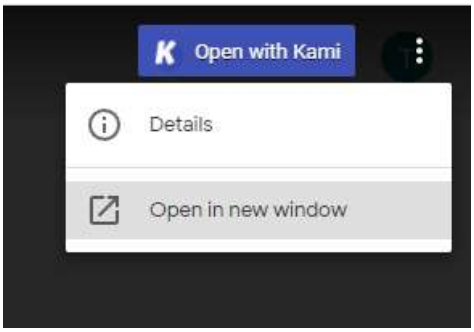
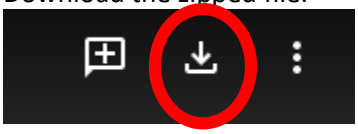
Part 1
Space Mission Directions

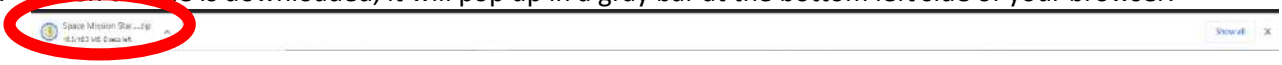
1. Navigate out to the Google Classroom for this class.
2. Locate the Space Mission Part 1 assignment.
3. Click on the, “Mission Python Starting Files” zip folder that is attached to the assignment.
4. Click the three dots at the top, then select, “Open in new window” from the list.



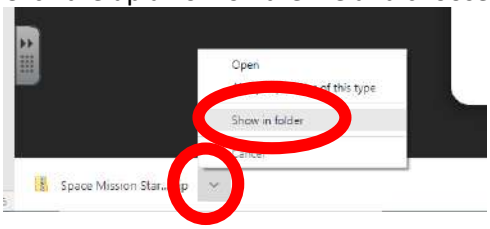
5. Download the zipped file.



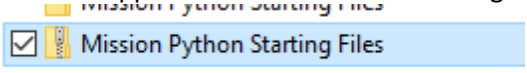
6. When the file is downloaded, it will pop up in a gray bar at the bottom left side of your browser.



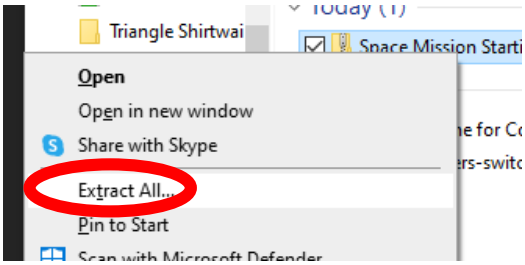
7. Click the up arrow on the file and choose “Show in Folder” from the list of options.



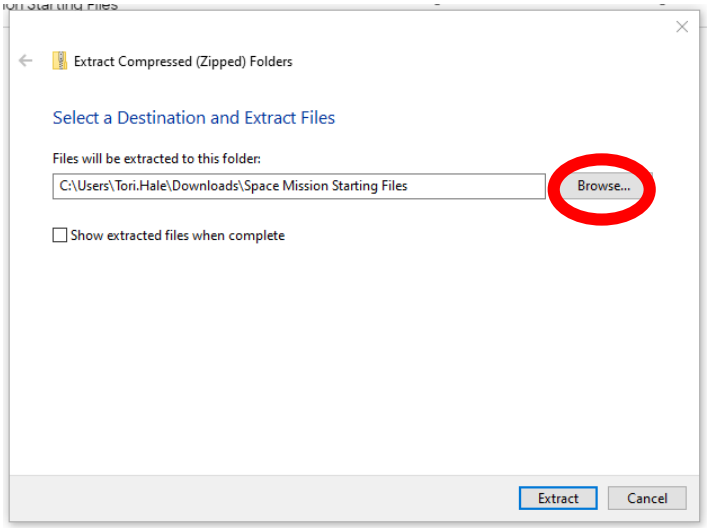
8. A dialog box showing all of the files in your “Downloads” folder will pop up on your computer.
9. Select the zipped folder so that the file is highlighted with a light blue background.



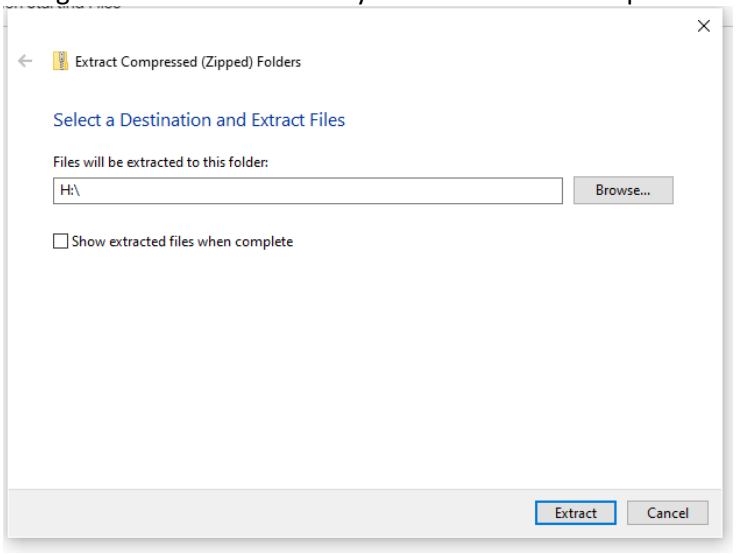
10. Right-click on the zipped folder.
11. Choose “Extract All” from the list of options.



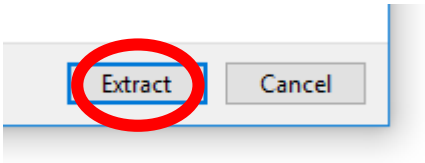
12. A dialog box will pop up asking you where you want to save the extracted file to. Click the "Browse" button.



