# Final Project (Python)

# Program Design & Methods

**Student Name:** Christopher Samuel Tendi

**Student ID:** 2440065030

**Project Name:** Pelatformer (Pygame)

**Class Code:** L1BC

**Lecturer:** Ida Bagus Kerthyayana

## Project Specification

After much debate and discussion with myself on what to program for the final project, I decided to go for a 2D platformer using Pygame, due to the nostalgic childhood memories I had playing Super Mario in my very first gaming device, Gameboy Advance, as well as the notorious side- scrolling game, Bounce Tales, which was available in all Nokia devices since 2008. Both these games were extremely addicting to play, but at the same time the feeling of genuine frustration was constant throughout the experiences I had with the pair due to the fact that once you failed to complete a particular level, whether or not you like it, you would have to repeat the exact same one, despite already being so close to the end.

Since I am to program a game, thus the main purpose of this project is to entertain. Additionally, it is also made to give a sense of nostalgia for the individuals whom have played the 2 aforementioned games. In terms of gameplay, it is fairly similar to any platformer game. The player will be asked to navigate through different levels with a red ball, which is also the character used in Bounce Tales, with each level having their own unique challenge and terrain. The main objective I had for this game is to program it in such a way that it is playable for most age groups. I decided to do this as I am a firm believer that gaming can be and should be enjoyed by everyone. Knowing that the game is a platformer, thus the inputs are simply the arrow keys, which will translate an output to be the movement of the character.

The decision to use Pygame as my primary module to program this game was mainly because of personal preference. Other options such as Arcade, despite its reputation for being a user-friendly, easy-to-use game development module, gave me a lot of confusion when I first tried utilising it. Moreover, since Pygame has been available for much longer than Arcade, there are way more tutorials and guides that were able help me in programming this game.

# Solution Design

**Design & Planning**

As I brainstormed on how I want the game to be made, I looked into the 3 things that are necessary to be in a game. These include proper gameplay, a main menu and sound effects. Taking into consideration the time I had to program, I decided that only 2 of the 3 would be present in my platformer game, which are the gameplay and the sound effects. As long as the game mechanics functions accordingly, there should be no issues to the game at all. Furthermore, considering the main objective I previously mentioned in programming this game, I decided to make it as simple and straight forward as possible.

In a platformer game, physics is unavoidable. Understanding the physics behind the collisions, movements and the momentum of the ball in relation to the tiles is imperative to achieve fluid gameplay. I had to look up tutorials that could better my understanding of the physics of a platformer game. Considering that I’m limited to the capabilities of Pygame, perfection is not something I should strive for, but instead attempt to achieve a polished gameplay that is playable. There are certainly many issues I faced regarding the mechanics of the player movement which I will explain further later on this report.

A platformer would also require a map or a world. Thus, it was also necessary for me to learn how to make a terrain in Pygame. And just as I figured, there are many ways in which this can be achieved. We can import an external file into our program but I instead decided to go for another alternative, which will be explained later on.

For the sprites, I chose pixel themed sprites. All the tiles and the player character are pixelated. As for the latter, I decided for it to be a 20 by 20 red ball, the same character used in Bounce Tales. The tiles are general 2D grass and dirt blocks of 16 by 16 in dimensions.

**Program Composition**

Below is the list of essentially what my game is made of:

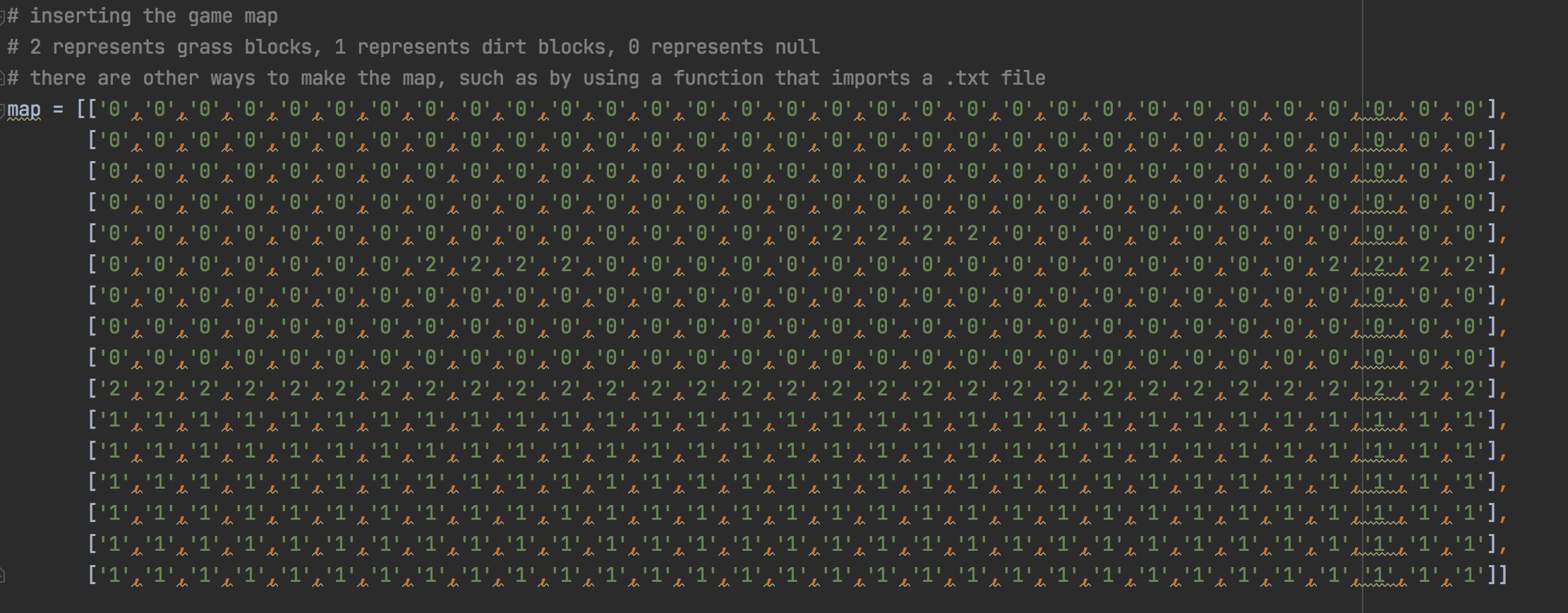
Gameplay: The player will be able to navigate through a map by using the **arrow** keys. Left and Right arrow keys to move horizontally and the Up arrow key to jump.

Sound Effects:The sound effects can be turned on/off using the **Z** and **X** keys**.** When the player presses the **Z** key, the SFX will be muted and can be turned back on when the **X** key is pressed.

## Implementation and Explanation of Code

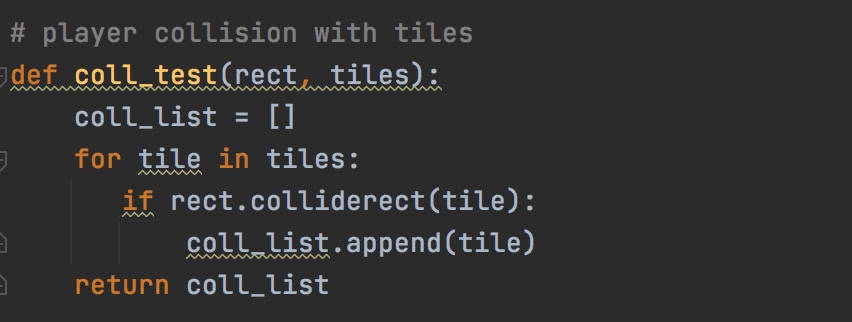
In this final project, all the coding will be done in 1 main.py file. I chose to do this as I very much prefer to see all my codes in a single file rather than in multiple files and then having these files be imported into the main.py file, which may result in unwanted errors. I will only be using 2 python modules, these include pygame and the sys module. The sys module in this program is used to handle file paths in functions when importing sprites and/or images into the program. The project also contains 2 main functions, which are used for the collision and player movement. A game loop that consists of functions for the parallax and scrolling background, player movement, collisions and the game events, which are functions for the keys utilised in the program. The most commonly used function in the program is rect(). Rect() is used for collision detection and to store blit positions of sprites and images .

