PROGRAMMING PROJECT REPORT

MARIO BROS

Eduardo Alarcón and María Isabel Hernández

Group 89

Computer Science

This project aims to emulate the game Mario Bros for the NES (Nintendo Entertainment System). After following the steps indicated on the project requisites, a final near-functional version has been reached. The code has been designed taking into account future maintainability, code understandability and new game functions as well as an easy way of editing the level with the pixel editor.

Design of the classes

Board: this class is the body of the game because in it we call all the functionalities: we get the map from pyxel, we constantly check the collisions, we draw and control many of the items and we get the movement of Mario to be controlled by keyboard.

Mario: the class Mario contains the features of Mario itself (that is, its size, sprite, lifes, score, time and coin counter) and also the functions for the collisions with the blocks, the blocks, the coins and for the mushrooms.

Question Blocks: with this class we can create objects that have a determined position and container. The container is either a coin, a mushroom or nothing. The breakable blocks contain nothing and the proper question blocks contain an item, but both types of blocks are of the same class.

Blocks: the class Blocks is for the blocks of the floor and the pipes, that is, the block with which Mario will collide. However, the collisions with the breakable blocks and the question blocks (which are both of the class Question Blocks) are controlled in a different way because Mario not only has to collide with them, he also has to interact with them as they can break or provide items. This class also contains a list with all the blocks, pipes and floor mapped.

Pipes: we use this class to create the pipes as objects, taking into account their position, but these objects are created in the file “blocks”.

Floor: we use this class to create the floor as an object, taking into account the position of each of the bricks that form the floor, but these objects are created in the file “blocks”.

Mushrooms: this class is for creating the mushrooms taking its coordinates. It contains the sprite of the mushrooms, which are created when some question blocks are kicked (in the file “mario”).

Coins: this class is for creating the coins taking its coordinates. It contains the sprite of the coins, which are created when some other question blocks are kicked (in the file “mario”).

Enemies: this class is for creating enemies that will have some position and a type, which is whether Koopa Troopa or Goomba. Each type of enemy has a different sprite, but they are created with the same class.

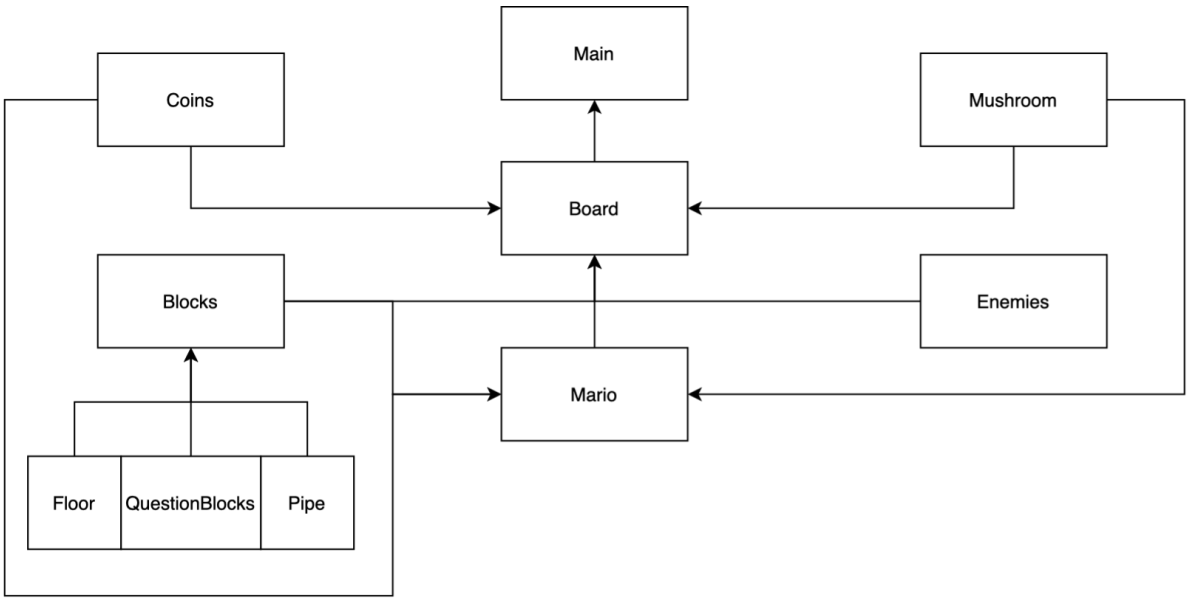

Diagram of how the classes are connected to one another

Main attributes and methods

react_on_collision: This function is responsible for letting mario move. It has different calls to the check_tilemap or its derivatives. As attributes, the function has the x and y position of mario and how much mario wants to move in the next frame, divided into a vector with the x and y coordinates. What the function does is check, via the tilemap_check_collision if the path is empty and allows mario to move, adding the “sign” to the desired quantity of movement

check_tilemap_collision: This function gets the coordinates of mario, and adds them to the width and height of the player, which then checks, in the range, if mario is between two tile, which he always is, as the tiles are 8 by 8 and mario is 13 pixels wide and 16 pixels tall when they are small and 16 by 32 when they are big and divides them by 8. This then checks the tile where mario wants to go and if it is in the list of tiles mario has to collide with, in this case self.player.col_tiles, (collidable tiles)

update: We will divide the update method into different sections:

- The first part is dedicated to mario’s movement, if some specific buttons are pressed, mario’s desired position will be added to be checked in the other methods.
- Then we have the movement and the scroll of the map
- The third part is the one related to controlling time and deaths

