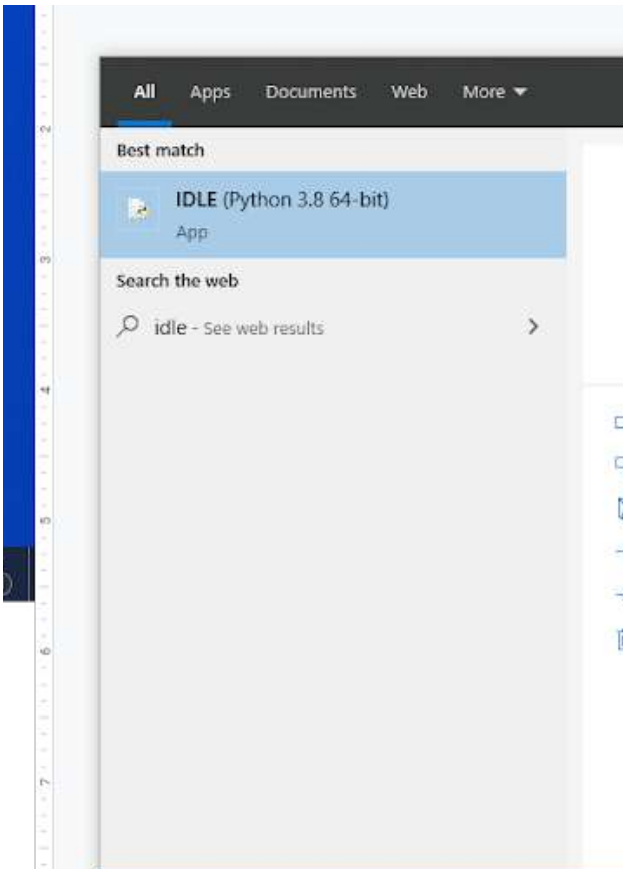
Part 5
Space Mission Directions

1. Navigate out to the Google Classroom for this class.

2. Locate the Space Mission Part 5 assignment.

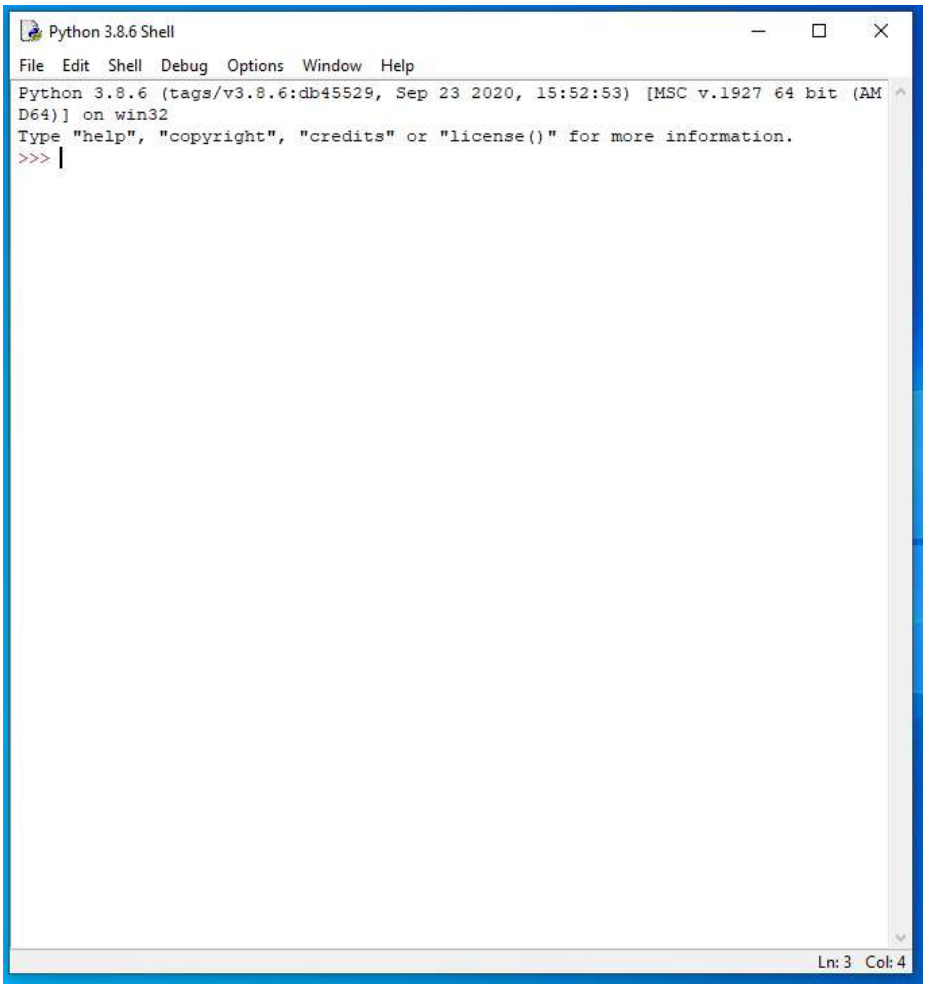
3. We are now ready to start adding code to our file. Using your Windows button menu, find and launch your IDLE program.



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.

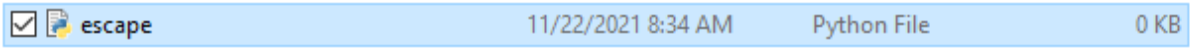
4. Your IDLE window should look something like this once it has launched.:



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.

5. Go to File > Open and then browse in the Starting Files folder I gave you to find the escape python file that we have been working on.