13. Navigate to the folder where you want to save the Space Mission Files at.

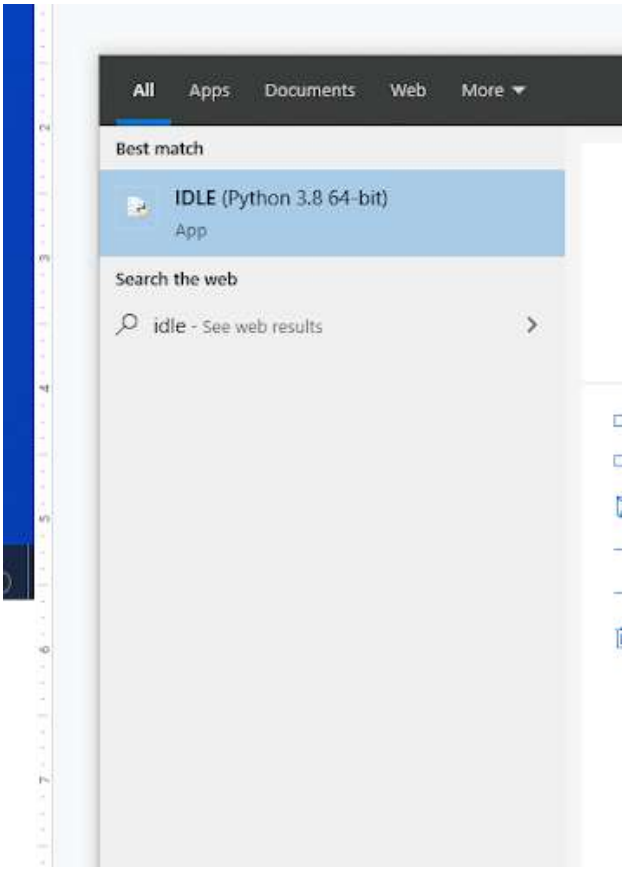


14. Click “Extract”



15. You should now see this file in your Student Drive in the folder that you specified.

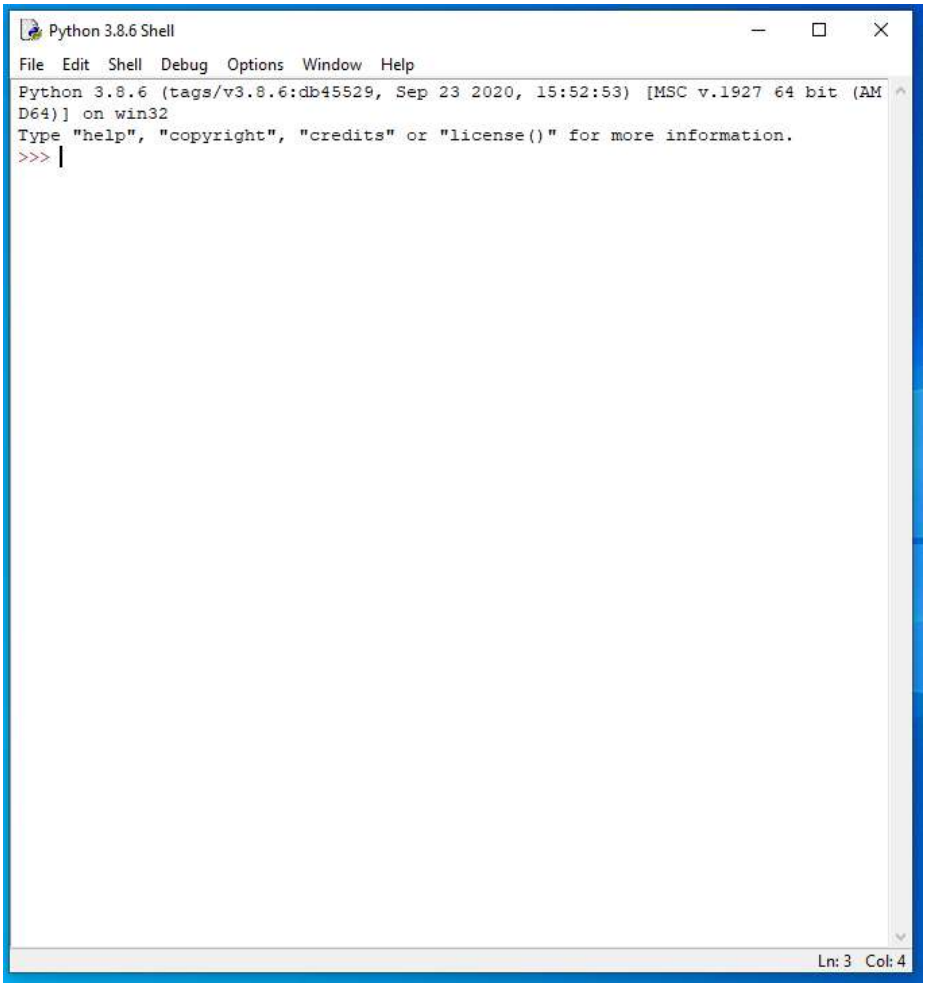
16. We are now ready to start adding code to our file. Using your Windows button menu, find and launch your IDLE program.



IDLE is the integrated development environment associated with Python. It is made up of a code editor where you type your code along with other helpful tools that allow you to write, save, and test run programs.

IDLE is designed to recognize Python code, compile Python code, and provide basic debugging tips to programmers if there are problems with their code.

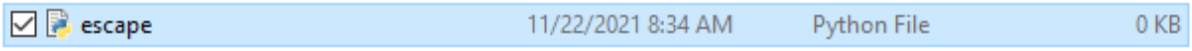
17. Your IDLE window should look something like this once it has launched.:



On Startup, IDLE will display the Python Shell, which can be used to give commands to the computer's operating system. Since we are viewing the shell through IDLE and not the actual command prompt window, the commands that we type into the Shell will not communicate directly with our operating system. However, you can type similar commands in the Python Shell directly from the Python program (not through IDLE) and, if you have permission to access the operating system's commands, you can communicate with the computer's operating system that way.

In IDLE, the shell is mainly used as a launching screen for other activities that we will do, like writing code for our game or debugging a file.

18. Go to File > Open and then browse in the Starting Files folder I gave you to find the escape python file that I gave you. This file is empty right now.



19. Your escape.py file will open up.

20. Type the code that you see on Lines 1 – 3 of the screenshot below.

```
1 # Escape
2
3 import time, random, math
```

Line 1 contains a comment with the name of the game.

Line 3 imports the time, random, and math modules into the code.

21. Press ENTER twice.

22. Type the code you see on Lines 5 – 21 of the screenshot below.

```
1  # Escape
2
3  import time, random, math
4
5  ###############
6  ## VARIABLES ##
7  ###############
8
9  WIDTH = 800 #window size
10 HEIGHT = 800
11
12 #PLAYER variables
13 PLAYER_NAME = "Alice"
14 FRIEND1_NAME = "Jack"
15 FRIEND2_NAME = "Matthew"
16 current_room = 31 # start room = 31
17
18 top_left_x = 100
19 top_left_y = 150
20
21 DEMO_OBJECTS = [images.floor, images.pillar, images.soil]
```

Lines 5 – 7 contains a comment designating this section of code as the place where the variables for the game are defined.

Line 9 creates the WIDTH variable and sets it equal to 800.

Line 10 creates the HEIGHT variable and sets it equal to 800. The window size of our game will be 800 x 800.

Line 12 contains a comment explaining that the variables under this section of code will deal with the player.

Line 13 creates a variable called PLAYER_NAME and sets it equal to the name Alice. You can change this name to your own name if you would like.

Line 14 creates a variable called FRIEND1_NAME. This variable is set to be equal to one of your friend's name. In the example, I use the name "Jack," but you can change this name to whatever you'd like.

Line 15 creates another variable called FRIEND2_NAME. This variable is set to be equal to another one of your friends' names. In the example, I use the name "Matthew," but you can change this name to whatever you'd like.

Line 16 creates another variable called current_room. The variable's starting value is set to 31, meaning that the player starts in room #31. We haven't created our room list yet, so we will refer back to this later in the program.

