**Final Project: Donkey Kong**

**Programming**

**2019 – 2020**

Sofía Pérez Pérez & Keyla Muñoz Guilcapi

Group 97

Data Science and Engineering

Table of contents

[Introduction 3](#_Toc26714171)

[1. Classes design, relevant fields and methods 4](#_Toc26714172)

[2. General description of the most relevant algorithms used 6](#_Toc26714173)

[3. Description of the work performed, functionality included, parts not performed, and/or extra functionalities provided. 7](#_Toc26714174)

[4. Conclusions 9](#_Toc26714175)

[4.1 Final summary of the work performed. 9](#_Toc26714176)

[4.2 Main problems found when implementing the game 9](#_Toc26714177)

[4.3 Personal comments 10](#_Toc26714178)

# Introduction

The aim of this project was to reproduce the Donkey Kong game by using Pyxel which is a retro game engine for Python. In this document we will report the work that we did in order to achieve our goal.

To get through this project we followed an incremental path which helped us to get everything well organized and increase the game functionalities step by step.

Moreover, we took into important account the syllabus learned in this course to perform the code of our game as well as the directives and recommendations of our teachers.

# Classes design, relevant fields and methods

Our game is formed by 7 classes, we created a class for each sprint because even though they are similar they do not contain the same functionalities, therefore we considered more appropriated to create one class per each sprite. Moreover, since we used private attributes in each class we needed to use the special method property, so we are able to use the attributes in the main program.

For this part we put in practice several object oriented techniques principally, and other syllabus learned in class such as functions, loops, etc.

**Barrel:**

We created this class to represent each of the barrels. It contains in a list, the images of the barrels, which change every 7 frames, to make them look not static. Also, it has the method which is in charge of the movement of our barrels.

**Donkey Kong:**

This class is used to represent the Donkey that will be throwing barrels. So, it contains the coordinates x and y to represent its location, and similar to the barrels, it contains a list including the images to make the Donkey look dynamic.

**Stairs:**

This class is used to represent each of the stairs. It is a simple class in which we programmed the x and y coordinates to then be used in the main program, as we will create several stairs it makes the work easier because we will only have to change the value of x and y.

**Mario:**

This class represents the character “Mario” which is the player controlled by the user. This is our most complex sprint because it is the one that has to “play the game”. Therefore, it contains some methods and attributes such as, the position of Mario, with the x and y coordinates, jumping and desjumping to show if Mario is jumping or if it is not, lists containing the appropriate images to make Mario dynamic to simulate movement, the code for the number of lifes, which starts with 3 lifes, and the total points that Mario gains every time he jumps a barrel.

**Platforms:**

This class is used to represent each of the platforms. The attributes that were needed were the x and y coordinates, to represent the location of the objects.

**Princess:**

This class is used to represent the princess, and it contains the attributes of the position x and y, to represent the position of the princess. Since the Princess is static, we did not need to create a method to move her.

**Game:**

This class is the most important one of our game, since it is going to be implemented in one class including all the methods and needed functionalities in order to get a well functional game. Here we set the attributes, that correspond to the characters of the game: the princess, Mario, Donkey Kong, and also de platforms, stairs and decoration.

Moreover, in this class we programmed the conditions to make Mario and the barrels move in our game, such conditions include, moving only if the exist a platform, falling if there is not such platform, going up or down (for Mario) if there are stairs, adding a 25% chance to the barrels to go down a stair, and also the conditions to lose a life or to gain points.

**Most relevant fields and methods**

Our most relevant methods have to do with the movement of our sprites. They are similar for each sprite, however, Mario has the complex one because in his ***move*** method we did not only manage to make him move right or left but also up and down.

Another important method for us are the ones in which we check if there is a platform or not, for both Mario and the barrels: ***existPlatformForMario*** and ***existPlatformForBarrel*** even though they work in the same concept, we did it separate because the coordinates needed for each one (Mario and Barrels) were different.

Now, according to the relevant fields, we consider very important the movement of our barrels because we had to manage to make them move automatically. To make sure they would not stop we implemented their movement when we press any bottom to make Mario move and also when Mario is not moving, and depending on which platform the barrel is, it would move right or left.

# General description of the most relevant algorithms used

Since the goal of the game is to reach the princess and save her, the **movement** of the objects was essential. In the classes of the main objects such as Mario, and the barrels, we needed to create the method “move”.

For example, every time we press de right, left, up or down bottom, Mario is going to move in that direction, as long as he is well positioned. Also related to the movement, we created in the class game a function called “existPlatformForMario” that checks if Mario is on a platform, and if he is not, he falls until the next one.

In the case of the barrels, creating movement has been a little more complicated since it was autonomous. First, we needed to create in their class a method move, that depending on which platform they are on, they would move to the right or to the left. We also had to implement this algorithm in all the bottoms we used, because if this was not done, whenever we moved Mario, the barrels would stop moving. As we did with Mario, we also created a method in the class game called “existPlatformForBarrel” that checks if each barrel is on a platform, and if it is not, they fall until the next one. Apart from that, each of the barrels, if they happened to be in the position of a stair, they would have 25% chances of going down on it. To do so, we created also in the class game a method called “existStairForBarrel”, that checks if the stair and the barrels have the same x and y coordinates, taking into account a range.

