

Final Project Report

1942

Ignacio Leal Sánchez 100495680

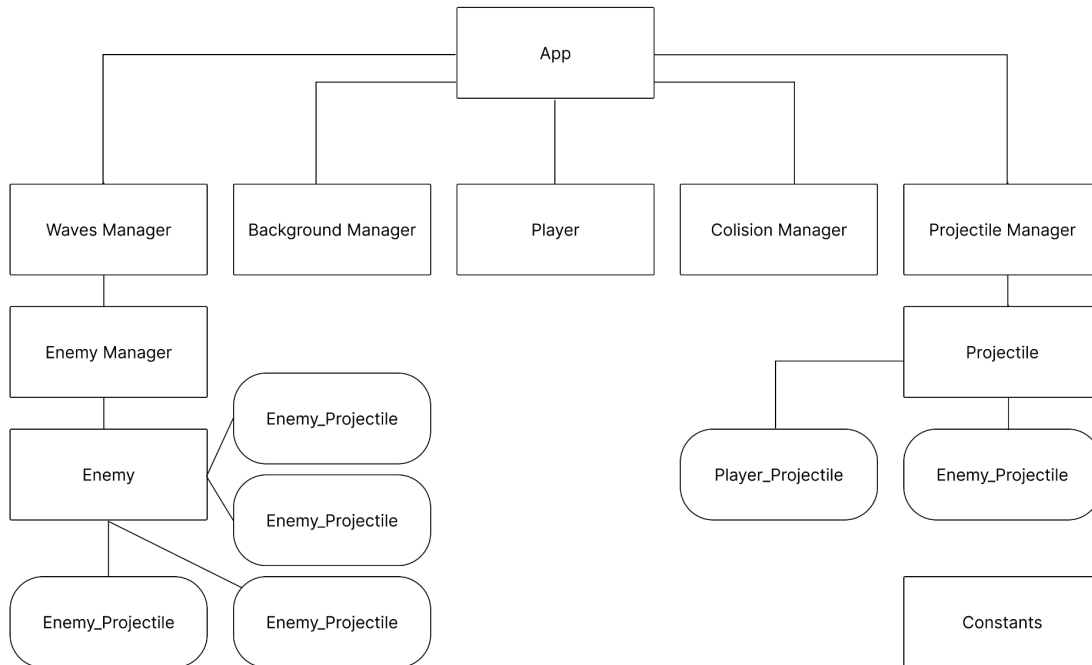
Alberto Pascau Sáez 100495775

Index:

1. [Class Definition](#)
2. [Important Methods](#)
3. [Work](#)
4. [Main Problems](#)
5. [Personal Feedback](#)

1. Class Definition

In this project we used 9 Classes with 6 subsequent subclasses and a file called constants that contains all the functions and values that are not inherit of an specific class



App: Is the main class of the game in the only one being executed and the one that calls the other classes and runs their update and draw methods. It will also detect the key inputs give by the player

Waves Manager: This is the class responsible for managing the appearance of enemies in waves and to grant the player immunity if it is hit. This class manages the enemies by calling the enemy manager class.

Enemy Manager: This class is responsible for the appearance and tracking of enemies this class will have a list with all the enemies on the game and in the update it will check if the enemies are still alive or the are dead if this true it will eliminate them from the list.

Enemy: This is the mother class of all the enemy types in this class. Some values will be given like the spawning position that will be inherited and other values like the enemy type that will be used to determine the enemy that needs to spawn.

Enemy's Childs: All these subclasses each have the data to spawn, animate and shoot for each of the different types of enemies in the game.

Projectile Manager: This class will choose what projectiles is told to spawn and put them on a list that will be drawn and updated

Projectile: This class will manage the two different types of projectiles and check if the have collided with something or not

Constants: Contains important methods used in more than 2 classes and values that are critical to the game and need to be shared between classes.

Collision Manager: Checks all collisions. It is based on a method called collision.

Player: The player class contains all methods for the player, movement, shooting, looping, and animations of the player.

Background Manager: In order to change between the different backgrounds during the game we created the background manager. It receives signals from the app class and changes the background based on them.

2.Important Methods

Constant methods:

These methods are defined in the constants file and are used in almost all managers. They are both given a list and call the update or the draw method of all the elements in that list. It's important to say that both methods use ***for i in range()*** as they need to make changes in the actual elements of the list instead of copies of them.

Update_list_and_delete: This method updates all elements of a list if these elements are alive. Each object that needs to be deleted at a point of the game contains a variable named ***is_alive***. The method checks for this variable to be true, if its not, it deletes the element

Draw_list: It takes a list and draws all its elements.

