DONKEY KONG PROJECT

Almudena Bravo Cerrada and Inés Martínez Merino

Data Science Degree

SUMMARY

This project consists in the designing of the Donkey Kong video game by implementing Object Oriented Programming. Our goal was to emulate the original game and achieve a good classes' design in order to do so.

The game features Mario's basic movements (going left, right, climbing and descending ladders and jumping over barrels) and score, and the barrels' movement over the platforms as well as their descending by ladders with a chance of 25%.

TABLE OF CONTENTS

1. CLASSES AND IMPORTANT METHODS

This project is based on a well-defined classes' design. In order to do so, we created six different classes: **player, barras**, **barril**, **kong**, **princess** and **stairs**. Also, these classes needed to have private attributes, so for each one we defined the properties necessary for us to be able to use them.

STAIRS, PRINCESS, KONG, BARRAS

The **stairs**, the **princess**, the **kong** and the **barras** classes only manage as attributes the X and Y position given when creating an object. We did not give them any more attributes or methods as they are static elements.

BARRIL

Within our **barril** class, we assigned 5 different attributes: the barrels' X position, Y position, direction and the image we use for the barrels to not be static. For the barrel's direction and image, we established a setter as we changed its values in the main program. The X and Y position don't need a setter as the are changed within the class (by using the move method). There is another method called **gira** that receives the direction of the barrel and it enables the barrels to appear as if they are rolling around. The attribute left to be described ("a" has an initial value of 0) are used exclusively within this method. There are 4 possible images the barrels can take, to which we gave the values from 0 to 3 for them to be drawn in the game. Depending if the direction entered in the method **gira** is left or right, the image will start by being 0 or 3. Now, the value of "a" will always start with 0 but we will be changing it by adding 1. Therefore, the value of "a" will determine the image the barrel will take. Once the value of "a" reaches 20, it will go back to being 0. If the direction entered is down ("do"), the barrel will one be taking the images 0 and 1 but following the same structure previously described.

PLAYER

This class is the one in charge of our Mario. It has several attributes: Mario's X postion, Y position, remaining lives, image, direction, score and two boolean attributes that define the adding of 100 points over Mario's head when jumping over a barrel and his death. There are also 3 attributes that are managed within the class (**a**, **b** and **salta**).

The class **player** manages 8 different methods:

- **NEWMOVE**
  This method receives 3 different values: **d** (the direction) and 2 lists named as **bars** and **stairs**. We created a for loop in order to check which in which platform we are according to Mario's Y position. Once inside it, if **d** is "left" we will call the method **move** and subtract 2 units of Mario's X position; we will also call the method **run** for Mario to look as if he were running and, finally, the attribute **direction** will be established as "left". If **d** is "right" we will do the same thing except for Mario's X position (we will be adding to units) and the **direction** will be "right".
  If **d** if "jump", a for loop checks in which platform Mario is and it will call the method **run.** If **d** is "up", a for loop checks if Mario's X position is near a ladder and if Mario's Y

position in the range of the ladder's height. If so, we will call the **move** method, subtract one unit of Mario's Y position and the **direction** will be established as "up". In case **d** is "do", we'll be doing the same thing except that we will be adding one unit to Mario's Y position and the **direction** will be established as "do".

- **MOVE**
  This method receives two parameters which enable Mario to move, that is, it changes the values of its X and Y positions.

- **JUMP** and **JUMPNO**
  When these methods are called, depending on the value of Mario's **direction** at the time, Mario's X position will be added or subtracted a value. The function **JUMP** will enable the first part of the player's jump: the Y position will decrease appearing as if Mario is at the top of his jump. Also, the class' boolean attribute (**salta**) will now be set as TRUE. Then the other function is called, it will perform the exact opposite: the Y position will increase and the Boolean attribute will be set as FALSE.

- **RUN**
  This method is used to change Mario's sprite as to appear he is running. It works in the same way the method **gira** does in the class **BARRIL.**

- **PERDERVIDAS**
  This method when called will decrease by one the value of the attribute **lives** as well as establish the **score** to be zero.

