3. WRITTEN RESPONSES

3 a.

3.a.i.

The purpose of this program is to entertain the user through a fun obstacle course in which the user has to move a red ball from the starting point to a golden mushroom without bumping into the walls, and while also earning points.

3.a.ii.

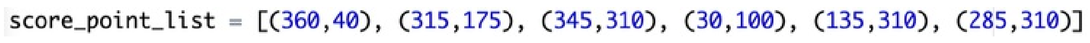
The video shows the game running with the red ball at a speed of 2x, and also at the speed of 3x. The video illustrates how each time the user's red ball bumped into a wall the ball was back at the starting position and the game restarted. The video also shows the end results page for the first run of the game in which the player earned 200 points, and took two tries to finish.

3.a.iii.

The inputs of this program include the mouse keys which move the red ball throughout the screen, and the mouse clicks used to proceed from the welcome screen, and the message banners that the program outputs. The program also outputs a result screen with the number of points the user earned, and the tries that they took to finish.

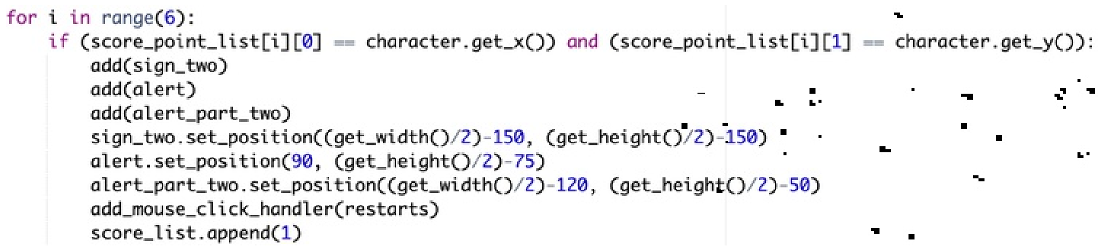
3 b.

3.b.i.



```
score_point_list = [(360,40), (315,175), (345,310), (30,100), (135,310), (285,310)]
```

3.b.ii.



```
for i in range(6):
    if (score_point_list[i][0] == character.get_x()) and (score_point_list[i][1] == character.get_y()):
        add(sign_two)
        add(alert)
        add(alert_part_two)
        sign_two.set_position((get_width()/2)-150, (get_height()/2)-150)
        alert.set_position(90, (get_height()/2)-75)
        alert_part_two.set_position((get_width()/2)-120, (get_height()/2)-50)
        add_mouse_click_handler(restarts)
        score_list.append(1)
```

3.b.iii.

The list being processed is named score_point_list.