6. Your escape.py file will open up.

7. Click at the end of Line 54.

```
45      "down": [images.spacesuit_front, images.spacesuit_front_1,
46               images.spacesuit_front_2, images.spacesuit_front_3,
47               images.spacesuit_front_4
48               ]
49      }
50
51  player_direction = "down"
52  player_frame = 0
53  player_image = PLAYER[player_direction][player_frame]
54  player_offset_x, player_offset_y = 0, 0
55
56
57  ################
58  ##     MAP     ##
59  ################
```

8. Press ENTER twice.

9. Type the code you see on Lines 56 – 73 of the screenshot below. Ensure your indentation and capitalization match what is shown in the screenshot.

```
51 player_direction = "down"
52 player_frame = 0
53 player_image = PLAYER[player_direction][player_frame]
54 player_offset_x, player_offset_y = 0, 0
55
56 PLAYER_SHADOW = {
57     "left": [images.spacesuit_left_shadow, images.spacesuit_left_1_shadow,
58              images.spacesuit_left_2_shadow, images.spacesuit_left_3_shadow,
59              images.spacesuit_left_3_shadow
60              ],
61     "right": [images.spacesuit_right_shadow, images.spacesuit_right_1_shadow,
62               images.spacesuit_right_2_shadow,
63               images.spacesuit_right_3_shadow, images.spacesuit_right_3_shadow
64               ],
65     "up": [images.spacesuit_back_shadow, images.spacesuit_back_1_shadow,
66            images.spacesuit_back_2_shadow, images.spacesuit_back_3_shadow,
67            images.spacesuit_back_3_shadow
68            ],
69     "down": [images.spacesuit_front_shadow, images.spacesuit_front_1_shadow,
70              images.spacesuit_front_2_shadow, images.spacesuit_front_3_shadow,
71              images.spacesuit_front_3_shadow
72              ]
73     }
74
75
76 ###############
```

Line 56 will create another dictionary in your game called PLAYER_SHADOW. This is similar to the player dictionary. It contains animation frames for the astronaut's shadow on the floor. As the astronaut moves, the shadow also changes shape.

Lines 57 – 73 contains dictionary keys ("left", "right", "up", and "down") and the images for each shadow position that the animation will iterate through.

10. Press ENTER twice.

11. Type the code you see on Line 75 of the screenshot below.

```
56 PLAYER_SHADOW = {
57     "left": [images.spacesuit_left_shadow, images.spacesuit_left_1_shadow,
58              images.spacesuit_left_2_shadow, images.spacesuit_left_3_shadow,
59              images.spacesuit_left_3_shadow
60              ],
61     "right": [images.spacesuit_right_shadow, images.spacesuit_right_1_shadow,
62               images.spacesuit_right_2_shadow,
63               images.spacesuit_right_3_shadow, images.spacesuit_right_3_shadow
64               ],
65     "up": [images.spacesuit_back_shadow, images.spacesuit_back_1_shadow,
66            images.spacesuit_back_2_shadow, images.spacesuit_back_3_shadow,
67            images.spacesuit_back_3_shadow
68            ],
69     "down": [images.spacesuit_front_shadow, images.spacesuit_front_1_shadow,
70              images.spacesuit_front_2_shadow, images.spacesuit_front_3_shadow,
71              images.spacesuit_front_3_shadow
72              ]
73     }
74
75 player_image_shadow = PLAYER_SHADOW["down"][0]
76
77
78 ###############
79 ##    MAP    ##
80 ###############
```

Line 75 creates a new variable called player_image_shadow. Its default value is equal to the first item for the “down” entry in the PLAYER_SHADOW dictionary (the item with the index value of 0). This variable will store the astronaut’s current shadow, like the player_image variable that we created earlier store’s the astronaut’s current image.

12. Press ENTER twice.

13. Type the code you see on Lines 77 – 80 of the screenshot below. Ensure your punctuation, indentation, and capitalization match what is shown in the screenshot.

```
56 PLAYER_SHADOW = {
57     "left": [images.spacesuit_left_shadow, images.spacesuit_left_1_shadow,
58              images.spacesuit_left_2_shadow, images.spacesuit_left_3_shadow,
59              images.spacesuit_left_3_shadow
60              ],
61     "right": [images.spacesuit_right_shadow, images.spacesuit_right_1_shadow,
62               images.spacesuit_right_2_shadow,
63               images.spacesuit_right_3_shadow, images.spacesuit_right_3_shadow
64               ],
65     "up": [images.spacesuit_back_shadow, images.spacesuit_back_1_shadow,
66            images.spacesuit_back_2_shadow, images.spacesuit_back_3_shadow,
67            images.spacesuit_back_3_shadow
68            ],
69     "down": [images.spacesuit_front_shadow, images.spacesuit_front_1_shadow,
70              images.spacesuit_front_2_shadow, images.spacesuit_front_3_shadow,
71              images.spacesuit_front_3_shadow
72              ]
73     }
74
75 player_image_shadow = PLAYER_SHADOW["down"][0]
76
77 PILLARS = [
78     images.pillar, images.pillar_95, images.pillar_80,
79     images.pillar_60, images.pillar_50
80     ]
81
82
83 ################
84 ##    MAP     ##
85 ################
```

Later in the chapter, we will add animation that fades out the front wall when you walk behind it so you can still see the astronaut. Line 77 creates the PILLARS list that lists different image frames the animation can move through (Lines 78 – 79) when we do this.

