# 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 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.

**Program Composition**

Below is the list of the program composition:

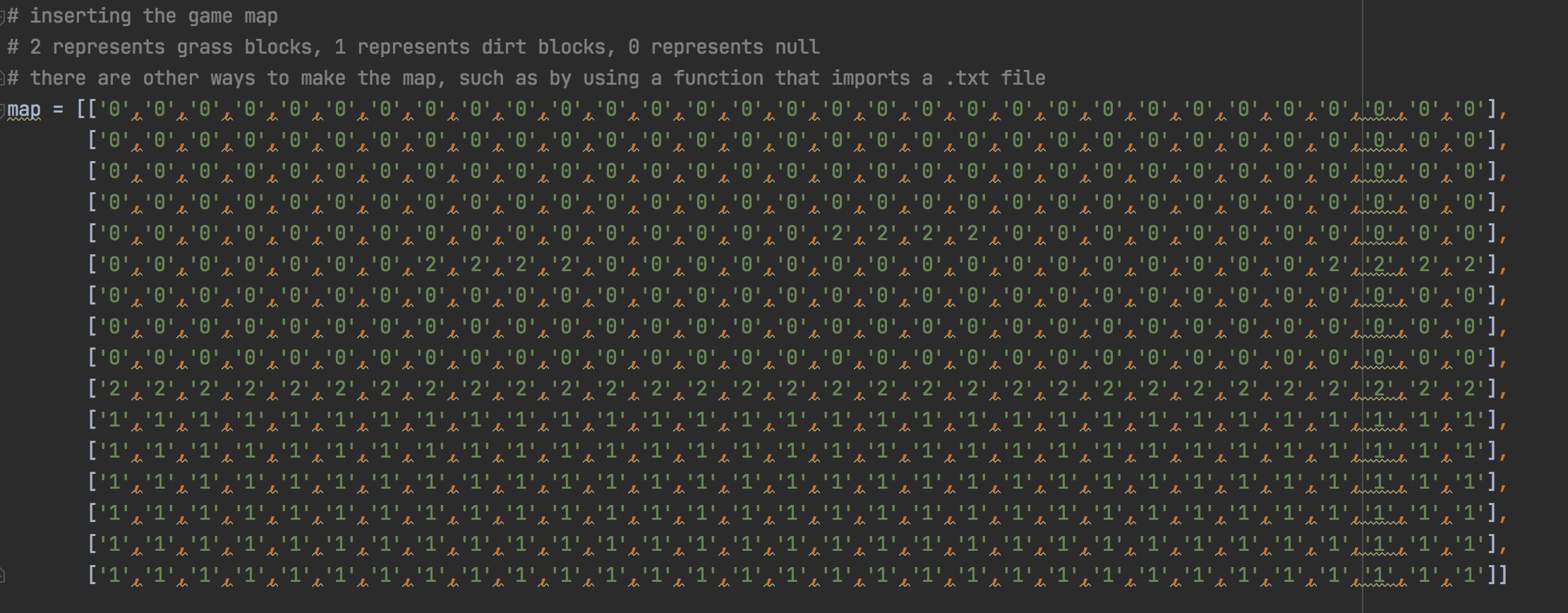
Gameplay: The player will be able to navigate through a map by using the **arrow** keys. Left and Right arrow keys to move horizontally and the Up arrow key to jump.

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

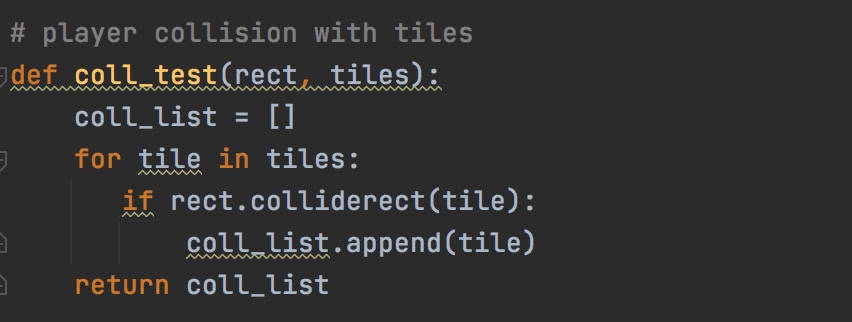
In this final project, all the coding will be done in 1 main.py file. I chose to do this as I very much prefer to see all my codes in a single file rather than in multiple files and then having these files be imported into the main.py file, which may result in unwanted errors. I will only be using 2 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**