Line 18 creates a variable called top_left_x and sets the value to 100. This variable will hold the position of the top left x coordinate for the room that we will draw.

Line 19 creates a variable called top_left_y and sets the value to 150. This variable will hold the position of the top left y coordinate for the room that we will draw.

Line 21 creates a variable called DEMO_OBJECTS. The DEMO_OBJECTS variable's value is set to be a list the includes the floor, pillar, and soil images from the images folder that you downloaded. The DEMO_OBJECTS variable list contains the images to use for the objects in the room.

23. Press ENTER three times.

24. Type the code you see on Lines 24 – 32 of the screenshot below.

```
1  # Escape
2
3  import time, random, math
4
5  ################
6  ## VARIABLES ##
7  ################
8
9  WIDTH = 800 #window size
10 HEIGHT = 800
11
12 #PLAYER variables
13 PLAYER_NAME = "Alice"
14 FRIEND1_NAME = "Jack"
15 FRIEND2_NAME = "Matthew"
16 current_room = 31 # start room = 31
17
18 top_left_x = 100
19 top_left_y = 150
20
21 DEMO_OBJECTS = [images.floor, images.pillar, images.soil]
22
23
24 ################
25 ##    MAP     ##
26 ################
27
28 MAP_WIDTH = 5
29 MAP_HEIGHT = 10
30 MAP_SIZE = MAP_WIDTH * MAP_HEIGHT
31
32 GAME_MAP = [ ["Room 0 - where unused objects are kept", 0, 0, False, False] ]
```

Lines 24 – 26 contain another comment designating this section of code as the place where the map for the game is defined.

Line 28 creates a variable called MAP_WIDTH and sets its initial value equal to 5. This variable designates that our map will be 5 tiles wide.

Line 29 creates a variable called MAP_HEIGHT and sets its initial value equal to 10. This variable designates that our map will be 10 tiles tall.

Line 30 creates a variable called MAP_SIZE. This variable's value is set to be the value of the MAP_WIDTH variable (in this case, it is 5) times the value of the MAP_HEIGHT variable (in this case, 10). So, in this example, the MAP_SIZE variable is equal to 50 tiles.

Line 32 creates a GAME_MAP variable and gives the game the data for room 0. This room will be used for storing items that aren't in the game yet because the player hasn't discovered them. However, we want to load all items at the start of the game to minimize load time later. This room isn't a real room but is used as a holding list/place for objects that the player hasn't found yet and can't use.

The parameters on Line 32 contain the room name (Room 0…), the height and width of the room (0 and 0, since it isn't a real room we want the player to visit), and whether it will have an exit at the top or right side of the room (both set to False).

25. Press ENTER twice.

26. Type the code you see on Lines 34 – 36 of the screenshot below. Ensure your indentation matches what is shown in the screenshot.

```
28 MAP_WIDTH = 5
29 MAP_HEIGHT = 10
30 MAP_SIZE = MAP_WIDTH * MAP_HEIGHT
31
32 GAME_MAP = [ ["Room 0 - where unused objects are kept", 0, 0, False, False] ]
33
34 outdoor_rooms = range(1, 26)
35 for planetsectors in range(1, 26): #rooms 1 to 25 are generated here
36     GAME_MAP.append( ["The dusty planet surface", 13, 13, True, True] )
```

Line 34 creates a variable called outdoor_rooms and sets its value to be the range from 1 to 26. Essentially, 25 rooms. We will have 25 outdoor rooms on our planet that all look the same.

Line 35 will begin a "for" loop that runs 25 times (for rooms in the range of 1 – 26; remember, the range includes all items UP TO the last number in the range. Rooms in the range of 1 – 25 will be created, stopping when it gets to room 26.)

Line 36 will append the data created for each room (Room 1 – 25). Remember, all outdoor rooms are the same and will look the same, so the same attributes will add 25 times.

The attributes added to the GAME_MAP rooms list include the room name ("The dusty planet surface"), the height and width of the room (13 and 13), and a top and right side exit (both True).

The Lines 35 – 36 are done looping, your GAME_MAP list contains a total of 26 rooms: Room 0 that is used to hold objects that haven't been unlocked by the player yet, and rooms 1 – 25, which are outside rooms that are 13 x 13 and include top and right side exits.

27. Press ENTER twice.

28. Type the code that you see on Lines 38 – 65 of the screenshot below. Ensure your indentation matches what is shown in the screenshot.

```
32 GAME_MAP = [ ["Room 0 - where unused objects are kept", 0, 0, False, False] ]
33
34 outdoor_rooms = range(1, 26)
35 for planetsectors in range(1, 26): #rooms 1 to 25 are generated here
36     GAME_MAP.append( ["The dusty planet surface", 13, 13, True, True] )
37
38 GAME_MAP  += [
39         #["Room name", height, width, Top exit?, Right exit?]
40         ["The airlock", 13, 5, True, False], # room 26
41         ["The engineering lab", 13, 13, False, False], # room 27
42         ["Poodle Mission Control", 9, 13, False, True], # room 28
43         ["The viewing gallery", 9, 15, False, False], # room 29
44         ["The crew's bathroom", 5, 5, False, False], # room 30
45         ["The airlock entry bay", 7, 11, True, True], # room 31
46         ["Left elbow room", 9, 7, True, False], # room 32
47         ["Right elbow room", 7, 13, True, True], # room 33
48         ["The science lab", 13, 13, False, True], # room 34
49         ["The greenhouse", 13, 13, True, False], # room 35
50         [PLAYER_NAME + "'s sleeping quarters", 9, 11, False, False], # room 36
51         ["West corridor", 15, 5, True, True], # room 37
52         ["The briefing room", 7, 13, False, True], # room 38
53         ["The crew's community room", 11, 13, True, False], # room 39
54         ["Main Mission Control", 14, 14, False, False], # room 40
55         ["The sick bay", 12, 7, True, False], # room 41
56         ["West corridor", 9, 7, True, False], # room 42
57         ["Utilities control room", 9, 9, False, True], # room 43
58         ["Systems engineering bay", 9, 11, False, False], # room 44
59         ["Security portal to Mission Control", 7, 7, True, False], # room 45
60         [FRIEND1_NAME + "'s sleeping quarters", 9, 11, True, True], # room 46
61         [FRIEND2_NAME + "'s sleeping quarters", 9, 11, True, True], # room 47
62         ["The pipeworks", 13, 11, True, False], # room 48
63         ["The chief scientist's office", 9, 7, True, True], # room 49
64         ["The robot workshop", 9, 11, True, False] # room 50
65         ]
```

Line 38 states that we will add a new list to the end of the GAME_MAP list variable. After this block of code, the GAME_MAP list variable will contain two lists: The list with Rooms 0 – 25 and the list with rooms 26 – 50.

Line 39 contains a comment explaining the information that is included in each room list.