14. Press ENTER twice.

15. Type the code you see on Line 82 of the screenshot below.

```
51 player_direction = "down"
52 player_frame = 0
53 player_image = PLAYER[player_direction][player_frame]
54 player_offset_x, player_offset_y = 0, 0
55
56 PLAYER_SHADOW = {
57     "left": [images.spacesuit_left_shadow, images.spacesuit_left_1_shadow,
58              images.spacesuit_left_2_shadow, images.spacesuit_left_3_shadow,
59              images.spacesuit_left_3_shadow
60              ],
61     "right": [images.spacesuit_right_shadow, images.spacesuit_right_1_shadow,
62               images.spacesuit_right_2_shadow,
63               images.spacesuit_right_3_shadow, images.spacesuit_right_3_shadow
64               ],
65     "up": [images.spacesuit_back_shadow, images.spacesuit_back_1_shadow,
66            images.spacesuit_back_2_shadow, images.spacesuit_back_3_shadow,
67            images.spacesuit_back_3_shadow
68            ],
69     "down": [images.spacesuit_front_shadow, images.spacesuit_front_1_shadow,
70              images.spacesuit_front_2_shadow, images.spacesuit_front_3_shadow,
71              images.spacesuit_front_3_shadow
72              ]
73     }
74
75 player_image_shadow = PLAYER_SHADOW["down"][0]
76
77 PILLARS = [
78     images.pillar, images.pillar_95, images.pillar_80,
79     images.pillar_60, images.pillar_50
80     ]
81
82 wall_transparency_frame = 0
83
84
85 ################
86 ##    MAP     ##
87 ################
```

Line 82 creates a new variable called wall_transparency_frame. This variable will remember the PILLARS image frame that is currently being displayed. It is set to the index value of 0, which will be the first item in the PILLARS list (in this case, the images.pillar).

16. Press ENTER twice.

17. Type the code you see on Lines 84 – 89 of the screenshot below.

```
77 PILLARS = [
78     images.pillar, images.pillar_95, images.pillar_80,
79     images.pillar_60, images.pillar_50
80     ]
81
82 wall_transparency_frame = 0
83
84 BLACK = (0, 0, 0)
85 BLUE = (0, 155, 255)
86 YELLOW = (255, 255, 0)
87 WHITE = (255, 255, 255)
88 GREEN = (0, 255, 0)
89 RED = (128, 0, 0)
90
91
92 ###############
93 ##    MAP    ##
94 ###############
```

Lines 84 – 89 will set up different variables named BLACK, BLUE, YELLOW, WHITE, GREEN, and RED. The numbers after the variables indicate the RGB color values for each variable. This sets the specific shade of each color you would like to use in your game.

Colors in Pygame are stored as tuples. A tuple is like a list whose content you can't change, and it uses parentheses instead of square brackets. You've seen tuples used for coordinates when drawing on the screen. Colors are stored as three numbers that specify the amount of red, green, and blue in the color, in that order. The scale for each color ranges from 0 to 255.

18. Ensure that your "MAP" comment runs on Lines 92 – 94 of your code. You may have to add or delete blank lines to make this happen.

19. Scroll and click at the end of Line 455.

```
443     if current_room in scenery:
444         for this_scenery in scenery[current_room]:
445             scenery_number = this_scenery[0]
446             scenery_y = this_scenery[1]
447             scenery_x = this_scenery[2]
448             room_map[scenery_y][scenery_x] = scenery_number
449
450             image_here = objects[scenery_number][0]
451             image_width = image_here.get_width()
452             image_width_in_tiles = int(image_width / TILE_SIZE)
453
454             for tile_number in range(1, image_width_in_tiles):
455                 room_map[scenery_y][scenery_x + tile_number] = 255
456
457 ###############
458 ## GAME LOOP ##
459 ###############
```

20. Press ENTER twice.

21. Type the code you see on Lines 457 – 462 of the screenshot below. Ensure your indentation matches what is shown in the screenshot.

```
443     if current_room in scenery:
444         for this_scenery in scenery[current_room]:
445             scenery_number = this_scenery[0]
446             scenery_y = this_scenery[1]
447             scenery_x = this_scenery[2]
448             room_map[scenery_y][scenery_x] = scenery_number
449 
450             image_here = objects[scenery_number][0]
451             image_width = image_here.get_width()
452             image_width_in_tiles = int(image_width / TILE_SIZE)
453 
454             for tile_number in range(1, image_width_in_tiles):
455                 room_map[scenery_y][scenery_x + tile_number] = 255
456 
457     center_y = int(HEIGHT / 2) # Center of game window
458     center_x = int(WIDTH / 2)
459     room_pixel_width = room_width * TILE_SIZE # Size of room in pixels
460     room_pixel_height = room_height * TILE_SIZE
461     top_left_x = center_x - 0.5 * room_pixel_width
462     top_left_y = (center_y - 0.5 * room_pixel_height) + 110
463 
464 
465 ###############
466 ## GAME LOOP ##
467 ###############
```

We now need to position the room on the screen.

The code you just typed starts by working out where the middle of the window in a room is. The HEIGHT and WIDTH variables store the window's size in pixels. Dividing them by 2 gives us the coordinates of the center of the window. We store these coordinates in the center_y and center_x variables (Lines 457 – 458).

This program also works out how wide or tall the image of the room is in pixels (Lines 459 – 460). This will be the width or height of the room in tiles multiplied by the size of a tile. The result is stored in the room_pixel_width and room_pixel_height variables.