****

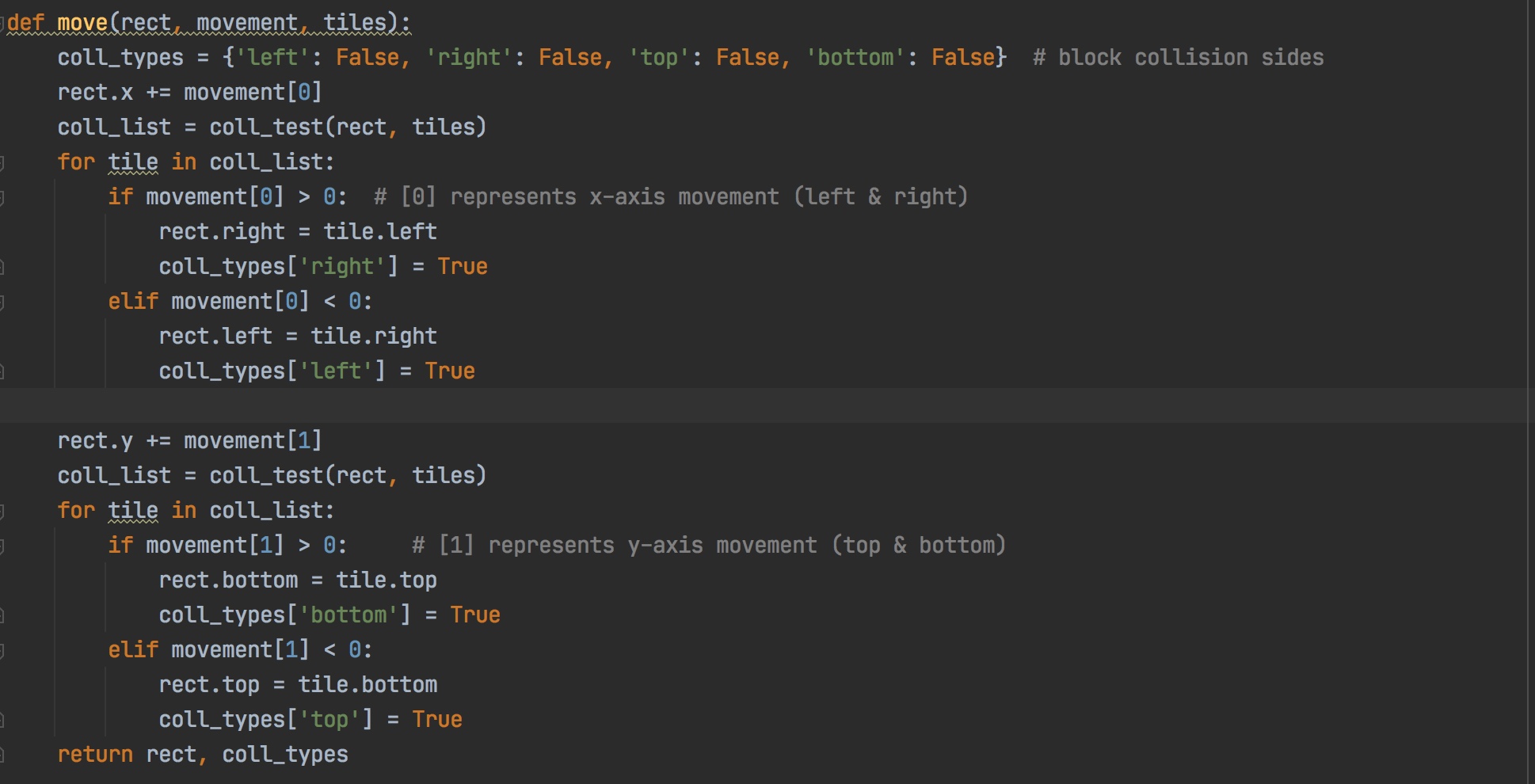
The code above shows the game map. The terrain is coded manually with “,” used as a separator. The number 2 represents the grass block, 1 represents the dirt block and 0 represents null or nothing. I decided to include the map in the main.py file instead of importing it as a .txt or a .rtf is due to 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**

****

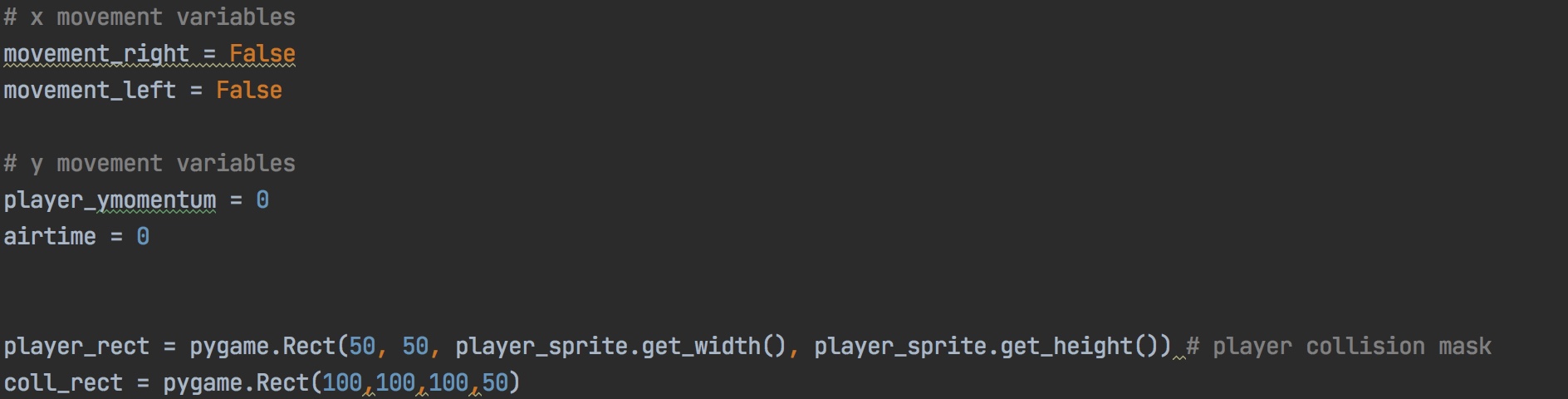
For player collision detection or collision mask, I made a function coll\_test so that the player can collide with the tiles in the map without the latter overlapping with other tiles. The function rect.colliderect() is used to prevent this. The if statement here is used so that when 2 tiles begins to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

