Report of the First Project

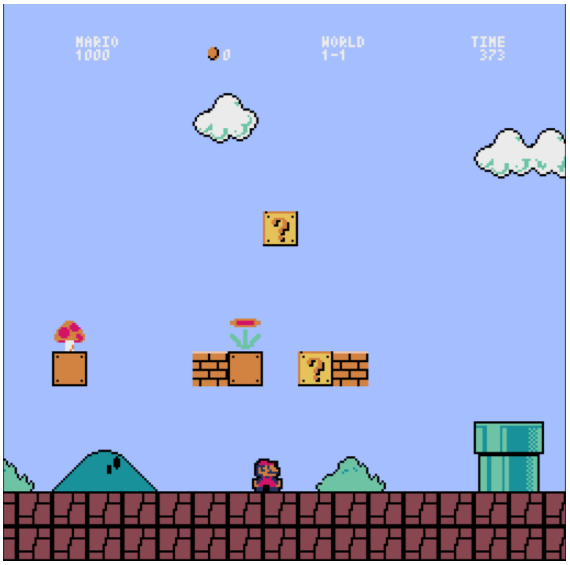
Programming

Creating a game

The objective of this project is to create, using the steps indicated the first level of the game mario bros.In this document we will present the steps followed in the development of the Final project. This document is attached to the code, it is a tool to understand it better.

Alejandro Ponce Fernández and Jaime Salafranca Pardo
Group : 196
December the 20th , Monday

Summary



I- Classes and attributes

We are going to present the different classes and their attributes, even if all the classes are important in the development of the game, we can also classify them by importance.

1. **class Scoreboard:**

In this class we only write the different constants that will be used to increase the score. For example the points given for killing an enemy or for taking a coin. The attributes are the initial score , the points for killing an enemy the points given for taking a:
-coin 500
-star 1 000
-mushroom 500
-fire flower 1 000

2. **class Timer:**

This class is the one used to count and put a timer in the screen. In this class we created two different methods:

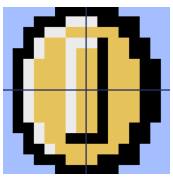
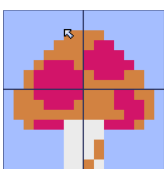
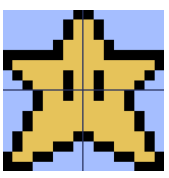
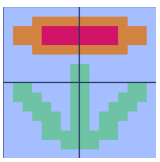
- First one using the pythons library time. By using a loop the program checks continuously if the time elapsed reaches 1 second and if it does it subtracts it from the actual count.
- The second one, is a quick countdown that executes when the game is over. It subtracts “one second” for every pyxel frame. We did this in order to be able to transform the seconds into points by using an animation.

3. **class Star:**

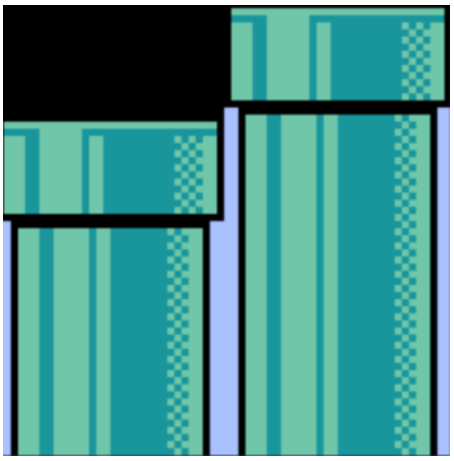
In this class we create each little object that Mario can take such as coins, flowers...
For these objects we could have done a lot of different things like creating a class for each object with multiple inheritance. But we thought that in our case the problem was easier to handle if for the object we had to introduce the type.

The parameters of these kinds of objects are the position with x and y. Inside this class in the init method we define the type with a random number. In function of the different numbers adjusting the probabilities, we give a name for the object (“coin”, “fire_flower”, “star”, mushroom”).

This makes sure that the Object of this class will have a random type. Finally we set the sprite of the object, the same way.

4. **class Pipes:**



This class creates the different kinds of pipes for a given parameter of x and y and a kind of pipe with the different heights alike single, double or triple. This object could be considered as blocks but, in order to make the game more complete, we thought that it was interesting to create that different class.

