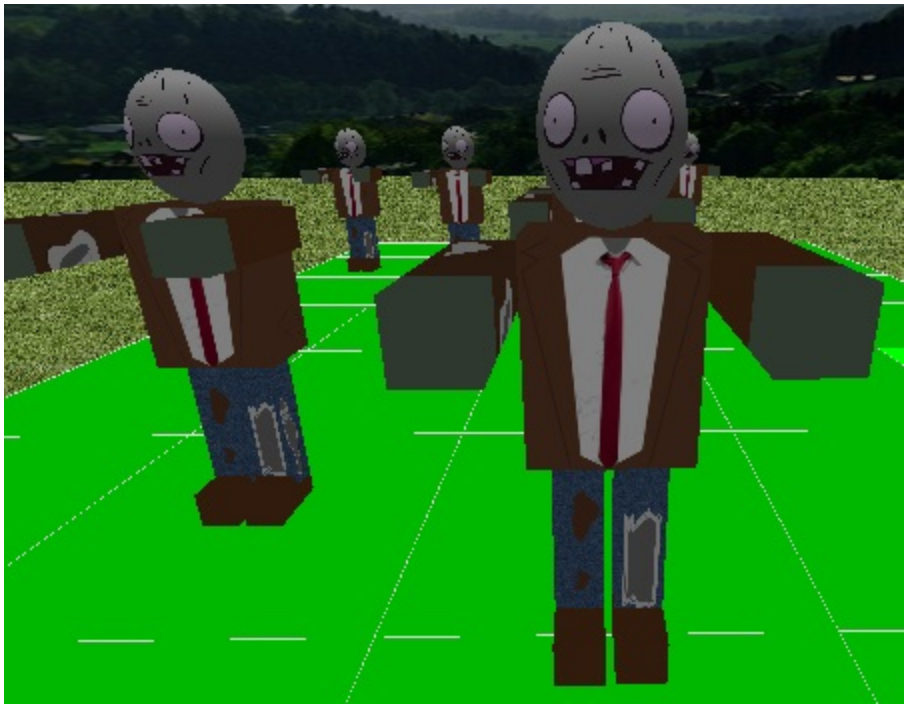
- Plants

In contrary to the original game which offers a great range of plants to defend your house, we decided to focus on the basic plants which are the Pea Shooters and the Sun Flowers in order to favorize quality over quantity. Our model are then as close to the original as possible, textured and animated. Indeed, Sun Flowers “dance” and Pea Shooters move while shooting (As you can see on the video attached to this report).



- Zombies

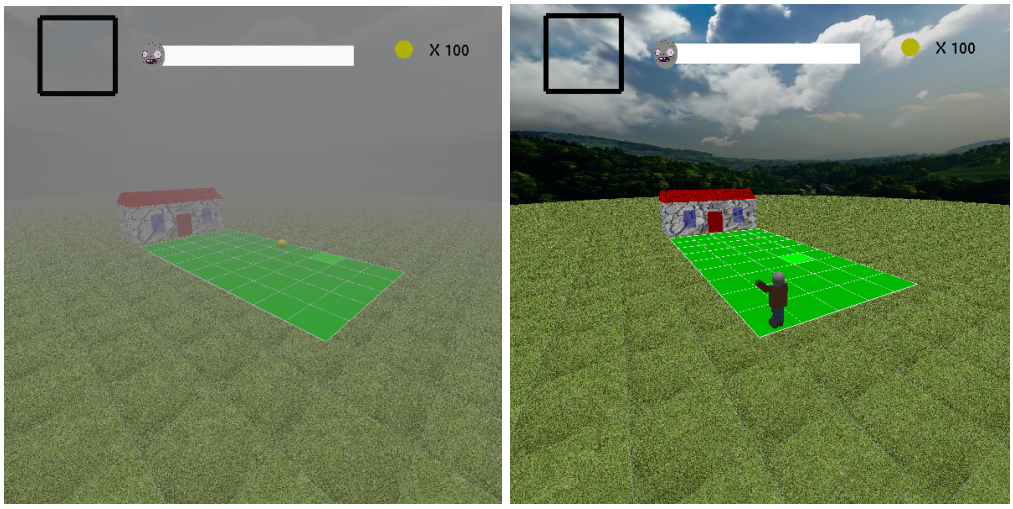
The same way as for the plants, for zombies we decided to recreate only one sort of zombies but to work on the texture and the animation. Indeed Zombies move their arms while moving towards your house and fall to pieces (literally) when they are hit by the Pea Shooters.