**Game Map**

****

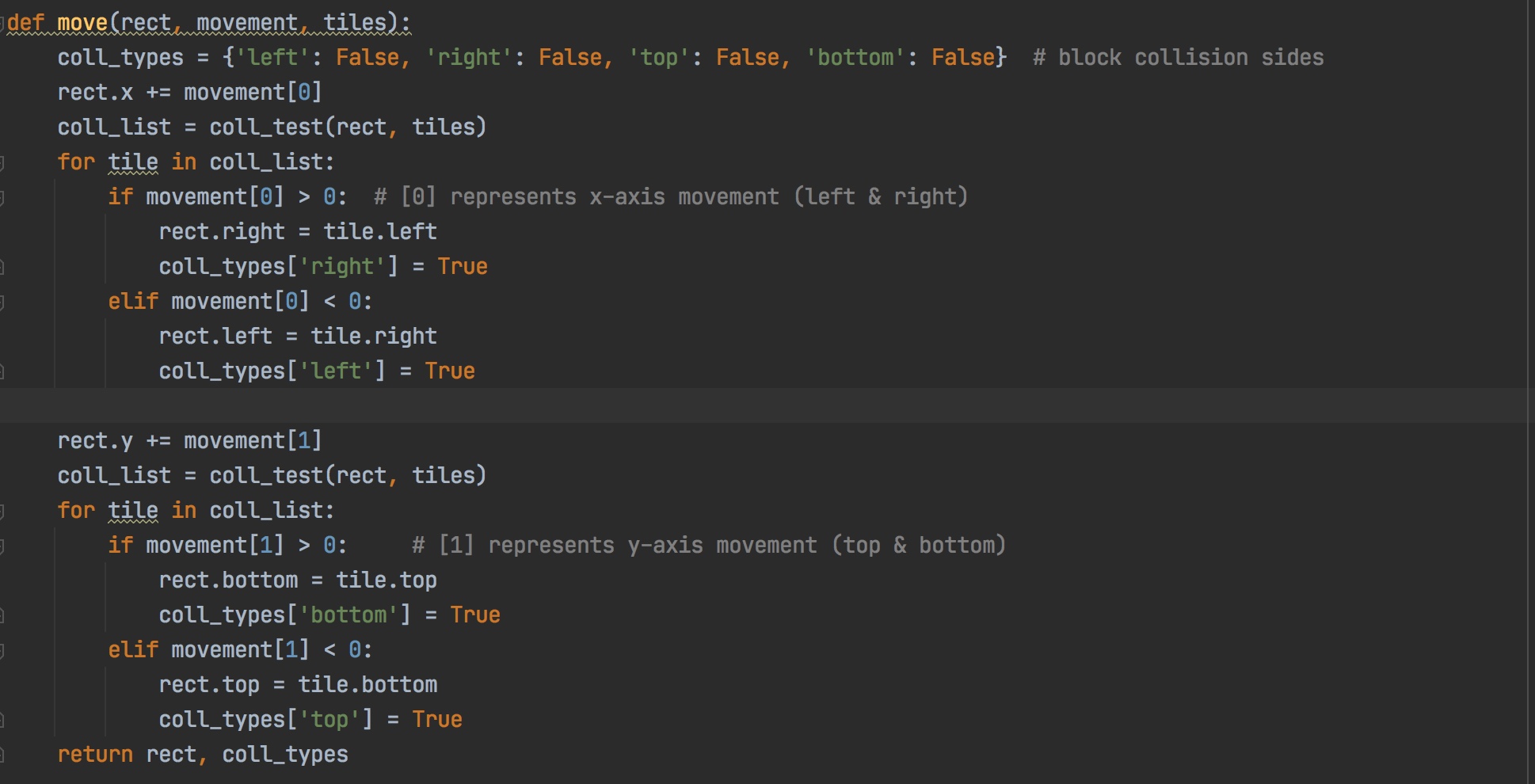
The code above shows the game map. The terrain is coded manually with “,” used as a separator. The number 2 represents the grass block, 1 represents the dirt block and 0 represents null or nothing. I decided to include the map in the main.py file instead of importing it as a .txt or a .rtf is due to the fact that the importing process caused an error in the map seen in the program, I will elaborate on this later on.