- **RESTART_LIVES**
  This method will establish the **lives** to be 3.

- **SUMPOINTS**
  It receives one parameter. If it is "remove" then the score will increase by 50 points, whereas if it receives a different value the score will increase by 100 points and the attribute **sco** will be set as TRUE.

GAME

This is our main class, where all the game follows through. It describes different types of attributes, such as objects (Mario, the princess, Donkey Kong), lists where barrels, stairs and platforms are stored and both numeric and boolean attributes.

Thanks to the function *pixel.run*, this class will be run every second hence simulating a big loop enabling the functionality of our game.

This class' methods are the following:

- **UPDATE_MARIO**
  Within this method, we control Mario's basic movement depending on which key is pressed. Also, it manages Mario's descent through the gaps between the platforms and

the edge of the screen, the ending of the game depending on Mario's position and Mario's limits within the game.

- **UPDATE_BARRELS**
  Within this method, new barrels are created as objects as long as there are no more than 10 rolling around and added to the list attribute that stores these barrels. There is a for loop in order to check every barrel within the list.
  This method controls the basic movement of the barrels, their descent through the gaps and the stairs, which in this last case can only be done with a 25% of chance, Mario's crash with them and, finally, their removal of the list and therefore of the game if they reach Mario's initial position.

- **RESTART**
  In case of calling this method, the game will take its original form, that is, Mario will return to its initial position and there will not be any barrels.

- **UPDATE**
  This method oversees the running of the game. Within it we call the **update_Mario** and **update_Barrels** methods which will implement both Mario and the barrels in our game.

- **DRAW_DEATH**
  This method will draw a screen with the score achieved during the game and will ask the player if he wants to play another game or quit it all together.

- **DRAW**
  In case Mario's boolean attribute **death** is FALSE, the main game will be drawn: the princess, the donkey, Mario, the barrels, the ladders, the platforms, the scores, Mario's remaining lives…
  Also, while the boolean attribute **help** remain TRUE, it will be drawn. However, once Mario reaches the princess the attribute will be FALSE and the attribute **Corazon** will now be TRUE, hence replacing the "help" sprite for a heart.
  However, if Mario's boolean attribute **death** is TRUE, the **draw_death** method will be called.

2. MOST RELEVANT ALGORITHMS

In this section we will describe the most relevant and appropriate algorithms used within our game.

- ✓ **Mario's jump:**
  In order to achieve Mario's jump, we initiated a numeric variable: **counter = 0.**
  Once the key space is pressed, the program will examine if Mario's attribute **salta** is FALSE to check if Mario is not jumping at that moment. If so, the counter will now by established as the **pixel.frame_count** (we check the time Mario's jump takes) Mario's method **newmove** will be called with the parameters "jump" and the list that contains the platforms' X and Y positions. By doing so, Mario's X position will increase or decrease depending on its current direction by 15 units, Mario's Y position will decrease by 15

units (he will be in the air) and now **salta** is TRUE. This latter change and the passing of time controlled by **counter** will end the jump: Mario's X position will increase or decrease another 15 units, its Y position will increase 15 units (he will be on the floor), **salta** will now be FALSE and **counter** zero.

- ✓ **Mario's descent through the gaps:**
  This is achieved by controlling Mario's X and Y positions. While the X position coincides with the gap and the Y position is less than the platform's bellow Mario, then its Y position will increase until the last condition is no longer met.

- ✓ **Score:**
  Mario's score will increase by 100 every time he jumps over a barrel and by 50 every time a barrel disappears. Therefore, once the space key is pressed, we will check if Mario is jumping over a barrel by analysing if the barrel's Y position is near him and if its X position is 15 units on the left or on the right of Mario (the same distance he takes when jumping). In case this is true, we will be calling Mario's method **sumPoints** with the parameter "jump" as we want 100 points to be added.
  Also, each time a barrel is removed, we will be calling this same method but with the parameter "remove" to add 50 points to the score.

