Final Programming Project Report

Alejandro Pérez Bueno &
Alfonso Larumbe Fernández

December 10th, 2019

Instructors:
Yolanda Escudero Martín
Juan Luis Vicente Carro

TABLE OF CONTENTS

1. CLASS DIAGRAM

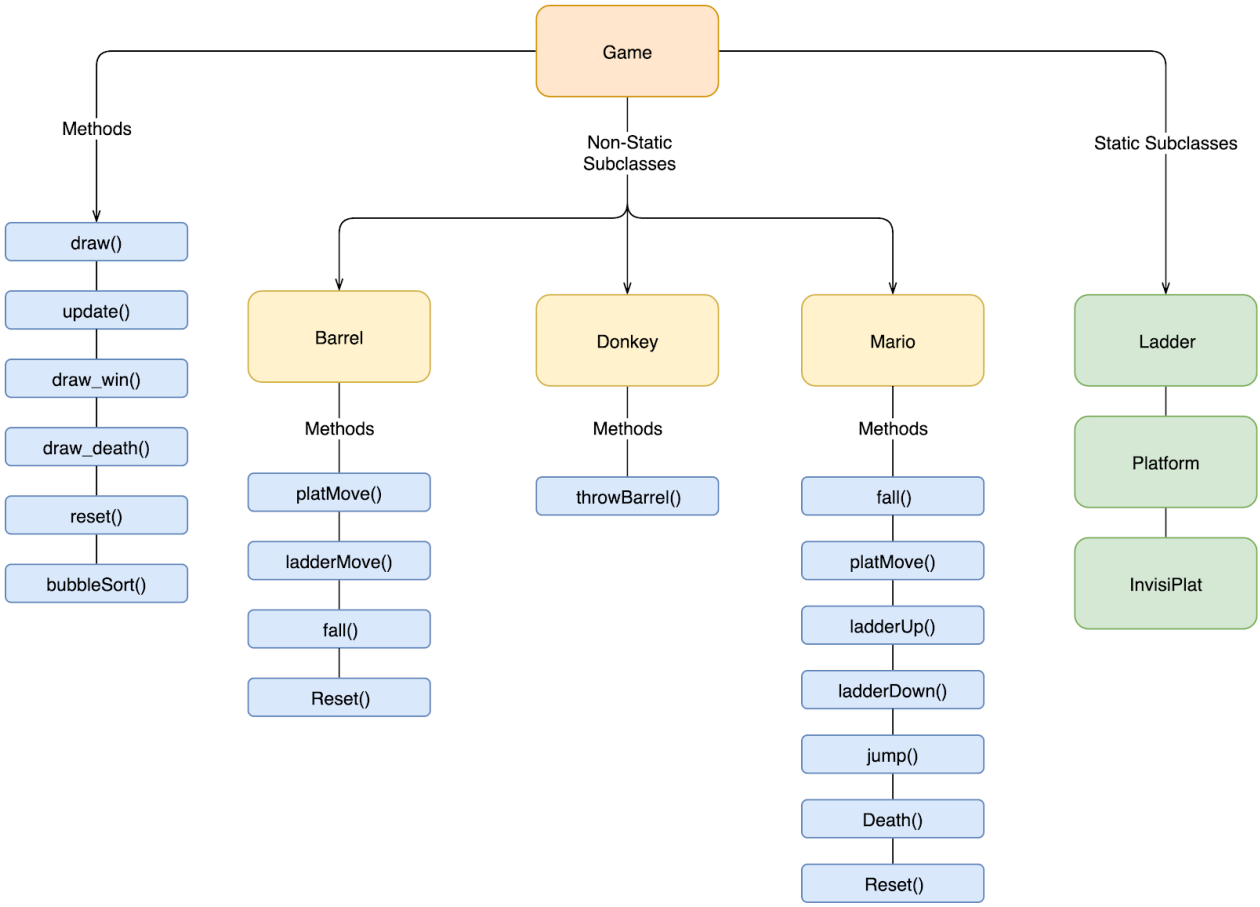

Fig 1: *Class Diagram*

In our project, we split all of the classes in yellow (see **Fig 1**), which represent non-static objects in our game, into separate files. This way our code will be cleaner and more truthful to the concept of Object-Oriented Programming (OOP). On the other hand, we kept all the classes in green, which represent static objects in the game, in the same file (static.py), since we considered that these classes are not very elaborate and can coexist inside the same file. As a remark, we chose not to include an extra class for the princess, Pauline, since we did not need to store any useful information about her in a new class, i.e. we did not need to keep track of her position, movement, and so on. Therefore, we simply drew and animated her, without creating an entire new class.