Lines 40 – 64 contain the various rooms that you will add to your game. Line 65 contains the final square list bracket that is needed to complete the list.

Each of the Rooms 26 – 50 will be different, so we need to enter their data manually. Each room in the list contains the room name, the height and width of the room, and whether the room will have a top exit or right edit.

Rooms 46 and 47 contain the sleeping quarters for your FRIEND1 and FRIEND2, so their names will be used in the title of the room.

Be sure to add a comma at the end of each line on Lines 40 – 63 since you will be creating smaller lists (with the room information and size) inside a bigger list (the GAME_MAP room list).

29. Press ENTER twice.

30. Type the code you see on Lines 67 – 68 of the screenshot below.

```
60          [FRIEND1_NAME + "'s sleeping quarters", 9, 11, True, True], # room 46
61          [FRIEND2_NAME + "'s sleeping quarters", 9, 11, True, True], # room 47
62          ["The pipeworks", 13, 11, True, False], # room 48
63          ["The chief scientist's office", 9, 7, True, True], # room 49
64          ["The robot workshop", 9, 11, True, False] # room 50
65          ]
66
67  #simple sanity check on map above to check data entry
68  assert len(GAME_MAP)-1 == MAP_SIZE, "Map size and GAME_MAP don't match"
```

Line 67 contains a comment.

Line 68 uses the "assert" method to check that the map data makes sense. We check whether the length of the GAME_MAP – 1 (we want to subtract 1 from the length because of the Room 0 that we don't want the player to visit) is equal to the size of the map, in tiles. In other words, do we have the same amount of items in the list as we have tiles in the map?

The assert() function is a simple function that will evaluate a condition. If a condition evaluates to "True" (in this case, if the number of items in the GAME_MAP is equal to the MAP_SIZE), the program will continue to run. If the assert() function returns as "False," the program will stop.

31. Press ENTER three times.

32. Type the code you see on Lines 71 – 73 of the screenshot below.

```
67  #simple sanity check on map above to check data entry
68  assert len(GAME_MAP)-1 == MAP_SIZE, "Map size and GAME_MAP don't match"
69
70
71  ###############
72  ## MAKE MAP  ##
73  ###############
```

Lines 71 - 73 contain another comment designating this section of code as the place where the map for the game is drawn.

33. Press ENTER twice.

34. Type the code you see on Lines 75 – 79 of the screenshot below. Ensure your indentation matches what is shown in the screenshot.

```
65          ]
66
67  #simple sanity check on map above to check data entry
68  assert len(GAME_MAP)-1 == MAP_SIZE, "Map size and GAME_MAP don't match"
69
70
71  ###############
72  ## MAKE MAP  ##
73  ###############
74
75  def get_floor_type():
76      if current_room in outdoor_rooms:
77          return 2 # soil
78      else:
79          return 0 # tiled floor
```

Line 75 creates a new method called get_floor_type.

Line 76 will check to see if the current_room that the player is in is in the outdoor_room variable list. If it is, the game will return the value of 2, which will eventually equal soil. (Line 77).

If Line 76 is NOT true, meaning that the player is not in an outdoor_room, than the game will return the value of 0. This will eventually mean that the player is on a tiled floor (Lone 79).

35. Press ENTER twice.

36. Type the code you see on Lines 81 – 89 of the screenshot below. Ensure your indentation matches what is shown in the screenshot.

```
71 ###############
72 ## MAKE MAP  ##
73 ###############
74
75 def get_floor_type():
76     if current_room in outdoor_rooms:
77         return 2 # soil
78     else:
79         return 0 # tiled floor
80
81 def generate_map():
82 # This function makes the map for the current room,
83 # using room data, scenery data and prop data.
84     global room_map, room_width, room_height, room_name, hazard_map
85     global top_left_x, top_left_y, wall_transparency_frame
86     room_data = GAME_MAP[current_room]
87     room_name = room_data[0]
88     room_height = room_data[1]
89     room_width = room_data[2]
```

Line 81 creates another function called generate_map.

Lines 82 and 83 contain comments describing what this function will do.

Lines 84 and 85 reference the variables room_map, room_width, room_height, room_name, hazard_map, top_left_x, top_left_y, and wall_transparency_frame. Some of these variables have already been created and some of these variables will be created later. When a programmer references variables that were created OUTSIDE of the current method, he/she either needs to change these variables the “global” variables (using the word “global”) or perform some other workaround so that he or she can access and change the values of these variable’s from within a method in which they were not originally defined. That is the reason you see the “global” keyword – we are not able to change the value of all of the variables listed from within the generate_map variable.

Line 86 creates a variable called room_data. The variable’s value is set to the list of data we created for the current_room in the GAME_MAP list. This data includes the name and size of the room along with the room’s exits.

Line 87 creates the room_name variable and sets the variable’s value to be equal to the first item in the room_data list (the item with the list index value of 0). If you remember, each room we created was set up in a list, with the name of the room, size of the room, and exits of the room. The room_data list gathers this list of data for the current_room the player is in and stores it in the variable called room_data. Then, the room_name variable searches for the first item in the room_data list (the room’s name) and returns in.

Line 88 creates the room_height variable. The variable's value is set to be the second item in the room_data list (the item with the list index value of 1). This will return the room height for the current room the player is in.

Line 89 creates the room_width variable. The variable's value is set to be the third item in the room_data list (the item with the list index value of 2). This will return the room width for the current room the player is in.

37. Press ENTER twice.

38. Type the code you see on Lines 91 – 100 of the screenshot below. Ensure your indentation matches what is shown in the screenshot.

```
81  def generate_map():
82  # This function makes the map for the current room,
83  # using room data, scenery data and prop data.
84      global room_map, room_width, room_height, room_name, hazard_map
85      global top_left_x, top_left_y, wall_transparency_frame
86      room_data = GAME_MAP[current_room]
87      room_name = room_data[0]
88      room_height = room_data[1]
89      room_width = room_data[2]
90
91      floor_type = get_floor_type()
92      if current_room in range(1, 21):
93          bottom_edge = 2 #soil
94          side_edge = 2 #soil
95      if current_room in range(21, 26):
96          bottom_edge = 1 #wall
97          side_edge = 2 #soil
98      if current_room > 25:
99          bottom_edge = 1 #wall
100         side_edge = 1 #wall
```

This code is a continuation of the generate_map method that we started defining reviously.

Line 91 creates the floor_type variable. The variable's value will be set to the result of the get_floor_type method. We defined this method on Lines 75 – 79 of the game. The method will return the number 2 if the player is in an outdoor room or the number 0 if the player is not in an outdoor room, meaning that they are on a tiled floor.

Line 92 will check to see if the value of the current_room variable (the current room that the player is in) is between the range of 1 and 20. If it is, then that means the player is in an outdoor room. The bottom_edge variable is created and its value is set to 2 indicating that the floor type is soil. (Line 93) The side_edge variable is also created on Line 94 and its value is also set to 2, indicating that the floor type is soil.