- Terrain

The terrain is composed of the GameBoard on which you can plant your plants and your house at one end of it. It is surrounded by a meadow and a skyDome gives the illusion of it being surrounded by mountains.

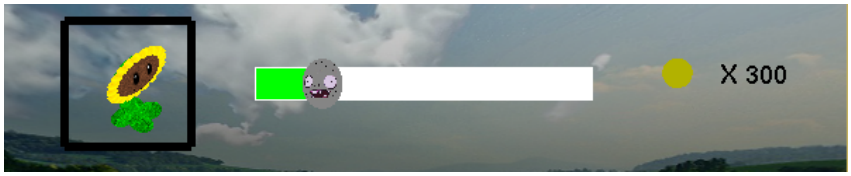
To strengthen the zombie apocalypse atmosphere fog has been added. However, its thickness can be change according to the player preference as explained in the user manual.

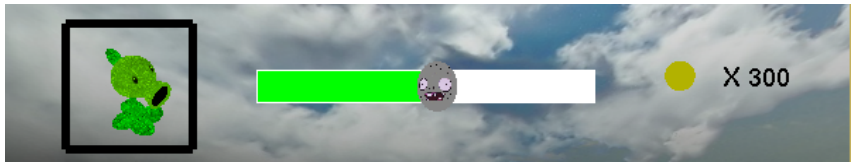


(nb: board tile are textured in the final game version)

- User Interface

To display the number of sun the player have, the progress in the game and which plant can be planted the following interface has been created.





This is also here that Game Over is indicated when the player lose, that is to say when a zombie reach the house.

- Camera Movement

To fully enjoy the scene we implemented some camera movement to allow the player to turn around the scene and zoom in and out.

- Collision Detection

The collision detection is made by comparison of each gameObjects position. It was first intended to be dealt with an external physic engine as “Bullet” or “PhysX”. However physic not being really used in the projects the required time to learn how to use and especially to install this library was a drawback strong enough to justify Using only position to deal with collision.
Furthermore, as in the original game, if you plant a plant on a same square as a zombie, even though the zombie as already exceeded the plant position, it will stop and eat this plant.

2. How to run the game

The Game can be run directly from the application (.exe) in the “PlantVsZombies3D\programme\PlantVSZombie3D\Debug” folder. Else the whole project can be run using visual studio as long as OpenGl and Glut are installed and linked.

For the Mac version, it is possible to run it by opening the xcode project file and run it. Just make sure to have the OpenGL and GLUT library linked correctly to the project.

3. User manual

- Camera Movement

Using the mouse, its is possible to rotate around the scene by keeping the right click down and moving the mouse. An upward movement allow to position oneself over the scene, a side movement to rotate around.
Keeping the middle button pressed and move the mouse forward/backward allows to zoom in and out.

- Fog

It is possible to change the fog density by hitting the “f” key to increase it or “F” to decrease it. This allows to strengthen the grim atmosphere and increase the challenge by reducing the visibility.

- Pick a Sun

To pick a sun it is sufficient to hover over it and press the left click. The counter in the top right corner is then incremented and you can then start planting.

- Plant

Sun Flowers are worth 50 suns and produces more sun, Pea Shooters are worth 100 suns and apply damages to the zombies on their line (9 bullets are required to kill a zombie).
To plant a Sun Flower the numerical key “2” has to be down, what will make the Sun Flower appears in the pre-visualization box on the top left corner. If you have the number of suns required a left click on an empty square of the board will then plant a Sun Flower and decrement your number of sun accordingly.The Sun Flower will then start producing suns.