**Collision Detection**

****

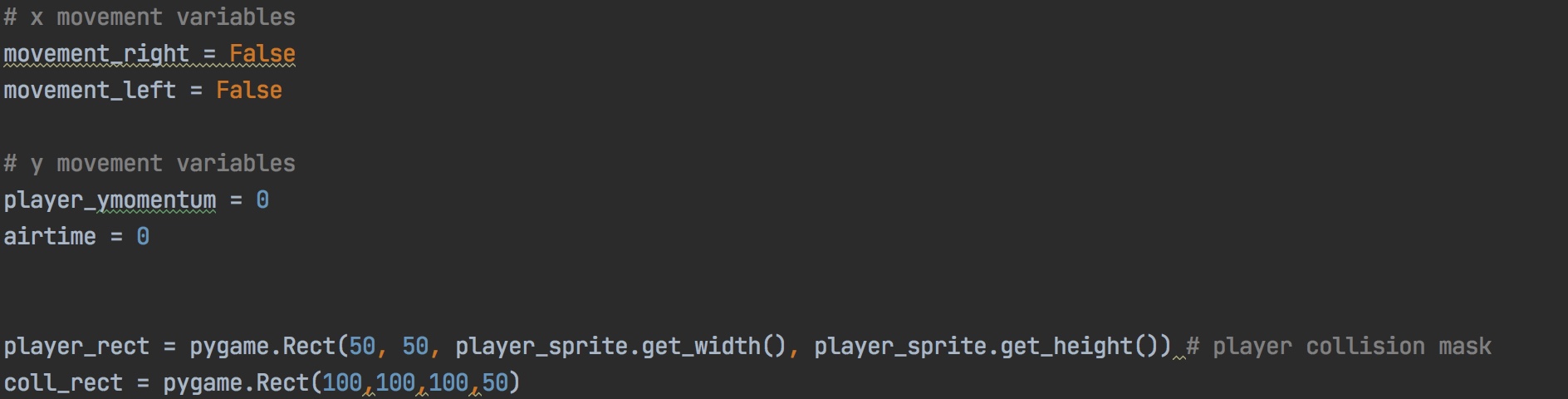
For player collision detection or collision mask, I made a function coll\_test so that the player can collide with the tiles in the map without the latter overlapping with other tiles. The function rect.colliderect() is used to prevent this. The if statement here is used so that when 2 tiles begins to overlap, another tile will be appended, which means that each tile is separate from one another.

**Movement**

****

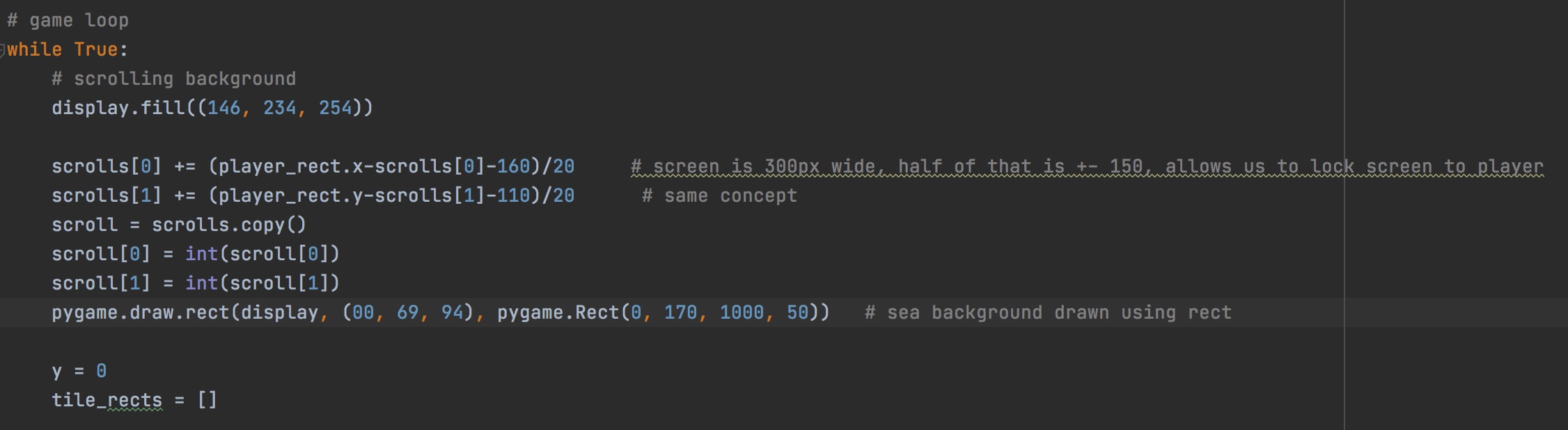
The def move function is used to manipulate player movement along the tiles. Coll\_types represent the collision sides of the player, the sprite essentially has 4 sides, top, bottom =, left and right. So for example, when the ball falls and lands on a block, the bottom collision mask is responsible so that the ball doesn’t just pass through the block. Rect.x is the collision mask along the x-axis of the tiles, movement [0], represents the horizontal movement of the player. If movement [0] > 0 means that if horizontal movement is greater than 0, the collision mask on the ‘right’ side of the block will be active or True. Otherwise, when movement [0] is less than 0, the collision mask on the left side of the block will be True.