If the current_room the player is in is NOT between room values 1 – 20, then Line 95 will run to check to see if the current_room number that the player is in is between room number 21 and 25. If this is true, a variable called bottom_edge will be created on Line 96 and its value will be

set to 1, meaning that the bottom edge of that room is the wall. Line 97 will create a variable called side_edge and its value will be set to 2, meaning that the side edge of the room is soil.

If Lines 92 and Line 95 are both false, Line 98 will run to check to see if the current_room number the player is in is greater than 25. If it is, the bottom_edge variable will be created on Line 99 and its value will be set to 1, indicating that the bottom edge of the room is a wall. Line 100 will also create a variable called side_edge and its value will also be set to 1, indicating that the side edge of the room is also a wall.

39. Press ENTER twice.

40. Type the code you see on Lines 102 – 109 of the screenshot below. Ensure your indentation matches what is shown in the screenshot.

```
81  def generate_map():
82  # This function makes the map for the current room,
83  # using room data, scenery data and prop data.
84      global room_map, room_width, room_height, room_name, hazard_map
85      global top_left_x, top_left_y, wall_transparency_frame
86      room_data = GAME_MAP[current_room]
87      room_name = room_data[0]
88      room_height = room_data[1]
89      room_width = room_data[2]
90
91      floor_type = get_floor_type()
92      if current_room in range(1, 21):
93          bottom_edge = 2 #soil
94          side_edge = 2 #soil
95      if current_room in range(21, 26):
96          bottom_edge = 1 #wall
97          side_edge = 2 #soil
98      if current_room > 25:
99          bottom_edge = 1 #wall
100         side_edge = 1 #wall
101
102     # Create top line of room map.
103     room_map=[[side_edge] * room_width]
104     # Add middle lines of room map (wall, floor to fill width, wall).
105     for y in range(room_height - 2):
106         room_map.append([side_edge]
107                         + [floor_type]*(room_width - 2) + [side_edge])
108     # Add bottom line of room map.
109     room_map.append([bottom_edge] * room_width)
```

Line 102 contains a comment.

Line 103 creates a new list variable called room_map. The variable's value is set to be equal to the value of the side_edge variable that was established in the previous lines multiplied by the room_width value. This will multiply the edge type by the width of the room. If the top edge has an exit in it, we will add that soon.

Line 104 contains another comment.

The middle rows of the room are made using a “for” loop on Line 105 that will add each row in turn to the end of the roommap list. All of the middle rows in a room are the same and are made up of the edge tile (either wall or soil) for the left side of the room, the floor in the middle, and the edge piece at the right side (either wall or soil). We subtract 2 from the room_width and room_height because we have two edge pieces on the room – the left and right side. Lines 106 – 107 will loop to generate the appropriate floor for each row in the room.

Line 108 contains another comment.

Line 109 creates the bottom edge of the room using the same method as we used on Line 103 to create the top edge of the room.

41. Press ENTER twice.

42. Type the code that you see on Lines 111 – 113 of the screenshot below. Ensure your indentation matches what is shown in the screenshot.

```
102     # Create top line of room map.
103     room_map=[[side_edge] * room_width]
104     # Add middle lines of room map (wall, floor to fill width, wall).
105     for y in range(room_height - 2):
106         room_map.append([side_edge]
107                         + [floor_type]*(room_width - 2) + [side_edge])
108     # Add bottom line of room map.
109     room_map.append([bottom_edge] * room_width)
110
111     # Add doorways.
112     middle_row = int(room_height / 2)
113     middle_column = int(room_width / 2)
```

Line 111 contains a comment.

We are going to put the exits in the middle of the walls. However, first we need to figure out exactly where the middle of the walls is. Line 112 creates a variable called middle_row. The variable’s value is set to be the integer value of the room_height variable divided by 2. We use the “int” operation because sometimes the division operation results in a number with a decimal. However, we want to work with whole numbers. The “int” operation will change the result of the division to a whole number.

Line 113 creates the middle_column variable. The variable’s value is set to be the integer value of the room_width variable divided by 2.

The two variables on Lines 112 and 113 will calculate the middle row of the room.

43. Press ENTER twice.

44. Type the code that you see on Lines 115 – 119 of the screenshot below. Ensure your indentation matches what is shown in the screenshot below.

```
111     # Add doorways.
112     middle_row = int(room_height / 2)
113     middle_column = int(room_width / 2)
114
115     if room_data[4]: # If exit at right of this room
116         room_map[middle_row][room_width - 1] = floor_type
117         room_map[middle_row+1][room_width - 1] = floor_type
118         room_map[middle_row-1][room_width - 1] = floor_type
```

Line 115 will check to see if the fifth value in the room_data list (the list item with the index of 4; this would be whether the room has an exit on the right side) is TRUE. If the room does have an exit on the right side, Lines 116 - 118 will run. If not, Lines 116 - 118 will be skipped and the program will move down to the code on Line 120.

If the room has right side exits, Lines 116 - 118 will change the three positions in the middle of the right wall from the edge type to the floor type, making a gap in the wall. Room_width - 1 finds the x position on the right edge of the exit (remember, index values start at 0).

For example, let's say the room width is 11 tiles. The index position for the right wall would be position 10. So, if we had the code change the tile with the index of 11, it wouldn't change anything because there would be no wall there to put a gap in.

45. Press ENTER twice.

46. Type the code you see on Lines 120 – 126 of the screenshot below. Ensure your indentation matches what is shown in the screenshot.

```
111     # Add doorways.
112     middle_row = int(room_height / 2)
113     middle_column = int(room_width / 2)
114
115     if room_data[4]: # If exit at right of this room
116         room_map[middle_row][room_width - 1] = floor_type
117         room_map[middle_row+1][room_width - 1] = floor_type
118         room_map[middle_row-1][room_width - 1] = floor_type
119
120     if current_room % MAP_WIDTH != 1: # If room is not on left of map
121         room_to_left = GAME_MAP[current_room - 1]
122         # If room on the left has a right exit, add left exit in this room
123         if room_to_left[4]:
124             room_map[middle_row][0] = floor_type
125             room_map[middle_row + 1][0] = floor_type
126             room_map[middle_row - 1][0] = floor_type
```

Before we check whether we need an exit for the left wall, we make sure the room isn't on the left edge of the map where there can be no exit. Line 120 uses the % operator to give us the remainder when we divide the current_room by the MAP_WIDTH variable (in this case, the number 5). If the position of the room the player is in is on the left edge, when we divide the room number by the number 5, the remainder would be 1. In this line, we are checking to make sure that the remainder after division is NOT 1 (in other words, Line 120 is checking to make sure that the room is NOT on the left edge of the map).

If Line 120 is true (meaning that the player is not in a room at the left edge of the map)m, Line 121 will create a new variable called room_to_left. The variable's value will be set to 1 less than the current number that the player is in on the game map. In other words, if the player is in room 40, then the room_to_left variable will be set to 39.

Line 122 contains a comment.