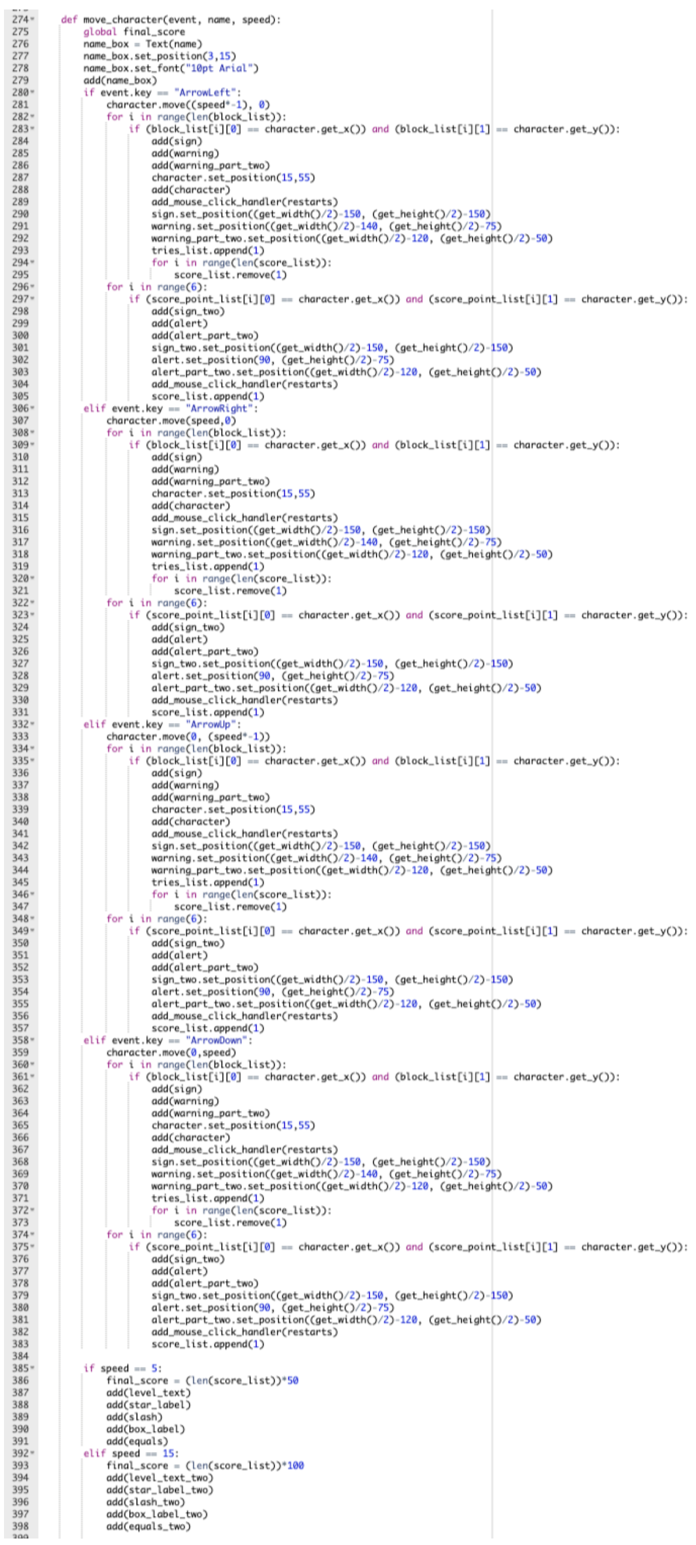
3.b.iv.

The score_point_list contains the coordinates of all the stars and mystery blocks which allows the program to identify where these items that give the user points are located.

3.b.v.

As shown in the second image, the list manages complexity in the program as it allows it to check if the user moved the red ball onto the stars or the mystery blocks all at once in a much quicker manner, through the use of a for loop and an if statement. If not for the list, I would've had to create several separate if statements for each of the different coordinate points stored in the list, and check if that equals the current location of the red ball. The list allows the for loop to access a different coordinate point each time the loop runs, allowing it to check through the whole list to see if the current coordinates of red ball equals one of the coordinates stored in the list.