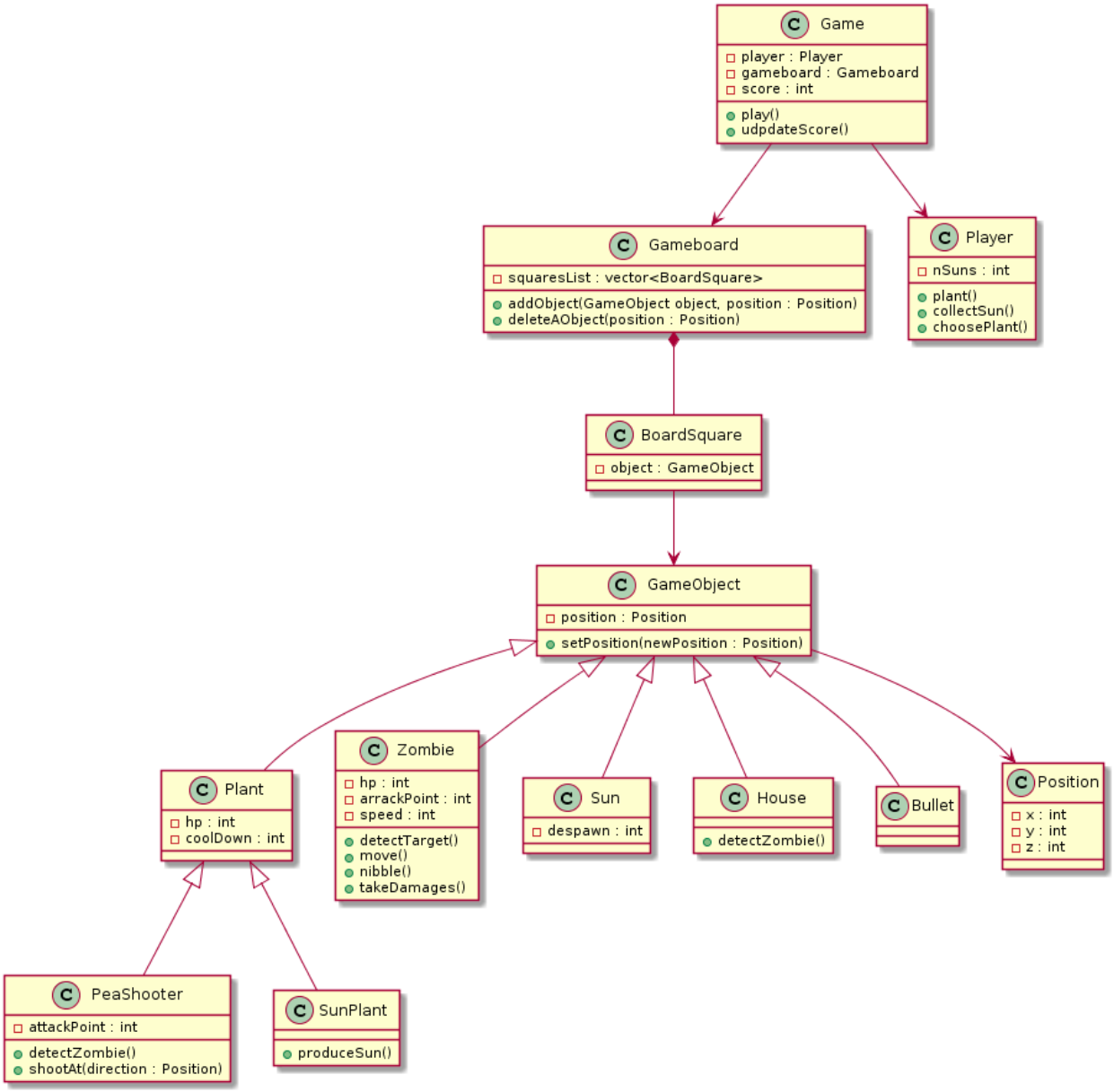
To plant a Pea Shooter the numerical key “1” has to be down, what will make the Pea Shooter appears in the pre-visualization box on the top left corner. If you have the number of suns required a left click on an empty square of the board will then plant a Pea Shooter and decrement your number of sun accordingly. The Pea Shooter will then start to shoot at the zombies.

- Input summary
  - ➔ right mouse button pressed + mouse movement: Rotate around the scene.
  - ➔ Middle mouse button pressed + mouse forward/backward movement: Zoom in/out.
  - ➔ f/F : increase/decrease fog density.
  - ➔ left mouse click on sun hover: pick a sun.
  - ➔ “1” key pressed + left mouse click on an empty square: plant a Sun Flower.
  - ➔ “2” key pressed + left mouse click on an empty square: plant a Pea Shooter.