Line 123 will begin an "if" function that will check to see if the room to the left of the player's current room has a right exit (the information that is in the fourth index value for that room in the GAME_MAP list).

If the room to the left of the player has a right exit, then Lines 124 - 126 will run to change the floor_type in the middle of the wall to create an exit on the left wall of the player's current room (or the right wall of the room_to_left).

47. Press ENTER twice.

48. Type the code you see on Lines 128 – 131 of the screenshot below. Ensure your indentation matches what is shown in the screenshot.

```
111     # Add doorways.
112     middle_row = int(room_height / 2)
113     middle_column = int(room_width / 2)
114
115     if room_data[4]: # If exit at right of this room
116         room_map[middle_row][room_width - 1] = floor_type
117         room_map[middle_row+1][room_width - 1] = floor_type
118         room_map[middle_row-1][room_width - 1] = floor_type
119
120     if current_room % MAP_WIDTH != 1: # If room is not on left of map
121         room_to_left = GAME_MAP[current_room - 1]
122         # If room on the left has a right exit, add left exit in this room
123         if room_to_left[4]:
124             room_map[middle_row][0] = floor_type
125             room_map[middle_row + 1][0] = floor_type
126             room_map[middle_row - 1][0] = floor_type
127
128     if room_data[3]: # If exit at top of this room
129         room_map[0][middle_column] = floor_type
130         room_map[0][middle_column + 1] = floor_type
131         room_map[0][middle_column - 1] = floor_type
```

Line 128 will check to see if the current room the player is in has an exit at the top (the fourth item in the room's list of information, or the item with the index value of three).

If Line 128 is true, meaning the room has an exit on the top, Lines 129 - 131 will change the floor type of the three tiles in the middle of the room's top wall to allow for an exit to appear.

49. Press ENTER twice.

50. Type the code you see on Lines 133 – 139 of the screenshot below. Ensure your indentation matches what is shown in the screenshot.

```
111     # Add doorways.
112     middle_row = int(room_height / 2)
113     middle_column = int(room_width / 2)
114
115     if room_data[4]: # If exit at right of this room
116         room_map[middle_row][room_width - 1] = floor_type
117         room_map[middle_row+1][room_width - 1] = floor_type
118         room_map[middle_row-1][room_width - 1] = floor_type
119
120     if current_room % MAP_WIDTH != 1: # If room is not on left of map
121         room_to_left = GAME_MAP[current_room - 1]
122         # If room on the left has a right exit, add left exit in this room
123         if room_to_left[4]:
124             room_map[middle_row][0] = floor_type
125             room_map[middle_row + 1][0] = floor_type
126             room_map[middle_row - 1][0] = floor_type
127
128     if room_data[3]: # If exit at top of this room
129         room_map[0][middle_column] = floor_type
130         room_map[0][middle_column + 1] = floor_type
131         room_map[0][middle_column - 1] = floor_type
132
133     if current_room <= MAP_SIZE - MAP_WIDTH: # If room is not on bottom row
134         room_below = GAME_MAP[current_room+MAP_WIDTH]
135         # If room below has a top exit, add exit at bottom of this one
136         if room_below[3]:
137             room_map[room_height-1][middle_column] = floor_type
138             room_map[room_height-1][middle_column + 1] = floor_type
139             room_map[room_height-1][middle_column - 1] = floor_type
```

Line 133 will check to make sure that the current room the player is in is less than or equal to the MAP_SIZE minus the MAP_WIDTH. (In this case, the map size is 50 and the map width is 5. So, Line 133 will check to make sure that the current room the player is in has a number that is less than or equal to 45.) Rooms #46 - 50 are on the bottom row, so we will not need to worry about checking them for bottom exits in the next block of code.

Line 134 creates the room_below variable. This variable's value is equal to the value of the current_room number plus the value of the MAP_WIDTH. For example, if the player is in room number 30 and the MAP_WIDTH is 5, then the room_below variable would be equal to 35. This calculation will figure out which room is directly below the player on the map.

Line 135 contains a comment.

Line 136 will check to see if the room below the player's current room has a top exit (the fourth item in the room's data list, or the item with the index value of 3). If this is true, Lines 137 - 139 will run to add an exit to the bottom of the player's current room (or to the top of the room below the player).

51. Press ENTER twice.

52. Type the code you see on Lines 141 – 143 of the screenshot below.

```
133     if current_room <= MAP_SIZE - MAP_WIDTH: # If room is not on bottom row
134         room_below = GAME_MAP[current_room+MAP_WIDTH]
135         # If room below has a top exit, add exit at bottom of this one
136         if room_below[3]:
137             room_map[room_height-1][middle_column] = floor_type
138             room_map[room_height-1][middle_column + 1] = floor_type
139             room_map[room_height-1][middle_column - 1] = floor_type
140
141 ###############
142 ## EXPLORER  ##
143 ###############
```

Line 141 - 143 contains a comment indicating that the code in this section will deal with the Explorer functions.

53. Press ENTER twice.

54. Type the code you see on Lines 145 – 148 of the screenshot below. Ensure your indentation matches what is shown in the screenshot.

```
141 ###############
142 ## EXPLORER  ##
143 ###############
144
145 def draw():
146     global room_height, room_width, room_map
147     generate_map()
148     screen.clear()
```

Line 145 defines a new method called draw.

Line 146 establishes the room_height, room_width, and room)map variables as global variables. We learned before that designating variables as global variables will allow a function to access and modify the values of those variables, even if they weren't created within that particular function.

Line 147 will call (run) the generate_map function.

Line 148 will clear all components on the screen using the screen.clear() method.

55. Press ENTER twice.

56. Type the code you see on Lines 150 – 155 of the screenshot below. Ensure your indentation matches what is shown in the screenshot.

```
141 ###############
142 ## EXPLORER  ##
143 ###############
144
145 def draw():
146     global room_height, room_width, room_map
147     generate_map()
148     screen.clear()
149
150     for y in range(room_height):
151         for x in range(room_width):
152             image_to_draw = DEMO_OBJECTS[room_map[y][x]]
153             screen.blit(image_to_draw,
154                 (top_left_x + (x*30),
155                  top_left_y + (y*30) - image_to_draw.get_height()))
```

Line 150 will run for every y coordinate in the room_height list.

Line 151 contains a nested “for” function that will loop for every x coordinate in the room_width list. In other words, Lines 150 and 151 will loop through every time in the room_height and room_width lists.

Line 152 will create a new variable called image_to_draw. This variable’s value will be set to whatever is in the DEMO_OBJECTS list for that particular room (using the y and x coordinates).

Lines 153 - 155 will blit the demo objects to the screen at the locations specified using the height of the objects in the image_to_draw variable.