Rect.y represents the collision mask along the y-axis of the tiles, movement [1] represents the vertical movement of the player. If movement [1] is greater than 0 thus the ‘bottom’ side collision mask of the tile will be active. Conversely, when movement [1] is less than 0, the ‘top’ side collision mask of the tile will be active or True.

****

The code above shows the x and y movement variables for the ball. Movement\_right and movement\_left represents horizontal player movement whereas player\_ymomentum depicts the jump momentum of the ball or basically the strength of the jump. Player\_rect represents the collision mask around the player. I used .get\_width and .get\_height so that the mask encompasses precisely the entire sprite.

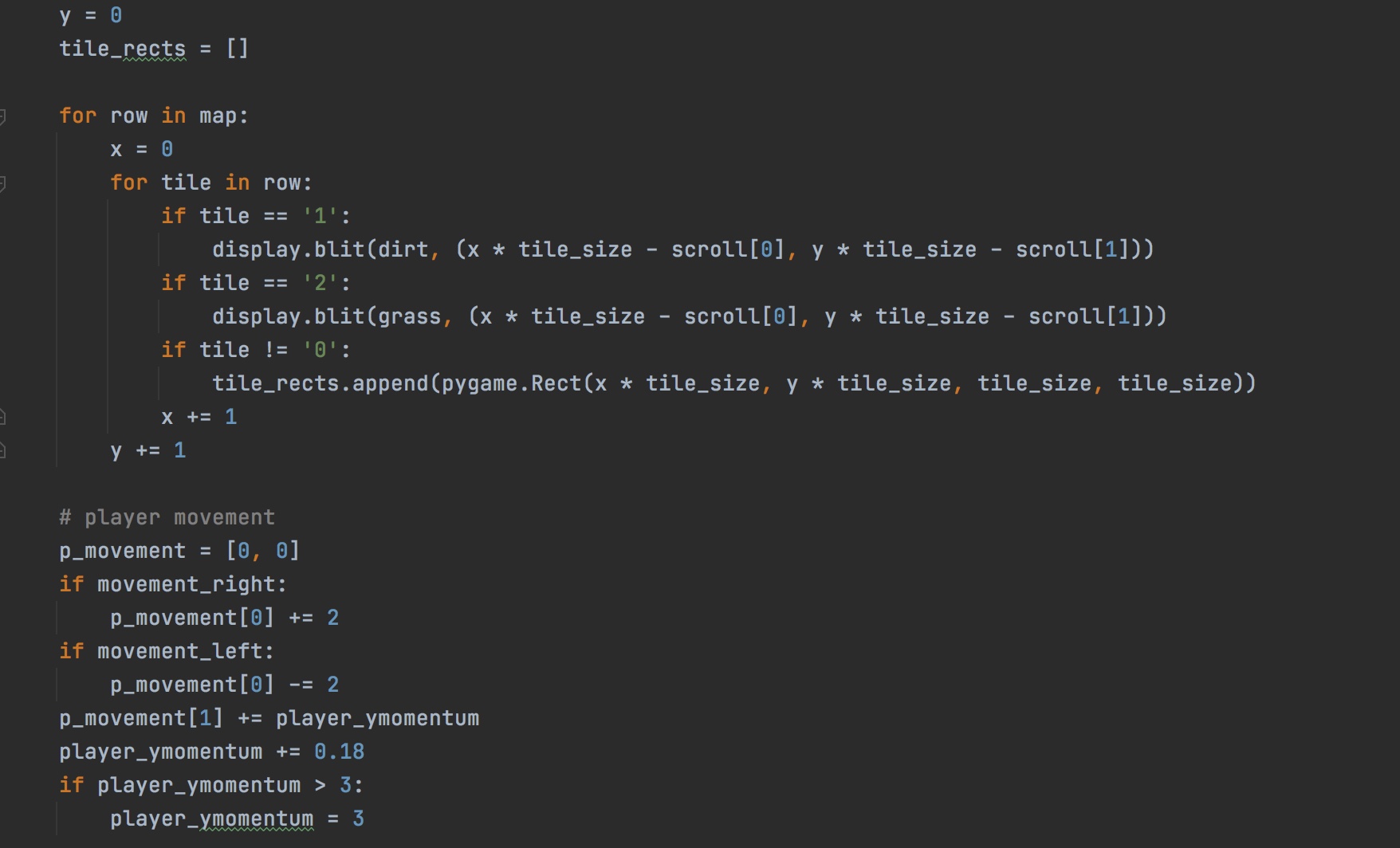
**Game Loop**

****

This is the main game loop, where a while True statement encompasses all the codes within so that if one of the codes do not function properly, then the game will not load. I used this as a pre-emptive for potential mistakes and errors in the functions. First, we shall discuss the scrolling background feature that is implemented in the game. I used display.fill to colour the entire background sky blue. Scrolls [0] and scrolls [1] are the x-axis scrolling and y-axis scrolling functions respectively.