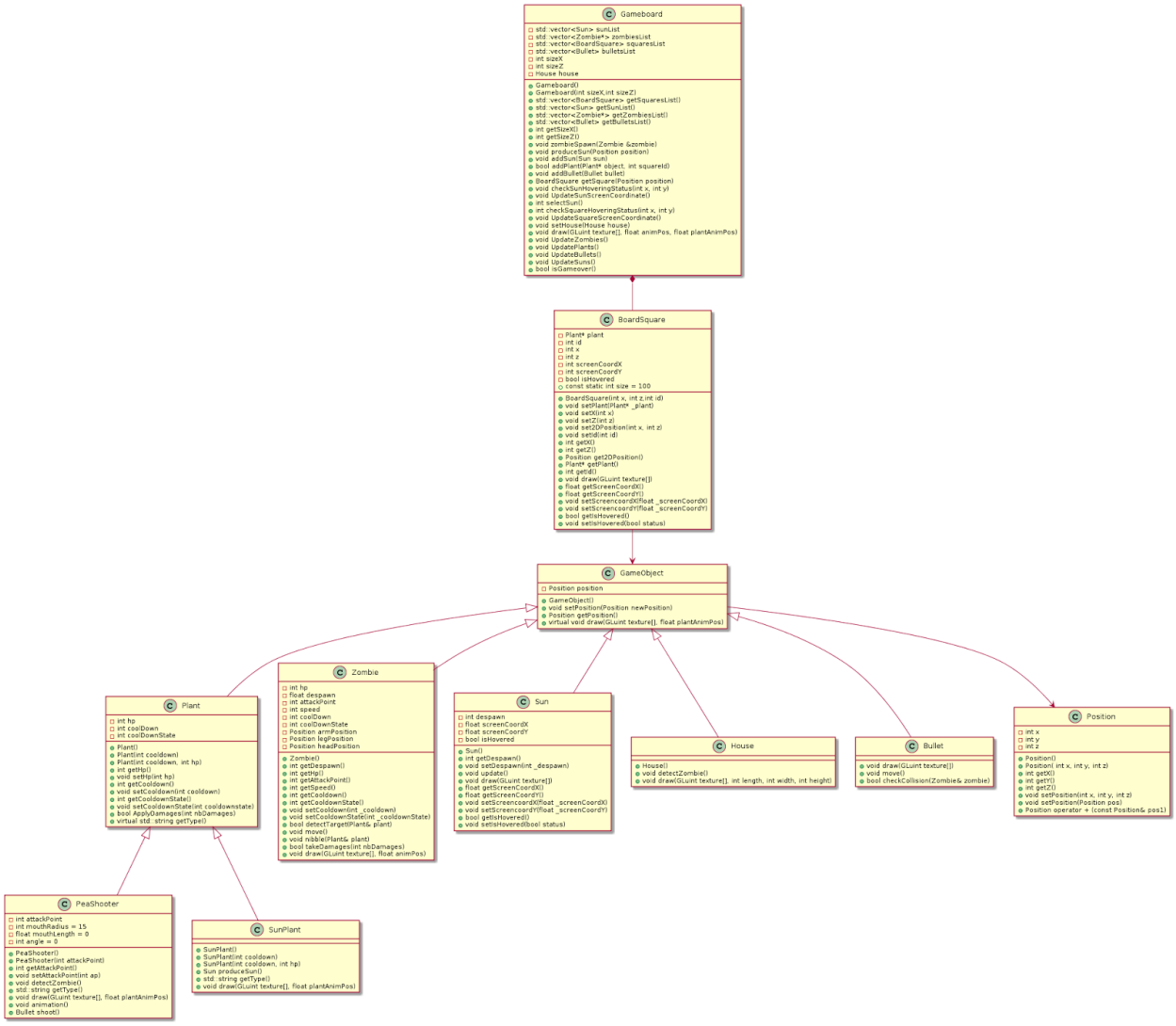
II. Design

1. Project architecture

This is the first class diagram for the project. This architecture as slightly evolved during the development. Indeed, no use has been done from the class player as the different elements that it represents were stored in the main. Certain methods have also been unused or replaced.



Here is the actual class diagram of the project:



2. Hierarchical Modeling

To modelize both zombies and plants hierarchical modeling has been used. Afterward this modelisation also greatly facilitated the animation.