- Lastly, we have the collision made with the different objects, the destruction of a block, the collision with them, the growth with a mushroom and picking up coins

kikblock: the function kikblocks takes each question block (also breakable blocks) in the question block list and checks if Mario is colliding with it from below. If Mario is big and the block is breakable (contains nothing) that block will be broken, that is, removed from the list of blocks. If a block that contains an item is kicked a coin or a mushroom is created and added to the list of coins/mushrooms.

grow_with_mushroom: this function takes each mushroom in the list of mushrooms and checks if the position of Mario is the same as the position of the mushroom. If so, Mario grows to Super Mario, so its size, sprite, position and score changes. Also, the mushroom is removed from its list.

pick_coin: this function takes each coin in the list of coins and checks if it is in the same position as Mario. If so, the coin counter value increases and the coin is removed from the list.

collisions_with_blocks: this function takes each question block (also breakable blocks) in the question block list and checks if Mario is colliding with if from below or from above comparing the position of the block with that of Mario and stopping Mario if they are similar.

game_ends: The game ends method resets the entire game by putting mario to the original position of the game, resetting the time to 0, removing one life, and making mario small.

Main algorithms

The most important algorithm that has been used in this program is the draw function dn the update function, that execute every frame of the game, one prints into a screen the player, the enemies, the blocks, the map and the player and the other one, the update, handles most of the logic of the game, executed else every frame. For example, in this algorithm, the collisions are handled, the movement, and the scroll of the map. The update is divided into the classes, for instance, Mario has an update that checks if he is big or small, to change the width and height, if he is colliding with a mushroom, if he needs to pick a coin and if the block needs to be destroyed.

Another interesting system to mention is the collision system. We created a map that is made up of 8 by 8 pixels tiles that are the ones that check for Mario's position, collision. This means when editing the map, we are effectively editing the level, without needing to code every single object. Although this seems to be easy, we also created an object for every block in the map, that we then used to check if Mario could jump only if he was standing on it. This was made possible by mapping every block with coordinates and checking if the coordinates of mario are similar to any coordinates of a block that collides.

All enemies that we were able to create have been static as the collision system with object did not work for us. This way, the enemies can kill the player and to be easily placed by another user, opening the map editor.

In the demo, the player can move, jump and dodge enemies, and die by touching an enemy or falling though the ground.

To be able to count the time mario has been playing and reset it, we use the pyxel.frame_count method and a variable of elapsed time, that is added one, when the division of the elapsed frames by the fps is 0 and then is subtracted to the total time the player has to end the level.

Work performed

In this project we have recreated the first game of Mario Bros. We achieved to implement completely the features of the sprite 1 and 2 and to code the sprite 4. About sprite 3, we did not fully code and implement the enemies.

In our program, Mario appears in the superior left side of the map and falls as if there was gravity. You are able to control the movements of Mario by keyboard with the keys “W”, “A”, and “D” or with the arrows. Mario collides with the floor and with the pipes, but not with the blocks. When you reach the end of the map, where the castle is placed, you win the game and a congratulation message is shown. If Mario falls in a hole, if an enemy kills him or if the time runs out, Mario loses a life and goes to the end of the map. If three lives are lost, the game ends and a “Game Over” message is shown.

We had some issues performing this project. We had a code with all the collisions working properly, but when we started to program the functions that make the blocks break or provide items, we had to change the collisions on the blocks, so we have a collision system for the floor and the pipes and another collision system for the blocks. The first one works but we didn’t achieve the second one. As question blocks do not work properly, mushrooms and coins do not appear in the game. Also, our enemies are static, they can interact with Mario, but they do not move.

As an extra element, we implemented a flagpole that makes mario grow, as if it would touch a checkpoint in the original game. We also print the x and y coordinates of mario each frame to be able to see where mario is at every moment, as well as small sprites of the lives mario has and the coins he has.

After a lot of work, we have created an object, that has some kind of collisions with mario and if they are hit by some direction, they break, leaving a coin, or a mushroom behind, that you can collect. The mushrooms makes mario grow and the goombas make mario decrease in size or reset the level. All of this is coded into the game, but it sometimes does not work.

To recap, we, as a team, have decided that the best way to submit this project is to have two different games, one, where the collisions work flawlessly and the other with the objects and collisions working but with some errors, this second one has different funciotnalities as well as the firs one. For example, enemies kill you in the first one but are static but in the second

one, when enemies appear, if they appear, they will move but they will not kill the player (the code is written down but for some reason, it is not executed propperly).

The last funcitonality we added was the change of wardrobe. In both games you can play as mario, our red plummer, like Luigi, his gren friend or as Wario, his purple enemy. To switch between them, you only need to press the “M” key to switch to mario, the “L” jey to switch to luigi and the “O” key to switch to wario. This can be done on every part of the program.

Conclusion:

During this project, we learned using python can be very stressful, but with the right time and patience, and reading the error codes, we were able to solve many code errors that prevented the program from even loading correctly. Creating the sprites was fun at the start, but we could not imagine how difficult it would become. Starting from scratch in a new library and in a month having a playable game, without major errors is something not many people could accomplish. The most difficult part of the project was the collision system of the blocks and creating the different objects for every block and entity.

The tool that made this whole experience easier than what we expected was the plugin in Pycharm called Collaborate, that allowed two people to code in the same local file at the same time. What we were missing was the ability to have a repository where we could connect and have the code accessible at all times without needing one of the participants to boot up their computer.

As a review, although we know that this project is meant to teach us how to use OOP in Python, we think that it would be interesting to make a game in a platform oriented to videogames creation, like Godoot, or have some kind of help making with the library, as we went without any kind of help more than just a PDF and we had to dig through japanese web pages to find any helpful resources. Even going to the subreddit of pyxel.