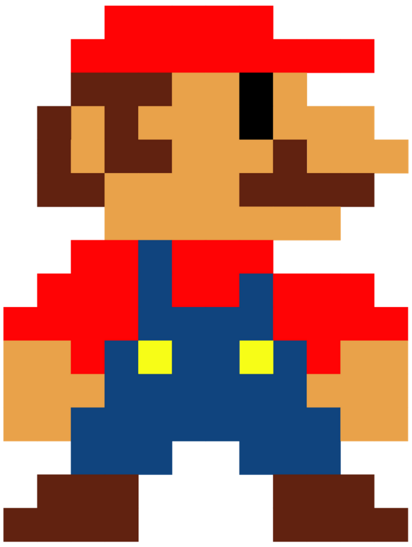
Mario Bros 1983



Authors:
Collin Martin
Jorge Aranda

11/12/2023

Group 96

Index

Introduction

Mario 1983 is a game designed to be played individually which consists of various levels. Mario, the main player can move around a map with 3 different heights created by successions of blocks. Enemies spawn from the tubes located on each side of the upper part of the map. Mario's job is to eliminate these enemies, all of this while collecting coins and making sure he does not get hit by them; as he will lose lives and eventually die. Both collecting coins and eliminating enemies increase your score on the game.

Instructions

- The instructions to run this game are rather simple.

-myresource.pyxel, compact.pyxers and TitleScreen.pyxers must be stored in the same folder as the project running in Pycharm is. To execute the game, main.py file must be run and space-bar pressed.

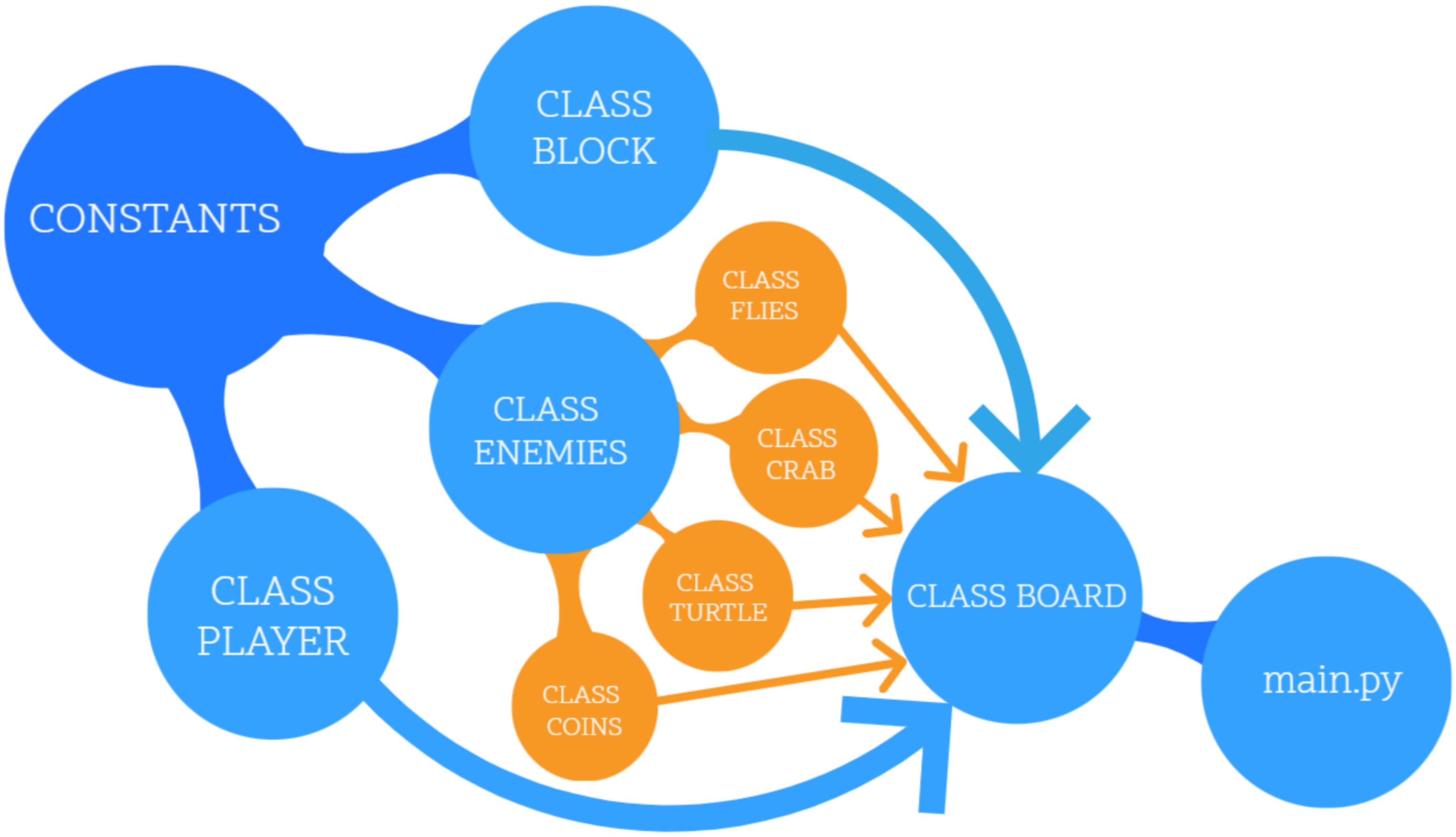
- Inside of the game, the instructions to actually play are the following.