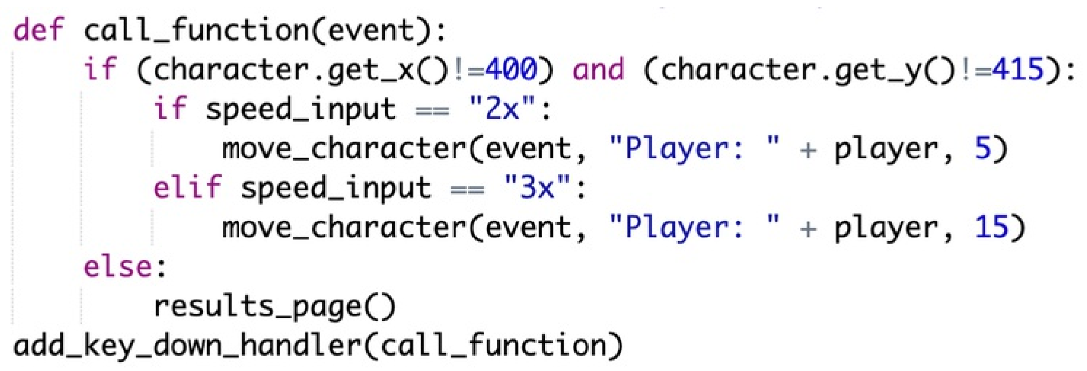
3 c.
3.c.i.



```python
def move_character(event, name, speed):
    global final_score
    name_box = Text(name)
    name_box.set_position(3,15)
    name_box.set_font("10pt Arial")
    add(name_box)
    if event.key == "ArrowLeft":
        character.move((speed*-1), 0)
        for i in range(len(block_list)):
            if (block_list[i][0] == character.get_x()) and (block_list[i][1] == character.get_y()):
                add(sign)
                add(warning)
                add(warning_part_two)
                character.set_position(15,55)
                add(character)
                add_mouse_click_handler(restarts)
                sign.set_position((get_width()/2)-150, (get_height()/2)-150)
                warning.set_position((get_width()/2)-140, (get_height()/2)-75)
                warning_part_two.set_position((get_width()/2)-120, (get_height()/2)-50)
                tries_list.append(1)
                for i in range(len(score_list)):
                    score_list.remove(1)
        for i in range(6):
            if (score_point_list[i][0] == character.get_x()) and (score_point_list[i][1] == character.get_y()):
                add(sign_two)
                add(alert)
                add(alert_part_two)
                sign_two.set_position((get_width()/2)-150, (get_height()/2)-150)
                alert.set_position(90, (get_height()/2)-75)
                alert_part_two.set_position((get_width()/2)-120, (get_height()/2)-50)
                add_mouse_click_handler(restarts)
                score_list.append(1)
    elif event.key == "ArrowRight":
        character.move(speed,0)
        for i in range(len(block_list)):
            if (block_list[i][0] == character.get_x()) and (block_list[i][1] == character.get_y()):
                add(sign)
                add(warning)
                add(warning_part_two)
                character.set_position(15,55)
                add(character)
                add_mouse_click_handler(restarts)
                sign.set_position((get_width()/2)-150, (get_height()/2)-150)
                warning.set_position((get_width()/2)-140, (get_height()/2)-75)
                warning_part_two.set_position((get_width()/2)-120, (get_height()/2)-50)
                tries_list.append(1)
                for i in range(len(score_list)):
                    score_list.remove(1)
        for i in range(6):
            if (score_point_list[i][0] == character.get_x()) and (score_point_list[i][1] == character.get_y()):
                add(sign_two)
                add(alert)
                add(alert_part_two)
                sign_two.set_position((get_width()/2)-150, (get_height()/2)-150)
                alert.set_position(90, (get_height()/2)-75)
                alert_part_two.set_position((get_width()/2)-120, (get_height()/2)-50)
                add_mouse_click_handler(restarts)
                score_list.append(1)
    elif event.key == "ArrowUp":
        character.move(0, (speed*-1))
        for i in range(len(block_list)):
            if (block_list[i][0] == character.get_x()) and (block_list[i][1] == character.get_y()):
                add(sign)
                add(warning)
                add(warning_part_two)
                character.set_position(15,55)
                add(character)
                add_mouse_click_handler(restarts)
                sign.set_position((get_width()/2)-150, (get_height()/2)-150)
                warning.set_position((get_width()/2)-140, (get_height()/2)-75)
                warning_part_two.set_position((get_width()/2)-120, (get_height()/2)-50)
                tries_list.append(1)
                for i in range(len(score_list)):
                    score_list.remove(1)
        for i in range(6):
            if (score_point_list[i][0] == character.get_x()) and (score_point_list[i][1] == character.get_y()):
                add(sign_two)
                add(alert)
                add(alert_part_two)
                sign_two.set_position((get_width()/2)-150, (get_height()/2)-150)
                alert.set_position(90, (get_height()/2)-75)
                alert_part_two.set_position((get_width()/2)-120, (get_height()/2)-50)
                add_mouse_click_handler(restarts)
                score_list.append(1)
    elif event.key == "ArrowDown":
        character.move(0,speed)
        for i in range(len(block_list)):
            if (block_list[i][0] == character.get_x()) and (block_list[i][1] == character.get_y()):
                add(sign)
                add(warning)
                add(warning_part_two)
                character.set_position(15,55)
                add(character)
                add_mouse_click_handler(restarts)
                sign.set_position((get_width()/2)-150, (get_height()/2)-150)
                warning.set_position((get_width()/2)-140, (get_height()/2)-75)
                warning_part_two.set_position((get_width()/2)-120, (get_height()/2)-50)
                tries_list.append(1)
                for i in range(len(score_list)):
                    score_list.remove(1)
        for i in range(6):
            if (score_point_list[i][0] == character.get_x()) and (score_point_list[i][1] == character.get_y()):
                add(sign_two)
                add(alert)
                add(alert_part_two)
                sign_two.set_position((get_width()/2)-150, (get_height()/2)-150)
                alert.set_position(90, (get_height()/2)-75)
                alert_part_two.set_position((get_width()/2)-120, (get_height()/2)-50)
                add_mouse_click_handler(restarts)
                score_list.append(1)

    if speed == 5:
        final_score = (len(score_list))*50
        add(level_text)
        add(star_label)
        add(slash)
        add(box_label)
        add(equals)
    elif speed == 15:
        final_score = (len(score_list))*100
        add(level_text_two)
        add(star_label_two)
        add(slash_two)
        add(box_label_two)
        add(equals_two)
```