To put the room image in the middle of the room, we want half the room to be to the left of the center line and half to the right. So, we subtract half the room width in pixels from the center line and start drawing the room there (Line 461). A similar calculation is made for the top_left_y variable, except we add 110 to the result because our final screen layout will use an area at the top of the screen as an information panel (Line 462).

22. Click at the end of Line 467.

```
457     center_y = int(HEIGHT / 2) # Center of game window
458     center_x = int(WIDTH / 2)
459     room_pixel_width = room_width * TILE_SIZE # Size of room in pixels
460     room_pixel_height = room_height * TILE_SIZE
461     top_left_x = center_x - 0.5 * room_pixel_width
462     top_left_y = (center_y - 0.5 * room_pixel_height) + 110
463
464
465 ###############
466 ## GAME LOOP ##
467 ###############
```

23. Press ENTER twice.

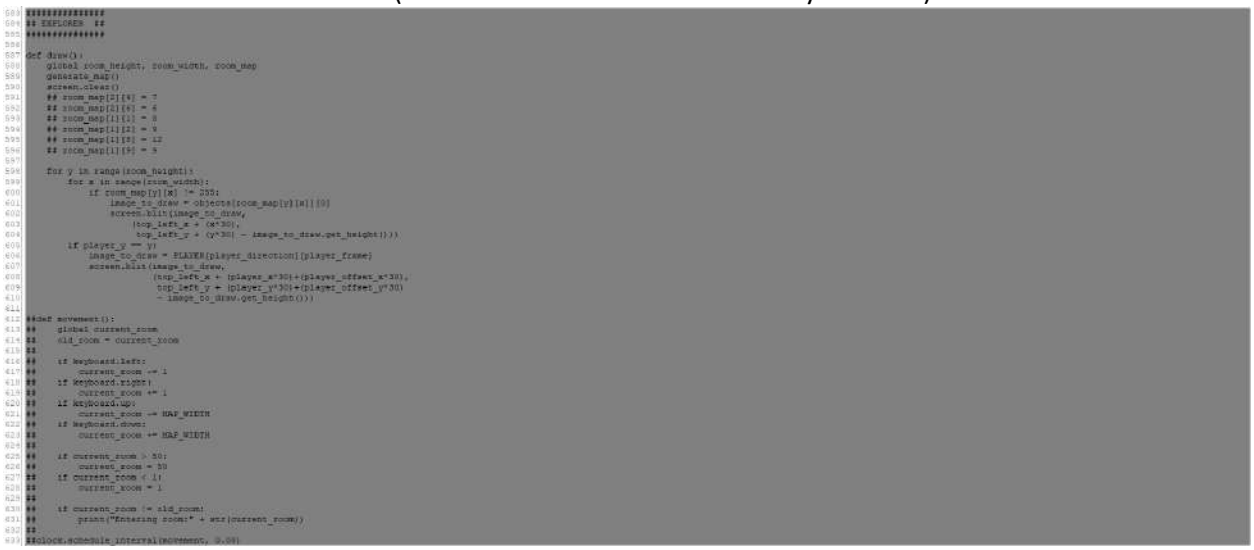
24. Type the code you see on Lines 469 – 470 of the screenshot below. Ensure your indentation matches what is shown in the screenshot.

```
465 ################
466 ## GAME LOOP ##
467 ################
468
469 def start_room():
470     show_text("You are here: " + room_name, 0)
471
472 def game_loop():
473     global player_x, player_y, current_room
474     global from_player_x, from_player_y
475     global player_image, player_image_shadow
476     global selected_item, item_carrying, energy
477     global player_offset_x, player_offset_y
478     global player_frame, player_direction
```

Line 469 creates a new function called start_room in the GAME LOOP section of your code.

Line 470 will run the show_text method to display the text, “You are here: “ along with the room_name, taken from the GAME_MAP list. The 0 at the end of the line of code indicates the line number. We haven’t created the show_text method in our code yet, so this will make more sense later.

25. Scroll and select Lines 583 – 633 (the entire “EXPLORER” section of your code).



26. Press BACKSPACE to delete this entire section. We won't need it anymore.

```
572     if player_direction == "right" and player_frame > 0:
573         player_offset_x = -1 + (0.25 * player_frame)
574     if player_direction == "left" and player_frame > 0:
575         player_offset_x = 1 - (0.25 * player_frame)
576     if player_direction == "up" and player_frame > 0:
577         player_offset_y = 1 - (0.25 * player_frame)
578     if player_direction == "down" and player_frame > 0:
579         player_offset_y = -1 + (0.25 * player_frame)
580
581
582
583
584 ###############
585 ##   START   ##
586 ###############
587
588 generate_map()
589 clock.schedule_interval(game_loop, 0.03)
```

27. Type the code you see on Lines 583 – 592 of the screenshot below. Ensure your indentation matches what is shown in the screenshot below.

```
583 ###############
584 ##  DISPLAY  ##
585 ###############
586
587 def draw_image(image, y, x):
588     screen.blit(
589         image,
590         (top_left_x + (x * TILE_SIZE),
591          top_left_y + (y * TILE_SIZE) - image.get_height())
592         )
593 ###############
594 ##   START   ##
595 ###############
596
597 generate_map()
598 clock.schedule_interval(game_loop, 0.03)
```

Lines 583 – 585 create a new section of code, called DISPLAU.

Line 587 creates a new method called draw_image. This method will require the image we want to draw and the and the y and x tile positions of the object in the room whenever it is called. The function will work out where on the screen to draw the image (the pixel position) based on the tile position in the room.

Line 588 - 592 will utilize the screen.blit command to blit the image at the location specified.