5. **class Enemy:**

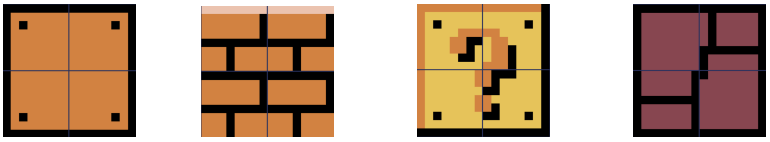
Such as in the precedent class we create an object that represents an enemy the enemy will be randomly a KOmba or a Goomba. Here also we don't do 2 classes but one where the sprite is decided with a random number. Without the sprite there isn't any other difference between the enemies. The other attributes are the position (x, y), the direction that is represented by a boolean where True is left and False is right. There is a final attribute, a dictionary, where we use for keys ("right", "left", "up" and "down") with a boolean that means the collision between the enemies and the blocks. There are two methods in this class for the move that implies a change of direction if there is a collision and the method fall that makes the enemy fall if there is not a collision down.

6. **class ObjectBoard:**

This class creates all the objects in the background, all these objects are very similar because they have any relationship with the rest of the game. In order to create an object of this class we have to give the position with x and y. In addition , we have to add the kind of object, such as cloud, bush or mountain. In order to recreate exactly the same level of Mario we added a castle and a flag at the end. This flag is the only object that has a mouvement in the game.

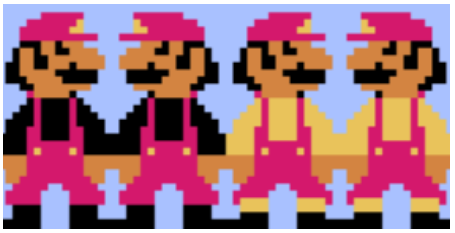
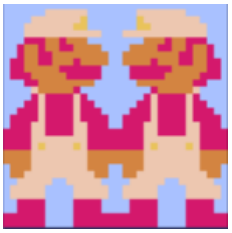
7. **class Block**

The way we created the blocks is the same way we did for all the other elements. All the blocks have the same way to work. The only things that change is the sprite and the block that disappear if they are touched. For that reason we decided to create them in the same class with some common attributes, such as the position and the type of block. In fonction of the type of block that is a string we create the attribute of the sprite. Some examples of sprites are the following:



8. **class Mario**

In this class we create Mario and all his attributes. First, create the attributes of the position given like (x,y). Then we can set the direction with a boolean where True is right and False is left. The attribute displacement will be used to move the board when mario arrives in the middle of the screen. The attribute jumping that indicates if mario is jumping, and a gravity and the dictionary of collisions like for the enemies. Finally we create the attribute of the type and in function of it and of the direction we set the sprite.

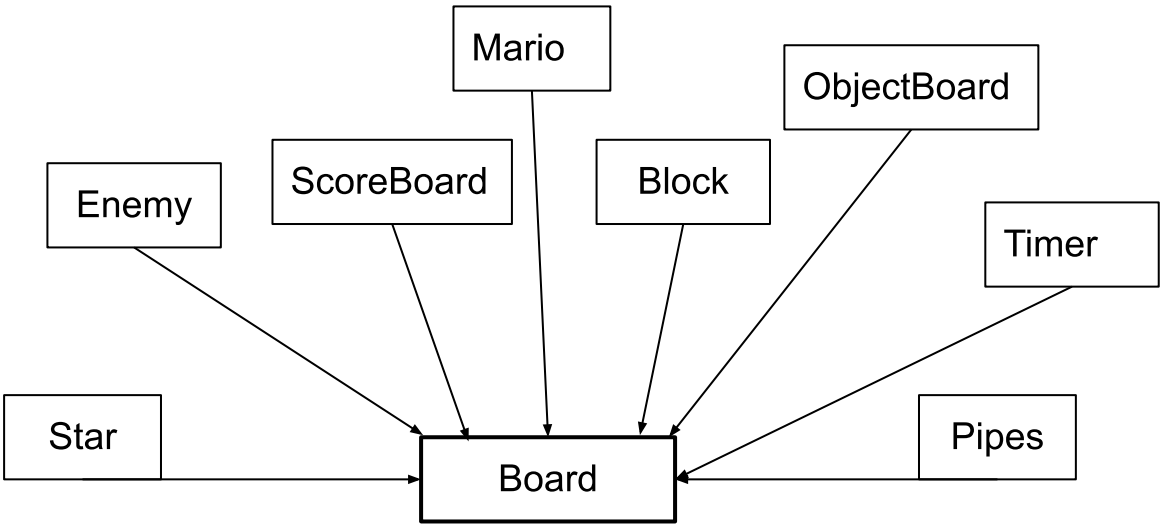




9. **class Board**

This class is the most complicated one. One of the reasons is that we had to create the level completely and create a lot of objects from all the classes. We used loops and lists in order to create all the background. Moreover we can say that all these lists are private attributes because they will only be used for collisions and drawing them.