3.c.ii.



```
def call_function(event):
    if (character.get_x()!=400) and (character.get_y()!=415):
        if speed_input == "2x":
            move_character(event, "Player: " + player, 5)
        elif speed_input == "3x":
            move_character(event, "Player: " + player, 15)
    else:
        results_page()
add_key_down_handler(call_function)
```

3.c.iii.

This selected procedure is the main function that makes the game work, by moving the ball, keeping track of the user's score, restarting the game when they bumped into a wall and letting the user know of such.

3.c.iv.

It moves the red ball depending on which mouse key the user inputted through the use of selection. If the user pushes the right key the ball moves to the right a certain amount, if user pushes the left key it moves to the left, if user enters the up key the ball moves up, and if user pushes the down key it moves down. After each time the ball moves, the procedure uses a for loop and if statements to check if the ball is on a coordinate of the walls, or the score items such as the stars, by accessing the lists that these coordinates are located in and by checking if even one of those coordinates matches the balls current coordinates. If the coordinates of the ball matches a coordinate of the wall, the procedure lets the user know that they bumped into a wall and makes them restart with zero points, however if the ball's coordinates match one of the coordinates of the score items, then it lets the user know that they got some points. It keeps track of these points by appending a number to a list each time the user gets a point. It then multiplies the length of the list by the amount of points of each item, and stores that in a variable which is the user's final score.

3 d.

3.d.i.

First call:

The two test cases are based on passing a different amount by which the red ball is moved each time the specific mouse keys are clicked, in order to create different speeds for the character. The first call is move_character(event, “Player: ” + player, 5).

Second call:

The second call is move_character(event, “Player: ” + player, 15).

3 d.ii.

Condition(s) tested by first call:

With the argument of the first call, the if-statement on line 385 which checks if the speed parameter is equal to 5 or if it equals 15, will execute the first part of this selection statement, which consists of lines 386-391.

Condition(s) tested by second call:

With the argument of the second call, the if-statement on line 385 which checks if the speed parameter is equal to 5 or if it equals 15, will execute the second part of this selection statement, which consists of lines 393-398.

3.d.iii.

Results of the first call:

First call results in the points of each the stars and mystery blocks to be 50, and displays this at the top of the screen. It also displays that the difficulty of the game is medium at this level, and with this argument the red ball moves in increments of 5.

Results of the second call:

Second call results in the points of each the stars and mystery blocks to be 100, and displays this at the top of the screen. It also displays that the difficulty of the game is hard at this level, and with this argument the red ball moves in increments of 15.