28. Press ENTER twice.

29. Type the code you see on Lines 594 – 599 of the screenshot below. Ensure your indentation matches what is shown in the screenshot.

```
583 ###############
584 ##  DISPLAY  ##
585 ###############
586
587 def draw_image(image, y, x):
588     screen.blit(
589         image,
590         (top_left_x + (x * TILE_SIZE),
591          top_left_y + (y * TILE_SIZE) - image.get_height())
592         )
593
594 def draw_shadow(image, y, x):
595     screen.blit(
596         image,
597         (top_left_x + (x * TILE_SIZE),
598          top_left_y + (y * TILE_SIZE))
599         )
600 ###############
601 ##   START   ##
602 ###############
```

Line 594 creates a new function called draw_shadow. This function will also require the image you want to use and the x and y tile positions when it is called.

Lines 595 – 599 will utilize the screen.blit command to blit the image at the location specified.

The draw_shadow() function is very similar to the function for drawing an image, except that the image's height is not subtracted when calculating its onscreen position. This is what places the shadow below the main image.

30. Press ENTER twice.

31. Type the code you see on Lines 601 – 607 of the screenshot below. Ensure your indentation matches what is shown in the screenshot.

```
583 ################
584 ##  DISPLAY  ##
585 ################
586
587 def draw_image(image, y, x):
588     screen.blit(
589         image,
590         (top_left_x + (x * TILE_SIZE),
591          top_left_y + (y * TILE_SIZE) - image.get_height())
592         )
593
594 def draw_shadow(image, y, x):
595     screen.blit(
596         image,
597         (top_left_x + (x * TILE_SIZE),
598          top_left_y + (y * TILE_SIZE))
599         )
600
601 def draw_player():
602     player_image = PLAYER[player_direction][player_frame]
603     draw_image(player_image, player_y + player_offset_y,
604                player_x + player_offset_x)
605     player_image_shadow = PLAYER_SHADOW[player_direction][player_frame]
606     draw_shadow(player_image_shadow, player_y + player_offset_y,
607                 player_x + player_offset_x)
```

Line 601 creates another new function called draw_player. This function will draw the astronaut.

First, it puts the correct animation frame into the player_image variable (Line 602).

Lines 603 - 604 then uses the draw_image function that we created on Line 587 to draw the astronaut's image. The draw_image function requires the following arguments:

- The variable player_image, which contains the image to draw.
- The result after adding the global variables for player_y and player_offset_y. This is the y position in tiles, which might include a decimal part.
- The result after adding player_x and player_offset_x for the x position in tiles.

Lines 605 – 607 use a similar code to draw the player's shadow: the correct animation frame from the PLAYER_SHADOW dictionary is put into the player_image_shadow variable. Then, the draw_shadow function is used to draw it. The draw_shadow function uses the same tile positions as the draw_image function.

32. Press ENTER twice.

33. Type the code you see on Lines 609 – 625 of the screenshot below. Ensure your indentation, punctuation, and line spacing matches what is shown in the screenshot.

```
601 def draw_player():
602     player_image = PLAYER[player_direction][player_frame]
603     draw_image(player_image, player_y + player_offset_y,
604                player_x + player_offset_x)
605     player_image_shadow = PLAYER_SHADOW[player_direction][player_frame]
606     draw_shadow(player_image_shadow, player_y + player_offset_y,
607                 player_x + player_offset_x)
608
609 def draw():
610     if game_over:
611         return
612
613     # Clear the game arena area.
614     box = Rect((0, 150), (800, 600))
615     screen.draw.filled_rect(box, RED)
616     box = Rect ((0, 0), (800, top_left_y + (room_height - 1)*30))
617     screen.surface.set_clip(box)
618     floor_type = get_floor_type()
619
620     for y in range(room_height): # Lay down floor tiles, then items on floor.
621         for x in range(room_width):
622             draw_image(objects[floor_type][0], y, x)
623             # Next line enables shadows to fall on top of objects on floor
624             if room_map[y][x] in items_player_may_stand_on:
625                 draw_image(objects[room_map[y][x]][0], y, x)
626 ###############
627 ##   START   ##
628 ###############
```

Line 609 creates a new function called draw.

Line 610 will check to see if the game_over variable is equal to True. If so, then Line 611 contains a return statement that will exit out of this particular block of code and skip down to the next function. This block of code doesn't need to run if the game is over.

Line 613 contains a comment.

Line 614 begins the process of clearing the game arena, where the space station will be drawn. It does this by drawing a big red rectangle, wiping out the previous screen display. The areas as the top and the bottom that give the player information are separate, so they are not changed.

There are two steps for putting a rectangle on the screen. First, you create the shape using a Pygame object called Rect. On Line 614, we create a rect object called box. The object is placed at the x and y coordinate location of 0 and 150 and is 800 pixels wide by 600 pixels tall.

Line 615 will draw the filled rect object named box on the screen, and will fill the rectangle with the color numbers saved in the RED variable. These numbers are the RGB color values we specified earlier.

Line 616 creates another rect object called box. This object is created after the previous rect object is drawn so that we overwrite the coordinates of the first rect object with the coordinates of the second. The second rect box object will have an x and y location of 0, 0. It will be 800 pixels wide and whatever height is appropriate for the current room the player is in.

You can also use the Rect shape to create a clipping area. This is like an invisible window through which you view the screen. If the program draws something outside the window, it can't be seen. Line 617 sets up a clipping area that's the height of the room to stop the player's shadow from spilling out of the bottom of the game when they're in the front doorway.