It is also interesting to say that there are some very relevant methods in this class. The only parameters needed to call a Board are the size of the board and an optional parameter, the number of lives. The relevant functions are for example moveboard, the collisions that we will explain later on. Finally we have the method update that is the one that is used every frame. Each time this function runs we add one to the frame count. Finally we have the method draw that puts in the screen all the sprites and also the information of the level at the top of the screen.

We can say that the classes seem similar one to the other ,except for the Board one, this class serves as a junction between all the classes.



II- Most important methods

We made a lot of methods but among them we can choose some ones that are very relevant and some of them were very difficult to find but are very helpful. For example, in mario we have the method fall and jumping or in the class Board the collisions or the moveboard one.

class Mario

method jumping:

This method is very interesting because of all the changes we did. First we started to do it with frame counts but we needed to import the frame count from Board, so we decided instead to create a counter initialized each time. We also created an attribute jumping that will be checked in the class Board in order to enable the UP_key when mario is jumping.

In the method we also put a speed to the jump and change the values of the counter but we let them like this to make the level easier to play and thest.

method fall:

This method is also used inside the precedent one. Indeed this method is only used to apply the gravity to mario. When mario is not jumping and mario doesn't have a block under him it makes him fall with a certain speed. All this is not inside the move method in order to be able to move left and right while our player is falling or jumping.

class Board:

method moveboard:

The objective of this method is making sure that all the background moves when mario tries to go right but he is in the middle of the screen. To see when we have to use the method we use the attribute of mario displacement that is true when we have to move the board. In order to make it easier we first create a list with all the things we have to move on the screen. Then we move it by subtracting a value to the coordinate x of each object.

This method executes differently if mario is in "star" mode as he moves faster the background must too.

method col:

This method checks the collisions of each block and mario and runs the different consequences in the different blocks.

Our first approach to this method consisted of "activating the collisions" when Marios "x" or "y" were the same as the block we were checking collisions with. However we realised that this was not always working as the function was updating every pyxel frame and as mario did not move one by one pyxel sometimes the program did not realize they were colliding.

So then we had the idea of creating some interval in order to adjust the collision. This solution was in some way better as even though it is not perfect it works in most of the situations. Then we thought about creating a class for the functions of the collisions, we thought that it was the best way to solve the issue. After trying it we didn't get a good result so we choose to use it like a method of board even if you are not completely satisfied with it.

Inside this method we also change the sprite of question blocks when hitted and make disappear the breakable blocks, by removing them from the list. This only works because we execute the method inside the update one.

We also had to make some changes in order for the collisions to work also with super mario as its width and height are different.

method collision_block_enemy:

This method is in some way very similar to the other one, the difference is that it only applies to the enemys and that we don't need to check the upper part. This makes the condition for the condition a lot easier to do with a better result. That 's why we don't use a unique method for those collisions.

Moreover,here the only thing that changes in the method appart the boolean of the collision is the direction of the movements of the enemies.

method collision_mario_enemy:

We use this method to check if mario has killed an enemy or if an enemy has killed mario. The things that we have to take into account this time are the type of mario, because if he is in a "super" form he only regains his original form. And also the number of lives that we subtract each time he falls or touches an enemy.

In this method we also restart the level if mario is touched by an enemy to do that we decided to call the ___init_- method of board.

III- Relevant attributes

self.collision

For both Mario and the enemies we created an attribute called collisions which was a dictionary with keys up, down, left and right, and all of them were booleans. This allowed us to check for every collision in every pyxel frame and easily return them to false every frame in order to recheck it.

self.lives

The private attribute lives is very interesting and relevant. Because it is the basis of the game. Each time the normal mario is touched by an enemy the level is restarted and the number of lives decreased. The same happens when Mario falls out of the level.

self.___block

This method is also very relevant. If we see the different objects we see that some objects are considered the same way in order to find the collisions. But in other methods we have to see the differences. One solution to this could be creating mother classes but another way was putting in a list as a private method of Board all the objects that appear in the board. This is very interesting and helpful and indeed it is the list we use in the methods of collisions.

IV- More information about the work

In this work some sprites were asked in order to complete the game but there were also some optional functions. Here we are going to explain the slides we did and the optional and other extra functions we did.