57. Press ENTER twice.

58. Type the code you see on Lines 157 – 159 of the screenshot below. Ensure your indentation matches what is shown in the screenshot.

```
145 def draw():
146     global room_height, room_width, room_map
147     generate_map()
148     screen.clear()
149 
150     for y in range(room_height):
151         for x in range(room_width):
152             image_to_draw = DEMO_OBJECTS[room_map[y][x]]
153             screen.blit(image_to_draw,
154                 (top_left_x + (x*30),
155                  top_left_y + (y*30) - image_to_draw.get_height()))
156 
157 def movement():
158     global current_room
159     old_room = current_room
```

Line 157 creates another method called movement.

The first thing this new method will do is to establish the current_room variable as a global variable, meaning that it can modify and reference that variable, even though it will be created outside of this method. (Line 158)

Line 159 creates a new variable called old_room and sets the variable's value to be equal to whatever current room the player is in (the value of the current_room variable).

59. Press ENTER twice.

60. Type the code you see on Lines 161 – 168 of the screenshot below. Ensure your indentation matches what is shown in the screenshot.

```
157 def movement():
158     global current_room
159     old_room = current_room
160 
161     if keyboard.left:
162         current_room -= 1
163     if keyboard.right:
164         current_room += 1
165     if keyboard.up:
166         current_room -= MAP_WIDTH
167     if keyboard.down:
168         current_room += MAP_WIDTH
```

Line 161 will check to see if the player has hit the left arrow key on the keyboard. If they have, Line 162 will run to subtract 1 from the player's current room number (in other words, moving left will move a player from room 50 to room 49, etc.).

Line 163 will check to see if the player has hit the right arrow key on the keyboard. If so, Line 164 will run to add 1 to the player's current room number.

Line 165 will check to see if the player has hit the up arrow key on their keyboard. If so, the program will subtract 1 from the MAP_WIDTH variable and set that value as the value for the current_room variable.

Line 167 will check to see if the player has hit the down arrow key on their keyboard. If so, the program will add one to the MAP_WIDTH variable and set that value as the value of the current_room variable.

61. Press ENTER twice.

62. Type the code you see on Lines 170 – 176 of the screenshot below. Ensure your indentation matches what is shown in the screenshot.

```
157 def movement():
158     global current_room
159     old_room = current_room
160 
161     if keyboard.left:
162         current_room -= 1
163     if keyboard.right:
164         current_room += 1
165     if keyboard.up:
166         current_room -= MAP_WIDTH
167     if keyboard.down:
168         current_room += MAP_WIDTH
169 
170     if current_room > 50:
171         current_room = 50
172     if current_room < 1:
173         current_room = 1
174 
175     if current_room != old_room:
176         print("Entering room:" + str(current_room))
```

Line 170 will check to see if the current_room variable is greater than 50. If so, the current_room variable will be reset back to 50. Remember, we only have 50 rooms. The player cannot be in a room with a number larger than 50.

Line 172 will check to see if the current_room variable is less than 1. Remember, other than our Room 0 that we use to store all of our undiscovered objects (which we do not want the player to be able to access), all of our rooms are labeled with the numbers 1 - 50. A player cannot be in a room with a number less than 1. If the current_room value is less than 1, then the variable's value will be reset to be equal to 1.

Line 175 will check to see if the current_room's value is not equal to the old_room's value. If this is true, then it will print a message telling the user what room they are entering.

63. Press ENTER twice.

64. Type the code you see on Line 178 of the screenshot below.

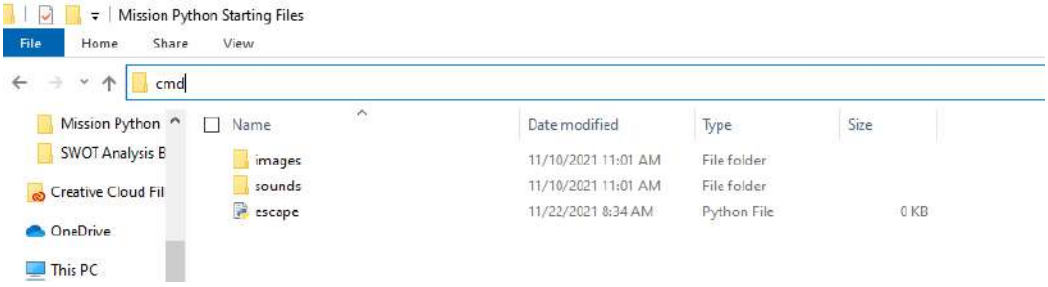
```
175     if current_room != old_room:
176         print("Entering room:" + str(current_room))
177 
178 clock.schedule_interval(movement, 0.08)
```

Line 178 will use the clock.schedule_interval method to run the movement function every 0.08 seconds automatically.

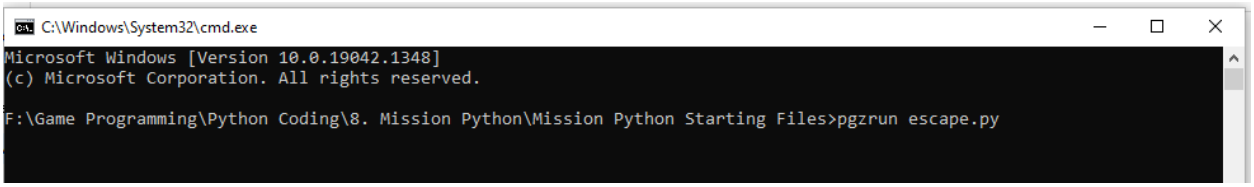
65. Go to File > Save to save your file.

66. Click the close button to close out of the game.

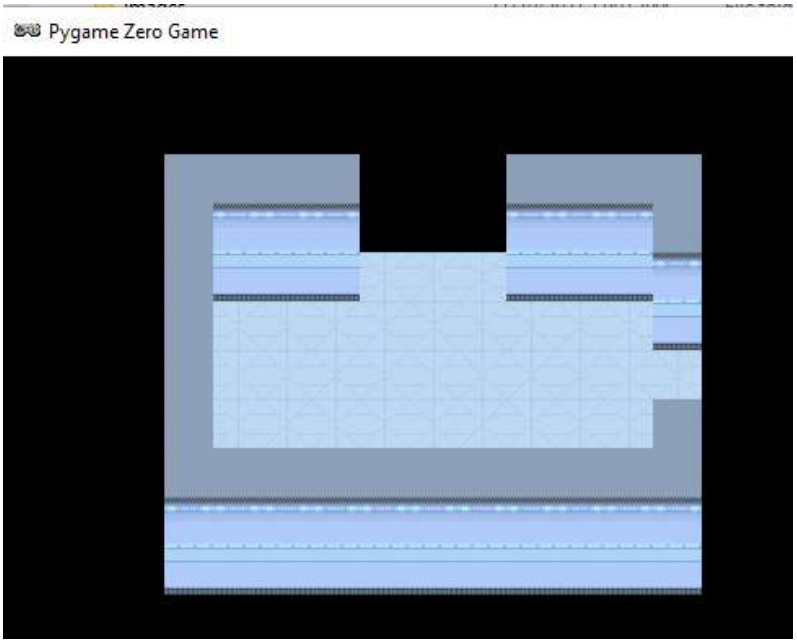
67. In order to preview the game, you will need to type “cmd” into the file path bar on the Windows Explorer window. Make sure you have your Mission Python Starting Files folder open in the Windows Explorer window before doing this.