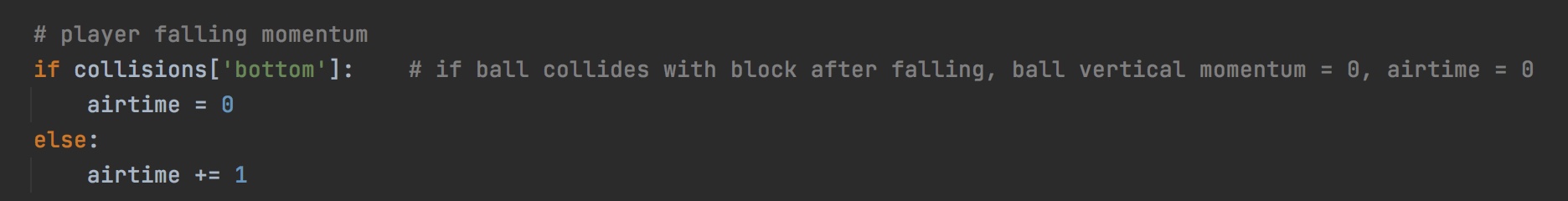
(player\_rect.x-scrolls[0]-160)/20 is used here as we want to position the camera right in the middle of the screen, thus we take 300px (screen width) divided by 2, which gives us 150px and add 10 to that because our sprite is 20px wide. Despite subtracting the scrolls[0] value to the player collision masks, we are actually adding to that value as in this case, subtracting to the value means adding to it. Furthermore, adding 10px afterwards is necessary so that the camera is able to position itself to the centre of the sprite instead of to the left by default. In order to further smoothen the camera movement, we divide the equation by 20. This results in whenever the ball slows down the camera moves quicker and when it moves faster the camera slows down, adding great emphasis towards the camera movement. The same logic is used for player\_rect.y. Since the screen height is 200 px, thus we divide that by 2 which gives us 100 px. We then add 10 as the sprite has dimensions of 20 by 20, then equation will be divided by 20 in the end. The scroll[0] and scroll [1] values are also made into integers to facilitate a smoother camera movement.

pygame.draw.rect(display, (00, 69, 94), pygame.Rect(0, 170, 1000, 50)) is the sea background seen behind the tiles which is part of the parallax effect or layer effect in the program.