Line 618 creates a variable called floor_type. This line will run the get_floor_type() method and store the result in the floor_type variable on Line 618.

The room is drawn in two stages. First, the program draws the floor tiles and anything that the player can walk on. Drawing them first enables scenery, the player, and shadows to be drawn on top of them. This solves the problem of black holes appearing under scenery, because there will be floor tiles in those spaces before the scenery is drawn.

Lines 620 – 621 create "for" loops that will loop for each y coordinates and x coordinate in the room.

Line 622 executes the draw_image function that will access the objects list based on the floor_type of the room that you calculated on Line 618. This will draw the appropriate floor type image on the screen at the appropriate y and x location (whatever location is currently being iterated through in the loop).

Line 623 contains a comment.

Line 624 contains an if function that will check to see if that particular y and x coordinate in the room_map contains items that the player is allowed to stand on.

If this is true, then Line 625 executes the draw_image function again to access the objects list for that particular y and x coordinate and returns the first item in the list, which is the image of that particular object. It will generate the item at the appropriate y and x position.

34. Press ENTER twice.

35. Type the code you see on Lines 627 – 650 of the screenshot below. Ensure your indentation, punctuation, and line spacing match what is shown in the screenshot.

```
620     for y in range(room_height): # Lay down floor tiles, then items on floor.
621         for x in range(room_width):
622             draw_image(objects[floor_type][0], y, x)
623             # Next line enables shadows to fall on top of objects on floor
624             if room_map[y][x] in items_player_may_stand_on:
625                 draw_image(objects[room_map[y][x]][0], y, x)
626
627      # Pressure pad in room 26 is added here, so props can go on top of it.
628     if current_room == 26:
629         draw_image(objects[39][0], 8, 2)
630         image_on_pad = room_map[8][2]
631         if image_on_pad > 0:
632             draw_image(objects[image_on_pad][0], 8, 2)
633
634     for y in range(room_height):
635         for x in range(room_width):
636             item_here = room_map[y][x]
637             # Player cannot walk on 255: it marks spaces used by wide objects.
638             if item_here not in items_player_may_stand_on + [255]:
639                 image = objects[item_here][0]
640
641                 if (current_room in outdoor_rooms
642                     and y == room_height - 1
643                     and room_map[y][x] == 1) or \
644                     (current_room not in outdoor_rooms
645                     and y == room_height - 1
646                     and room_map[y][x] == 1
647                     and x > 0
648                     and x < room_width - 1):
649                     # Add transparent wall image in the front row.
650                     image = PILLARS[wall_transparency_frame]
651
652 ###############
653 ##   START   ##
654 ###############
```

Line 627 contains a comment.

Line 628 contains an if function that checks to see if the value of the current_room variable is equal to room 26.

If it is, Lines 629 – 632 will execute.

Line 629 will execute the draw_image function to draw the floor pad image (item #39 in the objects list, the first item in the list is the image associated with the floor_pad). 8 is the y tile location for the image and 2 is the x tile location.

Line 630 creates a variable called image_on_map and sets it equal to be the y and x tile coordinate of 8 and 2 on the room_map.

Line 631 checks to see if the image_on_pad variable is greater than 0. If it is, it will draw the floor pad image at the proper tile location (Line 632).

Line 634 begins the second stage of drawing the room. The program will add scenery in the room, including shadows, using new loops that begin on Line 634. Because these loops come after the floor for the whole room has been drawn, the shadows will be drawn on top of the floor tiles and items on the floor. The shadows are transparent, so you can still see the object underneath the shadow.

Beginning on Line 641, the program begins drawing the transparent front wall.

When the program is drawing the front row of the room (when the y loop equals room_height – 1), it checks whether it needs to draw a semitransparent wall instead of the solid wall object taken from the room map. The semitransparent wall is used if the player is standing behind it.

On the planet surface, the program makes the whole wall transparent. Inside the space station, a transparent wall panel is used only if its not in one of the bottom former positions. The corners always use a solid wall panel. The reason is that it loops odd if you see the solid edge wall start in the second frow from the bottom.

Later one, we will add the code to animate the transparency on the wall by changing the number in wall_transparency_frame. You won't see the semi-transparent wall yet in the game.

36. Press ENTER twice.

37. Type the code you see on Lines 652 – 669 of the screenshot below. Ensure your indentation, punctuation, and line spacing match what is shown in the screenshot.

```
641                 if (current_room in outdoor_rooms
642                     and y == room_height - 1
643                     and room_map[y][x] == 1) or \
644                     (current_room not in outdoor_rooms
645                     and y == room_height - 1
646                     and room_map[y][x] == 1
647                     and x > 0
648                     and x < room_width - 1):
649                     # Add transparent wall image in the front row.
650                     image = PILLARS[wall_transparency_frame]
651
652                 draw_image(image, y, x)
653
654                 if objects[item_here][1] is not None: # If object has a shadow
655                     shadow_image = objects[item_here][1]
656                     # if shadow might need horizontal tiling
657                     if shadow_image in [images.half_shadow,
658                                         images.full_shadow]:
659                         shadow_width = int(image.get_width() / TILE_SIZE)
660                         # Use shadow across width of object.
661                         for z in range(0, shadow_width):
662                             draw_shadow(shadow_image, y, x+z)
663                     else:
664                         draw_shadow(shadow_image, y, x)
665
666         if (player_y == y):
667                 draw_player()
668
669     screen.surface.set_clip(None)
670 ###############
671 ##   START   ##
672 ###############
```

Line 652 draws the player on top of the floor.