2. MOST RELEVANT ALGORITHMS

- **Mario's on platform movement:** Programming Mario to walk on slanted platforms has definitely been the most difficult (and later on problematic) challenge that we have faced during the assignment. This algorithm has 2 variants: one for right-slanted platforms and the other for left-slanted platforms, but they work similarly.
  First of all it checks Mario's direction (*mario.direct*). After that, everytime the pixel corresponding to Mario's feet overlaps the upper left corner (or upper right corner if the platforms are slanted to the left) of a platform, Mario is raised 1 pixel upwards or downwards, depending on Mario's direction.

- **Floating Mario bug fix:** This bug was a result of how we programmed *onPlat* detection. Sometimes Mario would stand "floating" over a platform after jumping. The fix consists on checking whether there is a platform 2 or 3 pixels under Mario's feet, and if that was the case Mario's position would then be lowered accordingly.

- **Mario's death animation:** In order to properly animate Mario's death after colliding with a barrel, we created the attribute *condition* inside class Mario, and set it to False. Then, we checked if the (pyxel.frame_count // 6) % 7 was equal to zero, or if the *condition* was true, so as to start the animation when the frame count operation got to zero (because again, the *condition* attribute is set to false). Once the frame count condition was zero, we would set the mario.condition to True until the animation was completed. The animation itself would consist of several mario sprites which would synchronize with (pyxel.frame_count // 6) % 7.

- **Mario's jumping mechanics**: To code this, we created a *jump()* method, a *jumping* boolean attribute, initially set to False, and a gravity attribute initially set to zero. If mario is on a platform and the player presses the space key to jump, then what we did to make the jump "smooth" was simply lift mario off the platform, and then change the *gravity* attribute to a negative value (-4) so that mario would no longer be on a platform and thus the *fall()* method could take care of the rest. The fall method updates the Y component of mario to the value of our *gravity* attribute, and this *gravity* value decreases every frame. This process is repeated until our *gravity* is zero. At this point, we consider that mario is now at the maximum height of its jump, and now all we do is increase mario's Y position until it falls back on a platform.

- **Detecting when mario was *inRange* to increase score:** To control if mario is jumping a barrel, we created a *scoreable* attribute for the Barrel class, initially set to True, as well as an inRange boolean attribute set as True if mario was on top of a barrel. Mario is considered to be *inRange* if mario's X position is close to the barrel's X position, and if

mario's Y position is a bit higher than the barrel's. When these two conditions are fulfilled, the score is increased by 100, and afterwards, that barrel is no longer scorable, so as to not score multiple times in a single jump over the same barrel. This, however, makes it so that you can only "score" every barrel once. But, to be honest, this is something which is not at all intended to be done (Since the purpose of the game is to get to the top, you will likely only jump once over each barrel). Additionally, we took advantage of the *inRange* to add a text saying +100 next to mario which shows when mario is *inRange* until mario is once again on a platform.

- **Donkey Kong's barrel throw:** As previously shown in the class diagram, Donkey Kong has a method called *throwBarrel()*, which returns a barrel object and an ID for it to be stored in a dictionary. In order for Donkey Kong to randomly throw barrels, a random number between 1 and 3 is created. After that, if pyxel.frame_count is a multiple of 30 times that number (30, 60 or 90), then he will throw a barrel as long as the length of the dictionary where the barrels are stored is smaller than 10. This makes it so that Donkey Kong will throw a barrel either after one, two or three seconds.