- ✓ **Barrels descending using the ladders:**
  For this algorithm we used the barrel's attribute **num**. We introduced a for loop in order to compare the X and Y positions of every stair with the ones of the barrel. In case this was all true, then if **num** is different from 0 (which it is as we established it to be 5) it would check if a random number from 0 to 3 was 0. This allows the barrel to descend with a probability of 25%. In case the number is 0, now **num** for this specific barrel will be 0 and while its Y condition is less than the stair's it will de going down. Finally, **num** will be again established to a number different from zero in order for this barrel to be able to descend by any other ladder.

- ✓ **Mario crashes with the barrels:**
  This is based on knowing both Mario's and the barrel's X and Y positions. If both coincide, we call both Mario's method **perderVidas** to reduce by 1 Mario's remaining lives and **restart.** In case Mario's lives were 0, then Mario's attribute **death** would now be TRUE.

- ✓ **End of the game:**
  This is executed once Mario reaches the princess. An attribute is now **pyxel.frame_count.** This will check the minimum time established before **death** is TRUE and therefore the program will draw the score screen. If the Q key is pressed the game will finish and if the R key is pressed the game will be reset and we will start over. Also, another variable will be set as **pixel.frame_count** in order to check the minimum time established before the score screen is shut down.

3. PERFORMED WORK

In this section we will try to explain all the work we did and why. We try to create, within our capabilities, the most similar Mario game.

- **PROCESS**: first we started creating all the floors, using a *loop* and organising them in the way we liked it, but instead of inclined (that the only thing that we had to change is that every 2, the y incremented 0.5) we drew them straight because we wanted to do right all the functionalities that were necessary. After draw all the floors and stairs, we start to add things, such as Mario and its moves, the fire, the princess, the donkey and at the end the barrels.
- **FUNCIONALITY**: like we have explained in the previous sections, we created plenty of algorithms to simulate all the movements. The most relevant ones were the barrels', because we created an algorithm so they could be able to move by themselves, go down the gaps, and calculate a random number to see if they could go down the stairs; the other one was Mario's movement, such as the jump, the fall through the gaps, and the possibility of going up and down only were there was a stair. Also, he can climb little bit using the broken stairs, although not all the way through.
- **NOT PERFORMED**: we have tried to approach the original game, but within our capabilities, we couldn't achieve to create all the things we wanted to. The music, the hammer, the floor inclined (because that meant to change most of our code) and the correct movement of our Donkey.
- **EXTRA**: however, we instead included some extra elements, such as the correct animation with Mario and the barrels, that simulate the run and the roll through the screen. Also, we decide to create an end-game screen, where we decided to put a message "GAME OVER" and two different possibilities "RESTART" (that you can only press at this screen) and "QUIT", and also the possibility of seeing the score obtained once the game is over.

4. CONCLUSIONS

We tried to simulate the design of the Donkey Kong video game by implementing Object Oriented Programming, and therefore we achieve our goal, emulating the original game using a good classes' design in order to do so. Moreover, we included some extras as we have seen in the last section but also, we implemented the game's basic features: Mario's basic movements (going left, right, climbing and descending ladders and jumping over barrels) and score, and the barrels' movement over the platforms as well as their descending by ladders with a chance of 25%, but at the end, we could not emulate the entire game, because we couldn't make some of the methods, like the music, the harmer…

Most of the not performed methods that we didn't implement were due to a lack of time; there were so many things to do, things that we had to change because it wasn't the correct way to do so, and at the end, there was no more time to include any more extra items. As well, we found some more complications during our work, it was difficult to search for information in

internet because of the lack of it, so most of the algorithms and ways of doing the game were just more difficult to do.

Other than that, the project was very interesting, we learned a lot of things, and we enjoyed building our own game. It was a fun way to use all that we had learned during the course and therefore a good way to review it. However, during the game's development, we realised that this project was not very related with our degree (data science). It's fun to create a game, but we would have liked to do something that would be useful in the future regarding our career of choice.