**Programming exam: Castles war**

**By Elisa Totis, mat. 506950**

**Menu**

* General information
* Window setup
* Buildings
* Wall
* Tower
* Barrack
* Mine
* Armies
* Swordman
* Archer
* Worker
* Personal view

**General information**

* To make the programming procedure easier, I found it useful to create a file with the “main” coding, in which the loops for the actual functioning of the game were placed and all other additional functions that had to be called, but also the text and background settings; on the other hand, many other different files were created to define the classes of the game, both for the characters and both for the buildings. Additionally, some files were separately created for the constructions of different functions that were not needed in the “main” code, such as the ones for making the animation of a certain character start in a specific situation, or for the different collisions.
* All the different files created were than imported in the initial “import” section.
* To realize this game, the PyGame library was used. It consists of a set of modules specifically created for building videogames.

**Window setup**

* Creation of the windows

Thanks to the function *menu\_display()* I could insert all the desired aspects that I wanted to add in my screen. The first step needed was the creation of the actual screen of the game, with its dimensions and the possibility for the user to exit the screen trough the upper right button “X”.

Inside this function we can also find the background, which was loaded as a .png image, and the text components, which are the title and the command to start the game.

* To start and pause the game

To start and pause the game it was necessary to print on the screen the instructions for both actions and to define the event for the “enter” and “space” buttons pressed.

But also later in the main while loop of the game.

* Game over

Another similar method was used to establish the winner of the game. The program was used to count the two constants (*WALL\_HEALTH*) for each player in order to establish victory or loss. Once the *WALL\_HEALTH* ends up being equal to 0, the opposite player finally win the game. A text is printed on the screen to demonstrate who the victory belongs to.

**Buildings**

Each building has its own class created in different files, in which all the different constants were imported, such as the positions of the sprites.

In the main while loop these lines define the damage of the wall inside the dictionary. Moreover, if the health of one of the two walls comes to 0, the game is no more active.

* Walls

The walls are the main component of the game since they decide when the game is over. A dictionary was created to keep track of the data of each player.

* Towers

The tower is another important element for the players since it creates a defense for the tower. Additionally, to the basic features, the tower is implemented with a function for shooting arrows from its upper corner.

Also the arrows needed a specific class in which the sprites were uploaded and, most important, the update function thanks to which the arrows, once shoot, change their position by increasing x and y axis.

* Barracks

The barracks have the only function to create the armies. For each specific key pressing, a swordman, an archer or a miner is created, and it suddenly pops on the screen.

* Mines

The mines has the aim to produce resources that are necessary to create the soldiers. To activate the production of the resource a worker has to be dispatched from the barrack.

**Armies**

* Swordsman

One of our main units of offense and defence in the game is the swordsman. This unit can be trained from the barracks, run towards the castle, collide with the enemy units, and attack them. This unit, like the others visible features in our game, is a class called Swordsman.

The main point of the videogame is the animation of the characters. To achieve this aim I uploaded all the images of the swordman from different files in the same directory. Than I created the animation, in fact inside the class swordman, a function *\_\_init\_\_()* takes different arguments, which are the possible animation of the swordman.

In the if loop the different animations are activated.

To make the images exchange between the list, in a continuous and repetitive interval, I made use of self.tic and self.index; In this class, as well as the others that I will present, I increment self.tic by 1 till it reaches 6, at this point the index is incremented by 1 and the tic goes back to 0, creating a loop that will control the animation through indexes. If the index becomes bigger than it should, another if statement give its original value, zero, making sure the animations never stop and always start again from the beginning once the images available are finished.

Eventually I created another function to define the collision of the swordman with another component of the swordman army, wall, or an archer in the archer army.

* Archer

The *update()* method of this class is almost identical in construction to the one from the swordsman. The only noticeable difference between the construction of this class and the swordsman class is the attack phase in which I used a different index to make the animation when it is attacking slower. This was done to avoid iterating through the list too fast because there is only two elements on the list.

* Worker

In the *update()* module I gave the instructions, conditions, and limitations to have a correct movement of the worker and a fine fluent animation, with the images of one list exchanging at a pre-define time interval. But before everything, *self.time* makes sure that, after the worker spawns, it must wait a definite amount of time before being able to process any action.

**What I have learned**

When the code does not work properly, or as I planned, the most difficult part was to understand why. Trying again the code by eliminating or «commenting» some parts was really useful to understand in which line the bug was located.

To have a better control over the code, I had to keep testing and running everything to be sure of what I was writing.

Small pieces of code are better to work with.