**Movement**

****

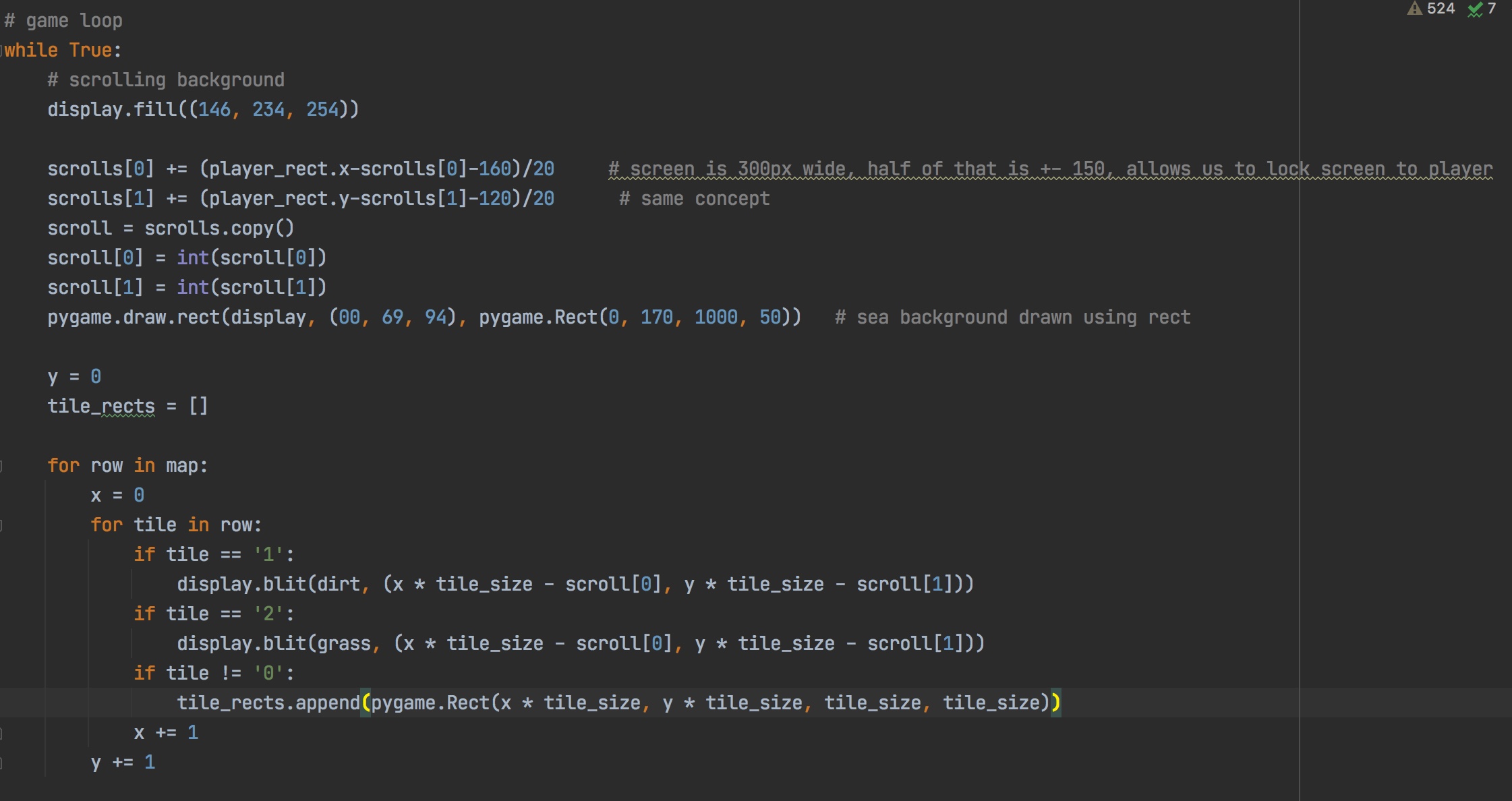
The def move function is used to manipulate player movement along the tiles. Coll\_types represent the block collision sides; a block has 4 sides thus there are 4 types needed. Rect.x is the collision mask along the x-axis of the tiles, movement [0], represents the horizontal movement of the player. 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.

# Flowchart

## Proof of Working Program

# Reflection & Experience

# References