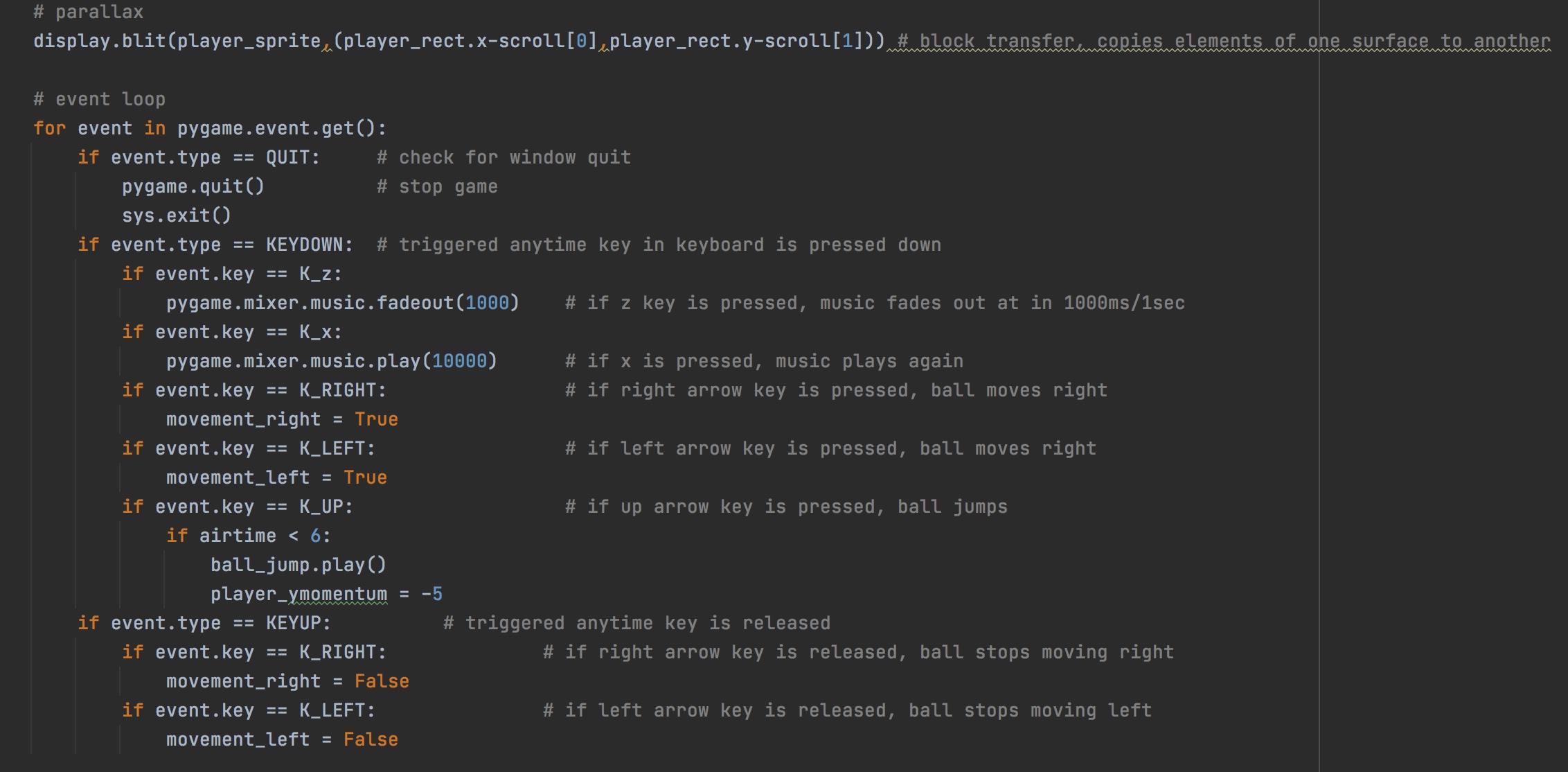
The code above shows the tile display and the player movement values. For row in map statement means that for each row in the map data. display.blit(dirt, (x \* tile\_size - scroll[0], y \* tile\_size - scroll[1])) is used to depict the tiles into the program. Display.blit is used to copy the contents of the map data and translate it into the program as blit stands for block transfer. X is multiplied by the tile size and subtracted by the scroll [0] variable used for the scrolling background effect. I previously mentioned that subtracting values in Pygame means adding to it. So this statement essentially means we are adding the scrolling effect into the tiles. For both x-axis and y-axis I only use 1 variable, which is tile\_size as the dimensions of the tiles are the same (16 by 16), thus multiplying both axis with tile\_size is possible. The same method is used for the y-axis in the tiles. As for tile == 0, since we want it to be empty or without any fill, basically what I did was append rectangles using rect.() with the same dimensions as the blocks, sort of like a puzzle where the null areas of the map is filed with transparent squares that fill up the gaps.

As for player movement values, p\_movement [0] is the horizontal movement and p\_movement is vertical movement. if movement\_right: p\_movement[0] += 2 means that when the player moves to the right, the p\_movement value will be greater than or equal to 2. On the contrary, if movement\_left: p\_movement [0] -= 2 means that when the player moves to the left, the p\_movement value will be less than or equal to 2. player\_ymomentum += 0.18 is the jump ratio or scale of the sprite. Meaning to say, the sprite will have a jump ratio of 0.18, we can alter this number and it will result in the sprite either having less or more force when jumping.



The code above showcases an if statement on the falling momentum of the ball. When the ball falls from a height and lands into a black, the side touching the block is the ‘bottom’ side. Hence if the ‘bottom’ side has collided with a block, thus the airtime or the downward force acting upon the ball will be equal to 0. Else, it will be greater than or equal to 1.

