When starting to integrate scripting into my platformer game, I first started with doing so by creating a ScriptManager class that would be for calling and handling different types of scripts to be managed in the game. Afterwards, I created new methods in the class named callInput(), callCollsion(), callSpawn() and callDeath() that would be mainly to raise and handle the respective event calls being managed in the game. In order to get the methods to call their respective events, I implemented the methods so that they would essentially be calling the respective EventManager methods by passing in similar parameters that are similarly used with their respective methods. Specifically, the callInput() method raised and handled the input event by calling the input() method in the EventManager class, the callCollision() method raised and handled the collision event by calling the collision() in the EventManager class with similar parameters, the callSpawn() method raised and handled the character spawn event by calling the spawn() method in the EventManager class with similar parameters, and the callDeath() method raised and handled the character death event by using similar implementation with the death() method in the EventManager class with similar parameters. By handling the ScriptManager class this way, it made it pretty efficient to create scripts for the required events by using a ScriptManager object while also updating various game objects by calling the methods defined in the ScriptManager class. After defining my ScriptManager class, I then created a ScriptManager object called “sm”, and using the object, I then replaced all the calls with my EventManager methods through my “ems” object in the main() method with the respective ScriptManager methods through my “sm” object. For instance, the ems.input() call was replaced with the sm.callInput() call, the ems.death() call was replaced with the sm.callDeath() call, and ems.collision() call was replaced with the sm.callCollision() call. When carrying the ScriptManager class implementation and use in the platformer game to my maze and racer games, the ScriptManager class implementation and usage in those games basically remained the same as how it was done in the platformer game. However, because the spawn() and death() methods in my EventManager class for those 2 games were slightly modified so that no parameters were passed in those methods, I had to modify the callSpawn() and callDeath() methods for the ScriptManager class for those 2 games so that they also similarly didn’t have any parameters passed into those methods. Overall, my script manager implementation was pretty effective as the majority of the implementation for the script manager remained constant with all 3 games I created, with only minor changes being made with the implementation when creating my maze and racer games. To go about implementing a form of input abstraction into my platformer game, I decided to go about editing the input() of my EventManger class implementation for the platformer game to register and handle a new input sequence that’s possibly inputted by the player. When deciding what new behavior to implement with the new input sequence along with what existing keys already being implemented to use for the input sequence, I decided to make some game implementation where if the player pressed both the left and right arrow keys at the same time, the game would essentially reset and the player would start back at the beginning of the game. In order to achieve this functionality with the new input combination, I added a new check at the beginning of the input() method to see if the player was pressing both the left and right arrow keys at the same time and if that was the case, then all the global variables used in the game such as “nextLevel” and “canJump” would be set back to their initial values when starting the game. In addition, I also set the positions of my player and moving platform objects back to their initial positions when starting the game, and doing it like this sets the player back to the start of the 1st level and renders the 1st level in the game on the screen while making sure everything is rendered in their correct positions.