Sprite 1: The objective was creating a board with mario that can move and add more than 20 blocks and some extra elements such as clouds and bushes.
Sprite 2: In this sprite Mario had to move and move the level. For this we used the attribute of mario displacement and the method of board moveboard.
Sprite 3: In this sprite we implement the enemies with the properties and the movement of the enemies.
Sprite 4: In this sprite we had to code some basic functions of the game.

Optional elements:

- We added the fire_flower that transforms mario into fire mario but we were not able to add the fire balls.
- We were not able to make the invisible block for one extra life but we created a Star that one time taken, increases mario's speed.

We couldn't do any of the other optional elements.

Extra elements:

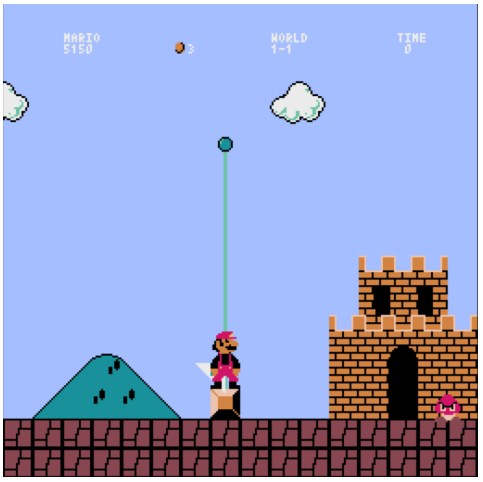
-Even if it was not asked we created our level exactly as the first level of the first mario. That's the reason we have a big list at the start and not a loop because the object had some exact positions.

- We added a castle at the end like in the first mario.

-We added a flag that goes down when mario arrives at the end.

-Instead of transforming the remaining seconds directly to points when the game ends, we did it by lowering the seconds very quickly (one second for each pyxel frame) and transforming it progressively to points. This creates a sort of animation.

-Finally we made the enemies rebound when a collision was made, we didn't know if that was compulsory or not but that made our game more realistic.



V- Problems found during the work

How to make the collisions ?

We had a lot of doubts about how to make the collisions, we were not sure if we had to create an extra class in order to make the collisions. One of our ideas was to create for Mario and each enemy an attribute for each side that could have a collision. But that was the synonym of creating at least 3 attributes more. So finally we decided to create a dictionary with a key for each side that could have a collision.

How to check the collisions?

Every object in the board has some coordinates but these coordinates are a point while the object usually measures 16 x 16. We first thought about checking if the coordinate was the same in base 16 but that was not precise because the first contact was made before tmario and the object has the same coordinate in base 16. So we decide to compare the coordinates of mario in a certain interval.

Pyxel frame count

It is true that we needed to use the pyxel frame count in order to move the enemies but we thought that this method was not so interesting with the changes of direction. Even if we did it like this at the beginning we thought that it was a mess and we decided to change the way of doing it even if certainly it is not the right way to do it.

Jump method

When we started to make the jump method we thought about making a loop but that was not very helpful because it was an infinite loop. That's how we came up with the idea of making a condition and creating two attributes one to see if mario was jumping and another that was used as a counter to repeat the condition a certain number of times.

Super mario

When we transform Mario into super mario or fire mario the way of taking his size and measuring the collision is a little bit different. Even if we have solved this problem for the collisions with the blocks, there are some problems that aren't solved. For example when mario takes the mushroom he passes through a block. In addition, sometimes when he passes through an object he can not take it. Finally we also have an error erasing the enemy that have been hitted by a "super " mario, we couldn't find a solution to that.

VI- Conclusion

In conclusion, we can tell our perspective of creating this program. In fact, at the start, arriving at a result as we did seemed impossible to us. Little by little, we started to think about solutions to our problems. Once we started to understand how to work we advanced very quickly. But our code was kind of a mess.

Then we started a second period, when we tried to come across the issues and make better code, cleaner and with more comments. We saw that that was the best way to solve problems while programming.

We spent a lot of time making our code understandable and reducing its length by grouping similar parts. We realised the importance of keeping a clean code and with as few repeated parts as possible in order to make future changes easily.

We know that for some method or some classes we could have done it better, but we also think that this project was very helpful for us. Through this project we have been able to understand how Object Oriented Programming works and implementing it into our own video game has allowed us to get a better understanding of it.

Lastly, working in pairs is very helpful in order to be open minded and learn how to code and see the issues in a different way.