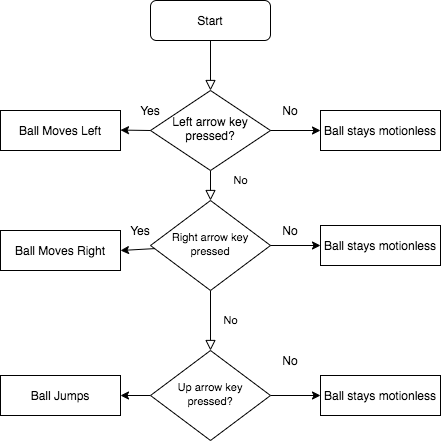
**Player Events**

****

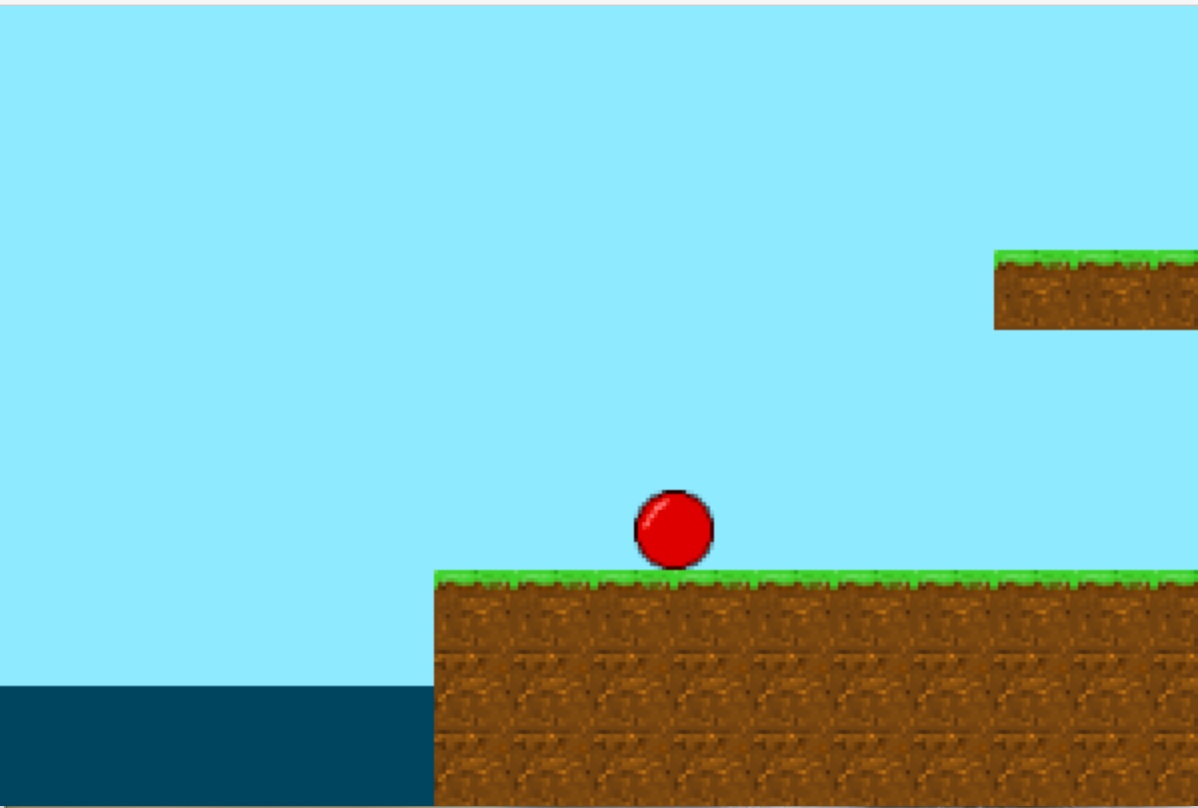
The player event loop covers the keys that are used in the gameplay. The first if statement, QUIT, checks the window if the user has pressed the quit button and if so, the program will be closed. The KEYDOWN statement is used for the keys that are pressed down. With K\_z or the ‘Z’ key used for fading out the music and the ‘X’ key for playing it again. K\_RIGHT, LEFT and UP are used for the horizontal and vertical movements. In the UP key statement, the if airtime < 6 means that if the airtime is greater than 6, the jump sound effect will be played and the player momentum will be subtracted by 5 so that the ball is able to fall back down. Last but not least, the KEYUP statement basically means when the key is no longer pressed, the player will not move left or right.

# Flowchart

**Flowchart for player movement and collisions**

****

## Proof of Working Program



# Reflection & Experience

Overall the experience was undoubtedly tiresome, I faced issues and challenges right from the start. From deciding on what to make up until the programming of the game, I was filled with worry and stress. But I was happy that I did my best

# References

* <https://www.youtube.com/channel/UCYNrBrBOgTfHswcz2DdZQFA> (inspiration)
* <https://freesound.org/people/cabled_mess/sounds/350906/>
* <http://soundimage.org/wp-content/uploads/2016/01/Lost-Jungle_Looping.mp3>
* <http://pixelartmaker.com/art/a7c07a3e29ae4f8>