Another important aspect of the game is to avoid the barrels thrown by DonkeyKong by **jumping**. To do so, we had to create a method in the class Mario that “jumps” and “desjumps”. We thought it would be convenient to divide the action in two parts, the first one when Mario goes up, by modifying the coordinates x and y of him, and the second one, when Mario goes down, in which we sum the value that had previously been subtracted from his y coordinate.

An interesting aspect that we considered important when implementing the game, was the **high score** and the **remain lifes.** For the first one, we took advantage of the algorithms “jump” and “desjump”, to check whenever Mario jumped over a barrel. If this happened, we would increase the total points by 100. Furthermore, as most of the games, Mario starts the game with 3 lifes, and when he loses (by touching a barrel) he goes back to the beginning, but with one life less. We did this just by checking if the position of Mario was close to the location of each of the barrels.

Finally, to make the main characters, such as Mario, Donkey Kong and the barrels **dynamic**, in the class of each of them, we created a list of lists with the coordinates of the images we wanted to change of the pyxeleditor, and by changing the index of each of the lists, each time they move, or every frame, with the method “getimage”, it is shown on the screen a different image of the said characters, simulating movement.

# Description of the work performed, functionality included, parts not performed, and/or extra functionalities provided.

The main objective of the game is to rescue the princess, captured by Donkeykong. Mario does it by going left or right on the platforms and going up and down on the stairs. At the same time, the Donkey throws barrels in order to kill Mario and avoid him to win. If Mario touches a barrel, he loses a life and goes back to the beginning position. When he runs out of lifes he loses and the game finishes. We were asked to understand the logic of the game, and think of a good way to implement it, using Object-Oriented Programming.

We are happy to say that we managed to complete that task, with a lot of problems and questions that needed to be solved. But finally, the game logic was implemented, not only in the game class, but also in the rest of the classes.

The structure of the code is very simple. We created a class per character or type of object, and also a class game when the whole game was developed. We stored each of them in one file, and all of them in the same folder.

We consider it would also be important to create a new file called “constantes” in which we store important values, such as the initial position of Mario, of the princess, etc.

In addition to the main objective of the project that was to simulate the game, we added some decorations, such as false stairs, that have been created apart from the real ones, in another list called “listdeco”. We did that so Mario or the barrels could not use them to go up or down, and they could use just the real ones. It was also added a broken heart that becomes a full heart when Mario reaches the princess, simulating the end of the game, and a blue barrel that contains fire. All of these structures were not indispensable, but we decided that a good game did not just consist on the behave of all the objects and their movements, but on the whole atmosphere and taking into consideration every detail. It is because of that, that we thought of putting some music while the game is being performed, just to make it more appetizing to play. We selected a relaxed song, to make it easier to the player to concentrate, and beat DonkeyKong.

The logic of the game is basically developed in the main program, in the class Game. First, the constructor of that class is the responsible of creating the objects, that are going to belong to the other classes previously created. Apart from the constructor, in that class there are the two main methods, update, that controls the movement and the exact position of all the objects, taking into consideration in which direction they have moved, and draw, that controls what appears on the screen. Both methods are run every frame, to keep up with the game every moment.

Inside of update we include all the methods and objects from the rest of the classes, and every possible situation is being taken care of, for example, if Mario´s coordinates are close to DonkeyKong´s ones, Mario loses a life and goes back to the beginning.

Draw´s method shows on the screen all the objects that are supposed to appear in a concrete frame. Not all the objects and images appear every frame, for example, every five frames, Mario is going to change his image, so the method draw is going to show in the screen a different one. We have decided that the images we were going to select belong to the folder called assets, that contains the images of the different characters of the game.

# Conclusions

## 4.1 Final summary of the work performed.

To conclude, in order to perform a good emulation of the Donkey Kong game, we put into practice the syllabus learned in class and the advice provided by our teachers.

We created several classes, each of them with the corresponding attributes and methods (game logic) to then add everything to our main class called Game() as objects. We started by creating the graphical interface, which contains our sprints (Mario, Donkey Kong, Princess, stairs and platforms). Also we added the score text and the remaining lifes using mini – Marios.

After that, we started implementing all the programming to set conditions in order to move our player (Mario), gain points, loose life, loose the game, make the barrels go down, etc. As we wanted to go a step forward, we implemented dynamic movements to our barrels, Mario and Donkey Kong, as well as a background sound while playing the game

## 4.2 Main problems found when implementing the game

Through all our work we have come across several difficulties, which were whether easy or difficult to solve for us. Nonetheless, we gave our best to solved them. First, when making Mario move we had to deal with making him move right or left only when he is on a platform, to solve that we stablished some conditions using a for loop and if, to compare the coordinates of Mario and the platform. Also, our main struggle was to make Mario and the barrels fall at the end of platforms, to solve this we had the help of our teacher and finally by including some constant values we created a function to check if under Mario the is a platform, and the same concept we used for barrels

## 4.3 Personal comments

In conclusion, we consider that we achieved our main purpose which was to improve our programming skills and learn through practice. Moreover, we are happy with the final result of our game because it has taken a lot of effort and work, which we see reflected on our final game.

In order to get some information about Pyxel we found good information in GitHub, however there were not many information on internet or tutorials as we would have wished. Nonetheless, the help of our teachers was very helpful and motivational to keep working hard on our project.

Finally, we believe this work has been very useful to put into practice what we learned in this course, as now we feel more confident to work in projects similar to this one.