Line 654 will check to see if the current object being drawn has a shadow.

If it does, Line 655 creates a variable called shadow_image and sets its value to be equal to the second item in the objects list for the particular item being drawn.

Line 656 contains a comment.

Lines 657 – 658 check to see if the object has a half_shadow or full_shadow, which would fill half a tile or a whole tile, respectively. These two standard shadows are used with block items (like electrical units and walls) that don't need a distinctive shadow outline. The program checks whether the shadow_image is in a list that contains those two standard images.

If the shadow is one of the standard images, the program then works out how wide the shadow should be in tiles. That is calculated by taking the width of the object casting the shadow and dividing it by the width of a tile (30 pixels).

The program then creates a loop to draw the standard shadow images, using the variable z. It starts at 0 and runs until the width of the shadow minus 1. That's because a range leaves out the last item. The z values are added to the x position from the main loop and are used to draw the shadow tiles.

If the shadow image is not one of the standard shadow images, the program will execute the draw_shadow function using the shadow_image from the objects dictionary and the current u and x tile location.

Line 666 will check to see if the player_y coordinate is equal to the y location that is currently being iterated through in the "for" loop that starts on Line 634. If this is true, the draw_player() function will execute (Line 667) to draw the player image. By drawing the player image after all the floor tiles and scenery, the player image will be placed on top.

Line 669 turns off the clipping area that was set earlier.

38. Press ENTER twice.

39. Type the code you see on Lines 671 – 677 of the screenshot below. Ensure your indentation and line spacing matches what is shown in the screenshot below.

```
663                     else:
664                         draw_shadow(shadow_image, y, x)
665
666         if (player_y == y):
667                 draw_player()
668
669     screen.surface.set_clip(None)
670
671 def adjust_wall_transparency():
672     global wall_transparency_frame
673
674     if (player_y == room_height - 2
675         and room_map[room_height - 1][player_x] == 1
676         and wall_transparency_frame < 4):
677         wall_transparency_frame += 1 # Fade wall out.
678 ################
679 ##   START   ##
680 ################
```

Line 671 creates another function called adjust_wall_transparency.

Line 672 converts the wall_transparency_frame to a global variable so that this function is able to access and modify the value of the wall_transparency_frame variable.

If the player is standing behind the wall, the flowing statements are true:

- Their u position will be equal to room_height-2.
- There is a piece of wall in the bottom row of the room that is in line with the player's x position.

If the player is behind the wall (Lines 674 – 675) and the wall transparency is not set to maximum (Line 676), the wall transparency is increased by 1, making the wall more transparent (Line 677)

40. Press ENTER twice.

41. Type the code you see on Lines 679 – 682 of the screenshot below. Ensure your indentation matches what is shown in the screenshot.

```
669     screen.surface.set_clip(None)
670
671 def adjust_wall_transparency():
672     global wall_transparency_frame
673
674     if (player_y == room_height - 2
675         and room_map[room_height - 1][player_x] == 1
676         and wall_transparency_frame < 4):
677         wall_transparency_frame += 1 # Fade wall out.
678
679     if ((player_y < room_height - 2
680             or room_map[room_height - 1][player_x] != 1)
681             and wall_transparency_frame > 0):
682         wall_transparency_frame -= 1 # Fade wall in.
683 ###############
684 ##   START   ##
685 ###############
```

If either of the following is true, it means the player isn't hidden by the wall:

- Their y position is less than room_height – 2. The player can be seen, at least in part, if they're farther back in the room.
- There is not a piece of wall in the bottom row of the room in line with their x position.

In these cases, if the wall transparency is set to more than the minimum, it's reduced by one.

42. Press ENTER twice.

43. Type the code you see on Lines 684 – 691 of the screenshot below. Ensure your indentation, punctuation, and line spacing match what is shown in the screenshot.

```
671 def adjust_wall_transparency():
672     global wall_transparency_frame
673
674     if (player_y == room_height - 2
675         and room_map[room_height - 1][player_x] == 1
676         and wall_transparency_frame < 4):
677         wall_transparency_frame += 1 # Fade wall out.
678
679     if ((player_y < room_height - 2
680             or room_map[room_height - 1][player_x] != 1)
681             and wall_transparency_frame > 0):
682         wall_transparency_frame -= 1 # Fade wall in.
683
684 def show_text(text_to_show, line_number):
685     if game_over:
686         return
687     text_lines = [15, 50]
688     box = Rect((0, text_lines[line_number]), (800, 35))
689     screen.draw.filled_rect(box, BLACK)
690     screen.draw.text(text_to_show,
691                      (20, text_lines[line_number]), color=GREEN)
692 ###############
693 ##   START   ##
694 ###############
```

Line 684 creates another method called show_text.

Line 685 checks to see if the game_over variable is set to True. If so, Line 686 contains a return statement to exit out of this block of code and skip down to the next function. This code doesn't need to execute if the game is over.

The line number will be either 0 for the top row or 1 for the second row, which is reserved for important messages. When the function is called, the message is put into the variable text_to_show and the row number goes into the line_number position.

We use a list called text_lines to remember the vertical positions (in pixels) of the two lines of text (Line 687).

Line 688 will define a box at the x-coordinate of 0 and the y-coordinate equal to the line_number specified when the function is called. The width of the rectangle will be 800 and the height is 35 pixels.

Line 689 fills the rectangle object with black to clear the row of text before the new message is drawn.

Finally, we use the screen.draw.text() function in Pygame to put the text on the screen. This function takes the text, the text's x and y position, and the text color. The position numbers go inside parentheses.