Enemies, enemy manager and wave manager:

These three classes work together in order to create waves of enemies appearing on the screen. Their most important methods are:

check_delete: This method changes the ***is_alive*** property of an enemy from True to false when the enemy dies.

wave: It is used to determine probability of spawning of the enemies depending on the wave we're on.

create_enemy: It will generate an enemy, the type of enemy is given to the method in a string.

Projectiles:

create_projectile: It will generate a projectile of the type specified in a string

check_delete: This method belongs to projectile and checks the time a projectile has been alive if it is bigger than 3 seconds it will delete

Collisions:

The collision manager class contains four methods critical to the game. These methods change the enemy, bullet and player variables in order to delete them or take lives. However, all these methods depend on one base method: **collision**.

Collision is a generalized method, it was created in order to avoid repeating the same lines of code in each type of collision. Given two objects, the method takes their positions and width and computes if there is a collision using the pythagorean theorem.

Player

The player methods can be divided into two: The methods used to move the player and shoot projectiles given some input, and the methods used to animate the player. Between those, the animations are really what makes the player complex.

animate_move_loop: Of all the player animation methods, this one is the most important one.

Once the plane is told to do a loop, this method will be activated, it not only animates the player but also moves it through the loop. The method also includes a loop cooldown so that the loops can't be continuously spanned.

3.Work

Beginning:

From the beginning of the project, we have organized the work to implement all the elements required in the 4 compulsory sprints, and some of the optional work. The basic structure of the project was decided at the beginning and the different classes that needed to be made were distributed between the two members of the group.

At this point, one of our main focuses was to learn how to create a git repository in order to share and work on the project smoothly.

Creation of new class structures:

After some weeks we decided that we needed some more classes to manage the list of objects that were being created: enemies and projectiles. We also needed a class to check collisions, manage the changes in the background and create waves of enemies. We created a new structure based on the previous one. With these changes in place:

- The movement of the enemies is made by the implementation of parabolic formulas so it is smooth and accurate in real world scenarios instead of the predetermined movements.
- The background changed based on what was happening in game
- The collisions worked and where deleting the objects when needed

At this state of development we created what we call developer mode; a method inside the app class that let us easily check for mistakes while creating new features. Because of the use of git, the process of merging code from both of us was easy.

Finishing touches and improving the graphics:

After all the main - game features were done, we focussed ourselves in the improvement of the experience while playing. This included:

- Balancing the difficulty so that it is easier to play without dying constantly
- Creating new graphics for the background and the first and initial screens.
- Adding some extra features such as the loop cooldown to balance the player abilities
- Adding new possibilities to the dev mode, the most interesting one being an invincibility mode when "i" is pressed.

4. Main Problems

- **Creating a git repository:** One of the biggest problems we encountered while doing the project was the setting up of git. However after some attempts and lost files we managed to make it work out and then it was more of an asset than a burden.
- **Generalization of some properties for all objects:** When first creating the collision manager class, we needed there to be position, width and height parameters for each and every one of the objects that collided, those parameters needed to be called the same to generalize the collision class.
We specified how every type of parameter should be called and which parameters should every object of a certain type have. Even though we debated about using a grandmother class sprite to generalize all of it, because of the complexity of it and our lack of knowledge about double inheritance at the time we ought not to do it.
- **Managing the player and projectile list in order to make waves of enemies work:** This problem occurred when we started to create a wave manager. We had different opinions about how it should be done and had to test different approaches trying to make the code as concise as possible.
- **Creation of sprites in the pyxel editor:** Pyxel has a very simple set of methods, this makes it really easy to use. However, the creation of sprites in pyxel is really slow, as importing them directly into the bank does not work properly and using outside images is not worth as most images are not 16x16
- Other Problems encountered were related with the enemy creation and movement, the very high difficulty or the inability to make loops work.


5. Personal Feedback

Ignacio Leal Sánchez: I have felt like maybe the project was a little bit too hard for the level I had and I would have certainly appreciated a little bit more theory on how to build a project like this one. However I do like a lot of practice based learning and in the real world you will encounter projects like this one in which you will not even know how to start. Overall I didn't really like the project but I think I have learned a lot and that this has certainly made me a much better programmer.

Alberto Pascau Sáez: I agree with Ignacio in most of it. The project was challenging at some points as we didn't have enough information about the creation of multi file projects. The information given about pyxel was not enough. If it is better for the course to choose a library with almost no documentation, at least make it so the basics about installing it, using it and creating basic images or code is explained.

Repository:

Video used to create the collision detection generalized method:

 Python Game Coding: Introduction to Collision Detection

Git repository:

https://github.com/Albrtito/1942_FinalGame_pyxel.git