-Mario can move left, right and jump into blocks but he cannot go through them.
-Mario has 3 lives and will die after being hit by an enemy 3(check) times.
-He can eliminate enemies by flipping them (hitting the block just beneath them) and then kicking them. This will increase the score by 500 points.
-Turtles and flies take a single hit to flip, crabs take 2.
-The POW block flips all enemies but can only be used 3 times.

-Coins collected increase score by 800 points.
-There is a “Scoreboard” and a “Live counter” on the upper right and left corners of the map, which keep track of your score and remaining lives.

Designed Classes

We have used a total of 8 classes on our project. Inheritance and multiple inheritance has been necessary to save time as well as maintaining an orderly and clean code. The following diagram shows the relationship and inheritance between classes. After it we will proceed to explain the methods and attributes inside them.



Brief explanation of the diagram. The Constants module was used primarily to store our sprites. Class Board essentially imports classes from all the modules. By setting it out like this, the main.py (which actually executes the game) only imports from “Board” keeping the code inside this file under 15 lines. Coins, turtles, crabs and flies inherit from the ENEMIES CLASS, they have very similar attributes and methods, thus saving a lot of time.

Classes-Outline of the main attributes and methods

Class Enemy:

The Enemy class was a parent class for Turtle, Flies, Crabs, and even Coins. The Enemy class included everything the child classes needed: such as movement and gravity implementation. Some of the main attributes were a boolean indicating whether it was alive, speed, checking for collision and implementing a pause (for staticness when flipped) and setting their constant direction (either continually right or left) with a simple random.choice. Except for pause, Coins could use all these attributes thus the reason we implemented it as a child class.

Individual Enemy Classes:

The child classes of the enemy used a *super() innit* to inherit everything from the enemy class. The only features we had to implement was the random direction from the enemy class and call it in a move method for each child class. The other method was the sprite method that drew their sprites depending on the actions they fell under. The sprites are dependent on each action they are doing. These actions include checking if paused(flipped) or checking the direction the enemy is going.

Class Player

The player class was one of the most important classes we had to implement. This class included many methods that were crucial to make our game. One of the most important was the Update method for Player.

Method: Update

This method is used to give Mario controllable movement. As opposed to the enemy Mario doesn't move in a random constant direction but can be maneuvered. This is done with the left and right keys.Inside that method we also allow Mario to jump when the spacebar is pressed. After checking that the spacebar is pressed and that he is not already jumping, (to avoid double jumps) its vertical speed is set to the jump strength assigned in attributes.

The *def register_hit(self)* is another method in the game that was crucial for our game to be run. It gives Mario a short period of invincibility after it has been hit by an enemy. This avoids the same enemy taking away various lives after a single collision. We implemented it after our first runs on the game because we were finding out that it was an issue.

Class Board

This in question was the most important class for our game to run. This constantly updates the sprites of each enemy, player, and coin while checking movement. Also it runs every updated x and y position for each of these characters. By doing this it also has the responsibility to check the collisions between all updated values for the positions of the characters. while checking all positions and keeping all of our created classes updated it cycles through levels by checking the status of the game. Through each status of the game it also clears and redraws the blocks of each level needed for the game to be constantly updated. Class board is by far the most important game class we have in our code.

Class Main:

The main.py file has very little code in it leaving it very ordered and clean. This is because everything needed is imported from other files. The main file imports the board module (which has all of the classes inherited into it) and then is responsible for constantly drawing and updating the board while running the game. It is the actual file you must run in order for the main screen to appear.

Work completed

Through intensive hard work and effort we have created a Mario 1983 game we are proud of. However because of several complications in our code, some details were very difficult to perform. After the process we made to check the collisions, it was hard to keep count of how many times a player touched a block or enemy.

In consequence we are missing two elements of our game that we are sad to say goodbye to. These elements are the pow block disappearing and not functioning after 3 hits, and crabs having two lives. The complications in our code made it very difficult and time consuming to implement these elements so we decided to leave them out.

Description of the work carried out

The game starts out with a very simple start screen that requires you to press space to start. Then runs the update method mentioned before to run the elements of the game and also cycle through levels. During each level a certain number of enemies are made through a series of random numbers then appended into the code.
If and when the enemies collide with Mario the enemy is removed from the list it was originally appended to. A method used to spawn the enemies required for each level was made and implemented as the game *state(level)* changed. To kill the enemies we constantly check for collisions between the block the enemy is on and the block that Mario top collides with *(jumps into)*. Next, the enemy is paused and able to kill. If not killed

in a certain amount of time (a timer mechanism that uses pyxel.frame_rates) it gets up, changes color, and increases speed. However, if Mario were to collide with an enemy that is not paused. 'Register Hit' is activated and Mario turns invincible for a certain amount of time. If the player manages to kill all enemies for the given level, that is: 5 Turtles in the first level, 7 Crabs for the second level, 7 Flies for the third level, and finally 11 enemies determined at random for the final level. If one manages to complete the game. A **'WINNER'** screen is displayed allowing Mario to dance around on floating blocks.

Conclusion

After the final moments of our project we looked back on the code we created and felt proud. However, there were many complications, bugs, and hardships during the process. Some of the main bugs we encountered had to do with the collision methods as it was the most difficult to implement. Although, through the many bugs we faced there was nothing better than finally getting something to work after intensive reflection.

Developing Mario 1983 really tested our understanding of python as a whole. From the implementation of object orientation in our Mario game, our code seemed to be way more organized and reduced the lines needed. On top of the skills we used in class we added additional resources from online services to further help our code.

In the end, the project was a great tool to further help us understand the basics of python. Through a lot of hard work we had made a Mario 1983 game that was very neat. Object Orientation, comments, and our skillset made the code of the game easier to understand and readable for further analysis and bug fixes.