- **Animating Donkey Kong:** This was probably the toughest thing for us to do, in terms of animation, since we had to play around with the frame count once again, and sync the animation with the Donkey object we called *kong*. To better understand the different states kong could be in, we created some attributes for the Donkey class. They're all booleans, and are pretty straightforward: *throwing*, *grabbing_barrel*, and *waiting*. We have one sprite corresponding to when each of these attributes are set to true, plus another sprite which is displayed when none of the attributes are true. To synchronize when the sprites were shown on the screen, we first initialled all of the mentioned attributes as False and then selectively set each one to True. Firstly, we would set the *throwing* attribute to True where we select to randomly choose *Donkey Kong's barrel throw* (explained in the previous point). Here, we also set *waiting* and *grabbing_barrel* to False. Then, in the *draw()* function from pyxel we set that if *throwing* was True, then it would draw the corresponding sprite with Donkey Kong throwing a barrel. Then, after some frames (using pyxel.frame_count), it would set *throwing* to False and *grabbing* to True. We did this procedure equivalently for the other previously mentioned kong attributes.

- **Barrels randomly rolling down ladders:** Since we knew from the start that barrels had a 25% chance of rolling down a ladder, the solution to this was pretty straightforward. First of all we check whether a given barrel's position is right over a ladder making use of for loops. When that event occurs, a random number between 1 and 4 is generated, and if that number happens to be equal to 3 then the barrel will switch from *platMove* (move sideways) to *ladderMove* (move downwards).

3. PERFORMED WORK

We didn't exactly follow the sprints model we were given on the assignment. Instead, we created each class, drew the sprites and animated them almost at the same time. It was quite hard for us to imagine which attributes would be necessary without visualizing the object in the actual game.

To store several objects of the same class we opted for dictionaries because it was easier to identify each individual object within the dictionary (the keys in our dictionaries work as IDs).

We began by making Mario and working on its attributes and methods, extending them along the way whenever it was required. After Mario came the platforms, barrels and ladders. We found out that the barrels and Mario had more in common than we thought, so we made the methods for barrels based on those we already had for Mario. Ladders gave us a bit of trouble, since we had to be very precise when measuring distances. Lastly, we created Donkey Kong.

There is an extra class named "InvisiPlat", which has the same attributes and role as a platform, but it's not drawn by the pyxel.blt function. The reason for it was to fix a bug which made Mario to walk slower while on the platform at the edge of each floor.

With that, the basic game was complete (Sprites 1 to 4). We found ourselves with a little extra time, and so we decided to add a few extra features like:

- Highscores table.
- Music (8bit cover of "Spider Dance" by Toby Fox).
- '+100' text popping up everytime Mario jumps over a barrel, and '-200' whenever Mario dies.
- Lives counter (with little Mario drawings).
- Animated Christmas Tree.

4. CONCLUSION

To put it in a nutshell, in this project we managed to learn a lot. Not only did we learn all about the pyxel library, but we also gained a much better understanding of how Object-Oriented Programming works. At first, we made a few mistakes, like mashing up the different classes into the same file, but after modifying our code and splitting all our classes into different files, we realized just how important it is to do this, as it really made the coding experience overall better, and it made reviewing the code for last-minute changes a whole lot easier.

Furthermore, thanks to this project we learned a lot about how to work in groups and code in parallel. It was very helpful to learn from our partner's ideas and learning their thought process. Sometimes one of us could be struggling to find a solution to a problem or bug in the code, and having a partner really helped in mitigating them. This project is probably the first piece of work we've had to do in which we had to work around an idea and present a final product. Doing this project was very useful, since it is probably something we'll have to do in the future very often.

Lastly, this project has made us improve in the way we code as a whole. We found small changes that can often result in a cleaner, less repetitive code. However, we consider that there are certain things we could have improved in our code, but didn't look into them due to lack of time. Still, we noted them so we can keep them in mind next time. One of those ideas was to use GitHub to better control our code and greatly speed up the way we worked on the project. But again, even though we would have liked to work from GitHub, we didn't have the knowledge nor the time to learn all about this platform.

5. PERSONAL REMARKS

Overall, we both loved the project. Recreating a classic game from the 80s was actually brilliant. Not only was it fun to make, but it was also a great way to learn to face and correct bugs for the first time. However, there are a couple of things we would like to see improved for future projects similar to this. We think it would have been a good idea to know that it was not mandatory to make the platforms slanted a bit sooner, because by the time we were told about it we had already finished working on the platforms. Also, we would have appreciated to have had some links provided to us with some information on the pyxel library, because at times we did not know where to look for solutions to our problems. Additionally, as we mentioned before, we think that using some online version control tools like GitHub or GitLab should be encouraged, as they are really useful to improve productivity. Other than these few things, we must say we liked everything about the project.