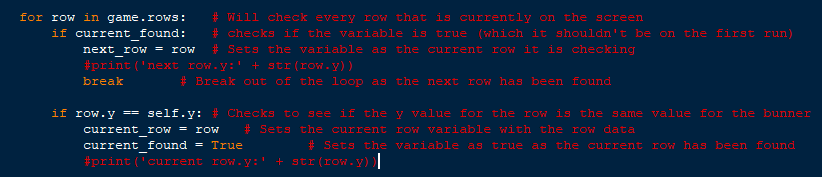
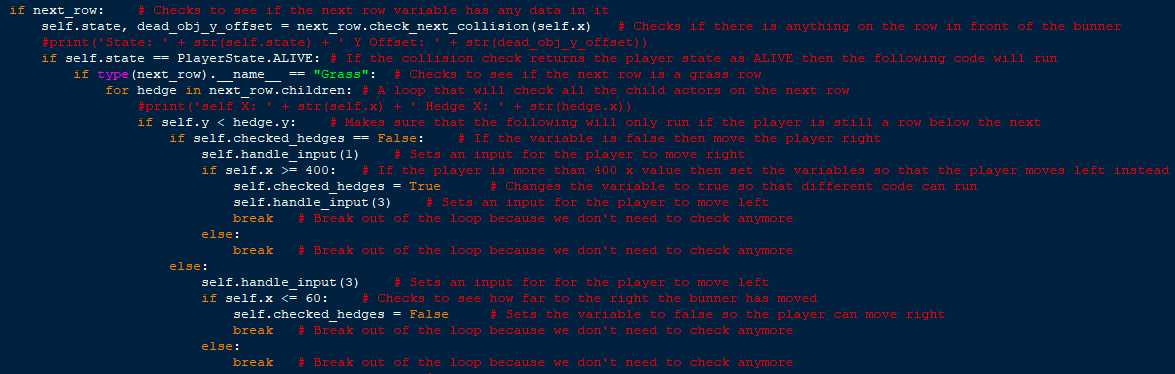
**Artificial Intelligence for Game Developers Course Work 1 Technical Report**

The objective of this project was to create an AI that would be able to traverse obstacles in the Code the Classics game “Infinite Bunner” with the addition of some human like elements such as being able to make some mistakes and dying. The implementation developed uses Finite State Machine (FSM) technique to decide whether the Bunner should move depending on what is in front of it. I decided against using A\* pathfinding as it would make the code more complicated and would have to search every row instead of just the one ahead of the Bunner.

Using the FSM, the AI was able to determine if it would die if it moved forward or if it would have to move around a hedge before moving forward. Before starting the implementation however, there were a number of problems that had to be identified. These problems being, how to check what is on the row ahead of the Bunner, checking if the Bunner would die if it moved forward and how to add human like elements. As stated, before A\* pathfinding would mean the AI would have to search every row and all its child actors to determine when and where it could move for the best outcome. Based on this it was decided FSM was a better option.

The first issue that was addressed was the finding the row in front of the Bunner problem. In order to do this the code checks each row that is on the screen and finds the row that currently has the Bunner on it using the following code. 

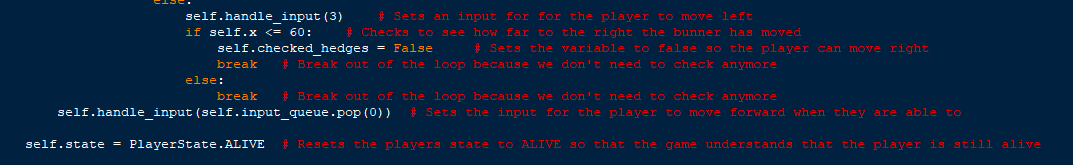
After the AI has found the current row the Bunner is on it can then find the next row by running the same loop but this time if the current has already been found it will run a different bit of code which sets the next row variable. We can confirm this works too by doing a simple print command to display the y value of both the current row and next row. As we know that the difference between rows y value is 40, we know this code is finding the correct rows.

 The next thing the AI does is find out what is on the next row. This is done with the following code

After the next row has been found, the AI then does a collision check with the next row variable to see if there is anything there that would kill the Bunner. This check will return a player state based on what is there. It will then check if the collision check returned an ALIVE state in which it will then check to see what type of row it is. Again, we can test this with a simple print command that will show the state that has been returned. 

As you can see when there was a car in front of the Bunner the print command would return the SPLAT state but once the car is no longer there it returns the ALIVE state.

If it is a Grass row it will then all the child actors on that row to see if there is a hedge on that row. There is then an If statement to make sure the Bunner is on the previous row as the hedge so that it can run the correct code that will allow it to go around the hedge. After checking to see if the checked\_hedges variable is set too false, which it is by default, in which the Bunner will then move right until it has reached a certain threshold and if it still hasn’t past the hedge yet it will then move left until it has. Finally, the AI inputs a forward command to continue moving forward when it can.

 After all these checks are complete it resets the player state to ALIVE so that it is not still a dead state after the next row collision check.