Zombies:

torso
|____head
|____left arm
| |____left hand
|____right arm
| |____right hand
|____left leg

| |____left foot
|____right leg
  |____right foot

SunFlowers:

head
|____petals
|____heart
body
|____stem
foot
|____leaves

Pea Shooters:

head
|____bulb
|____mouth
|____back head leaf
body
|____stem
foot
|____leaves

III. Implementation

1. Global architecture

The project was supposed to be implemented over two different layers : the logic layer and the user interface layer. It was important to separate these two layers in order to create a flexible and reusable application. However, for this project, it was a little difficult to separate distinctively these two parts. In fact, the problem was to have the objects made periodical action (eg SunPlant producing Suns, Zombies moving) independently from other objects. It would have required to implement Timer functions and handling some Threads. With the short period we had before the deadline, we decided to give up this strategy and mix the logic layer with the user interface using OpenGL that will enable us to use the glutTimerFunc() method. So now, each object of the project has its own attributes (eg : health point, position, etc), its own action (eg : move, inflict damages, shoot peas, etc) and finally the methods for the graphical part (eg : draw).

The game is basically handled by the main file, that is to say, the time, the period, the number of zombies to appear, etc. This is also the file that contain all non-game graphical

elements like the sky (sphere mapping), the floor (triangularly meshed with texture), the light (directional light). For the graphics of the game itself, the Gameboard class will draw all the remaining game elements (zombies, plants, bullets, etc). The Gameboard class is also the class that will able to manage all the zombies and the plants actions.

2. Updating the game state

To update the game state and as we based collision detection on our different gameObject position, it was necessary to know where each element was at each frame. The easiest way of doing was then to create vector of elements (plants, zombies, bullets, squares) and to iterate on these vectors. We were afraid that it might slow the game down but thanks to special care in optimising this iteration by avoiding useless elements along with C++ efficiency the game turn in real time as was wished.

Our design making the best out of the object oriented programming paradigm also allows this use of the vector and the call of update and draw function inherent to each of the class. Then each class deal with updating it's elements. This was only a problem when it came to animation as the timer function directing the animation had to be in the main and forced the addition of a variable in the draw method.

With the GameBoard class as conductor it was then possible to link all our different actors and to make the game work.

3. Animation

To create simple animation, for the zombies arms or the Sun Flowers, a timer function allow the variation of a variable in a defined range. It is then sufficient to translate an element of the object or the whole object using this variable to render a simple but nice animation.

Concerning the Peashooters, the animation is a bit special. Indeed the Peashooter are animated when they shoot a pea. There should be a animation in order to make it more realistic. So depending of the time before or after the pea shooting, the drawing of the Peashooter will differ. Before shooting, the Peashooter will bend backward in order to charge its bullet, and inflates its mouth. Then blows the bullet moving back to its normal position. The radius of its mouth changes as well.

Zombies have also some special animations. When they reached a certain amount of health point, they start to lose part of their body. It start with his left arm, then his right leg and his head. Once he is dead, the torso disappear into ashes.

IV. Discussion

1. Improvement since the demo

Since the demo, as we decided to make the game more lively, zombies, pea shooters and Sun Flowers have been animated. Now, zombies move their arms while moving and are dismembered to give an idea to the player of their remaining life. Pea shooters have a shoot animation and Sun Flowers are “dancing”.

The house has also been added and textured to fully stick to the original game. Another addition was the fog which improves the game atmosphere.

The billboard to display the progress of the game, the number of sun and the current plant the player can put on the field as also been remade. Formerly a viewPort, it is now part of the scene what improve the global aesthetic.

2. What follow-up work could be done.

The next step to improve the game would be the addition of other plants present in the original game. Indeed, a variety of plants making the gameplay more fun or allowing to defeat stronger zombies exists. To improve zombies characteristics by making their speed or number of health point increase could follow. The graphics of the zombies can be improved as well.

The addition of sound would finalize this game.

3. What did we learn by doing this project.

This project gave us the opportunity to use concepts seen in class such as hierarchical modeling and to push them forward by, for this example, creating animation. This project also allowed us to reuse texturing and to see how important it was to greatly improve a project. We also learn how to create fog using OpenGL. It also made us think further of the addition of OpenGL to a program and the architecture to use in order to ease the creation of graphics and the use of behaviours defined by the code.

Conclusion

To conclude, we succeed in recreating a videoGame using openGl and to combine both a logical layer which form the gameplay and the graphical pipeline to render graphics that are more than satisfying.

However, it appears that OpenGL, despise its power, is nowaday not the right tool to modelize complex model. Use specific software as Blender, Maya or 3DSMax and import the object seems then a better approach.

Moreover, this introduction to OpenGL made us curious toward the last version of OpenGL which is nonetheless less documented due to its “youth”.