68. When the command prompt opens up, type the command, “pgzrun escape.py” in the command prompt, then press ENTER.



69. Your game should load. The arrow keys on your keyboard should allow you to view the different rooms. We haven’t added objects to our map yet, so you should only see the floor, wall, and soil tiles.



Final Code:

```
# Escape

import time, random, math

###############
## VARIABLES ##
###############

WIDTH = 800 #window size
HEIGHT = 800

#PLAYER variables
PLAYER_NAME = "Alice"
FRIEND1_NAME = "Jace"
FRIEND2_NAME = "Matthew"
current_room = 31 # start room = 31

top_left_x = 190
top_left_y = 150

DEMO_OBJECTS = [images.floor, images.pillar, images.soil]


###############
##    MAP    ##
###############

MAP_WIDTH = 5
MAP_HEIGHT = 10
MAP_SIZE = MAP_WIDTH * MAP_HEIGHT

GAME_MAP = [ ["Room 0 - where unused objects are kept", 0, 0, False, False] ]

outdoor_rooms = range(1, 26)
for planetsectors in range(1, 26): #rooms 1 to 25 are generated here
    GAME_MAP.append( ["The dusty planet surface", 13, 13, True, True] )

GAME_MAP  += [
        #["Room name", height, width, Top exit?, Right exit?]
        ["The airlock", 13, 5, True, False], # room 26
        ["The engineering lab", 13, 13, False, False], # room 27
        ["Middle Mission Control", 9, 15, False, True], # room 28
        ["The viewing gallery", 9, 15, False, False], # room 29
        ["The crew's bathroom", 5, 5, False, False], # room 30
        ["The science entry bay", 7, 11, True, True], # room 31
        ["Left elbow room", 5, 7, True, False], # room 32
        ["Right elbow room", 7, 13, True, True], # room 33
        ["The science lab", 13, 13, False, True], # room 34
        ["The greenhouse", 13, 13, True, False], # room 35
        [PLAYER_NAME + "'s sleeping quarters", 5, 11, False, False], # room 36
        ["West corridor", 15, 5, True, True], # room 37
        ["The briefing room", 7, 13, False, True], # room 38
        ["The crew's community room", 13, 13, True, False], # room 39
        ["Main Mission Control", 14, 14, False, False], # room 40
        ["The sick bay", 12, 7, True, False], # room 41
        ["West corridor", 9, 7, True, False], # room 42
        ["Utilities control room", 9, 9, False, True], # room 43
        ["Systems engineering bay", 9, 11, False, False], # room 44
        ["Security portal to Mission Control", 7, 7, True, False], # room 45
        [FRIEND1_NAME + "'s sleeping quarters", 9, 11, True, True], # room 46
        [FRIEND2_NAME + "'s sleeping quarters", 9, 11, True, True], # room 47
        [FRIEND3_NAME + "'s sleeping quarters", 9, 11, True, True], # room 48
        ["The pipeworks", 13, 11, True, False], # room 49
        ["The chief scientist's office", 9, 7, True, True], # room 50
        ["The robot workshop", 9, 11, True, False] # room 51
        ]

#simple sanity check on map above to check data entry
assert len(GAME_MAP)-1 == MAP_SIZE, "Map size and GAME_MAP don't match"


###############
## MAKE MAP  ##
###############

def get_floor_type():
    if current_room in outdoor_rooms:
        return 2 # soil
    else:
        return 0 # tiled floor

def generate_map():
# This function makes the map for the current room,
# using room data, scenery data and prop data.
    global room_map, room_width, room_height, room_name, hazard_map
    global top_left_x, top_left_y, wall_transparency_frame
    room_data = GAME_MAP[current_room]
    room_name = room_data[0]
    room_height = room_data[1]
    room_width = room_data[2]

    floor_type = get_floor_type()
    if current_room in range(1, 21):
        bottom_edge = 2 #soil
        side_edge = 2 #soil
    if current_room in range(21, 26):
        bottom_edge = 1 #wall
        side_edge = 2 #soil
    if current_room > 25:
        bottom_edge = 1 #wall
        side_edge = 1 #wall

    # Create top line of room map.
    room_map=[[side_edge] * room_width]
    # Add middle lines of room map (wall, floor to fill width, wall).
    for y in range(room_height - 2):
        room_map.append([side_edge]
                        + [floor_type]*(room_width - 2) + [side_edge])
    # Add bottom line of room map.
    room_map.append([bottom_edge] * room_width)

    # Add doorways.
    middle_row = int(room_height / 2)
    middle_column = int(room_width / 2)

    if room_data[4]: # If exit at right of this room
        room_map[middle_row][room_width - 1] = floor_type
        room_map[middle_row+1][room_width - 1] = floor_type
        room_map[middle_row-1][room_width - 1] = floor_type
```

```
    if current_room % MAP_WIDTH != 1: # If room is not on left of map
        room_to_left = GAME_MAP[current_room - 1]
        # If room on the left has a right exit, add left exit in this room
        if room_to_left[4]:
            room_map[middle_row][0] = floor_type
            room_map[middle_row + 1][0] = floor_type
            room_map[middle_row - 1][0] = floor_type

    if room_data[3]: # If exit at top of this room
        room_map[0][middle_column] = floor_type
        room_map[0][middle_column + 1] = floor_type
        room_map[0][middle_column - 1] = floor_type

    if current_room <= MAP_SIZE - MAP_WIDTH: # If room is not on bottom row
        room_below = GAME_MAP[current_room+MAP_WIDTH]
        # If room below has a top exit, add exit at bottom of this one
        if room_below[3]:
            room_map[room_height-1][middle_column] = floor_type
            room_map[room_height-1][middle_column + 1] = floor_type
            room_map[room_height-1][middle_column - 1] = floor_type

###############
## EXPLORER  ##
###############

def draw():
    global room_height, room_width, room_map
    generate_map()
    screen.clear()

    for y in range(room_height):
        for x in range(room_width):
            image_to_draw = DEMO_OBJECTS[room_map[y][x]]
            screen.blit(image_to_draw,
                (top_left_x + (x*30),
                 top_left_y + (y*30) - image_to_draw.get_height()))

def movement():
    global current_room
    old_room = current_room

    if keyboard.left:
        current_room -= 1
    if keyboard.right:
        current_room += 1
    if keyboard.up:
        current_room -= MAP_WIDTH
    if keyboard.down:
        current_room += MAP_WIDTH

    if current_room > 50:
        current_room = 50
    if current_room < 1:
        current_room = 1

    if current_room != old_room:
        print("Entering room" + str(current_room))

clock.schedule_interval(movement, 0.08)
```