The x position is 20 pixels from the left and the vertical position is taken from the text_lines list, using the number in line_number as the list index.

44. Press ENTER three times. Ensure your "START" comment runs on Lines 695 – 697.

```
684 def show_text(text_to_show, line_number):
685     if game_over:
686         return
687     text_lines = [15, 50]
688     box = Rect((0, text_lines[line_number]), (800, 35))
689     screen.draw.filled_rect(box, BLACK)
690     screen.draw.text(text_to_show,
691                      (20, text_lines[line_number]), color=GREEN)
692
693
694
695 ################
696 ##   START    ##
697 ################
698
699 generate_map()
700 clock.schedule_interval(game_loop, 0.03)
```

45. Click at the end of Line 700.

```
695 ################
696 ##   START    ##
697 ################
698
699 generate_map()
700 clock.schedule_interval(game_loop, 0.03)
```

46. Press ENTER.

47. Type the code you see on Line 701 of the screenshot below.

```
695 ################
696 ##   START    ##
697 ################
698
699 generate_map()
700 clock.schedule_interval(game_loop, 0.03)
701 clock.schedule_interval(adjust_wall_transparency, 0.05)
```

Line 701 will run the adjust_wall_transparency function every 0.5 seconds. This makes the wall fade in or out as necessary as the player walks around the room.

48. Go to File > Save Now to save your code.

Final Code:

```
# Escape

import time, random, math

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

WIDTH = 800 #window size
HEIGHT = 800

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

top_left_x = 100
top_left_y = 150

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

LANDER_SECTOR = random.randint(1, 24)
LANDER_X = random.randint(2, 11)
LANDER_Y = random.randint(2, 11)

TILE_SIZE = 80

player_y, player_x = 2, 5
game_over = False

PLAYER = {
    "left": [images.spacesuit_left, images.spacesuit_left_1,
             images.spacesuit_left_2, images.spacesuit_left_3,
             images.spacesuit_left_4
             ],
    "right": [images.spacesuit_right, images.spacesuit_right_1,
              images.spacesuit_right_2, images.spacesuit_right_3,
              images.spacesuit_right_4
              ],
    "up": [images.spacesuit_back, images.spacesuit_back_1,
           images.spacesuit_back_2, images.spacesuit_back_3,
           images.spacesuit_back_4
           ],
    "down": [images.spacesuit_front, images.spacesuit_front_1,
             images.spacesuit_front_2, images.spacesuit_front_3,
             images.spacesuit_front_4
             ]
    }

player_direction = "down"
player_frame = 0
player_image = PLAYER[player_direction][player_frame]
player_offset_x, player_offset_y = 0, 0

PLAYER_SHADOW = {
    "left": [images.spacesuit_left_shadow, images.spacesuit_left_1_shadow,
             images.spacesuit_left_2_shadow, images.spacesuit_left_3_shadow,
             images.spacesuit_left_3_shadow
             ],
    "right": [images.spacesuit_right_shadow, images.spacesuit_right_1_shadow,
              images.spacesuit_right_2_shadow,
              images.spacesuit_right_3_shadow, images.spacesuit_right_3_shadow
              ],
    "up": [images.spacesuit_back_shadow, images.spacesuit_back_1_shadow,
           images.spacesuit_back_2_shadow, images.spacesuit_back_3_shadow,
           images.spacesuit_back_3_shadow
           ],
    "down": [images.spacesuit_front_shadow, images.spacesuit_front_1_shadow,
             images.spacesuit_front_2_shadow, images.spacesuit_front_3_shadow,
             images.spacesuit_front_3_shadow
             ]
    }

player_image_shadow = PLAYER_SHADOW["down"][0]

PILLARS = {
    images.pillar, images.pillar_95, images.pillar_80,
    images.pillar_60, images.pillar_50
    }

wall_transparency_frame = 0

BLACK = (0, 0, 0)
BLUE = (0, 155, 255)
YELLOW = (255, 255, 0)
WHITE = (255, 255, 255)
GREEN = (0, 255, 0)
RED = (128, 0, 0)


###############
##    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
        ["Poodle Mission Control", 9, 13, False, True], # room 28
        ["The viewing gallery", 9, 15, False, False], # room 29
        ["The crew's bathroom", 5, 5, False, False], # room 30
        ["The airlock entry bay", 7, 11, True, True], # room 31
        ["Left elbow room", 9, 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", 9, 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", 11, 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
        ["The pipeworks", 13, 11, True, False], # room 48
        ["The chief scientist's office", 9, 7, True, True], # room 49
        ["The robot workshop", 9, 11, True, False] # room 50
        ]

#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"


###############
##  OBJECTS  ##
###############

objects = [
    0: [images.floor, None, "The floor is shiny and clean"],
    1: [images.pillar, images.full_shadow, "The wall is smooth and cold"],
    2: [images.soil, None, "It's like a desert. Or should that be dessert?"],
    3: [images.pillar_low, images.half_shadow, "The wall is smooth and cold"],
    4: [images.bed, images.half_shadow, "A tidy and comfortable bed"],
    5: [images.table, images.half_shadow, "It's made from strong plastic."],
    6: [images.chair_left, None, "A chair with a soft cushion"],
    7: [images.chair_right, None, "A chair with a soft cushion"],
    8: [images.bookcase_tall, images.full_shadow,
        "Bookshelves, stacked with reference books"],
    9: [images.bookcase_small, images.half_shadow,
        "Bookshelves, stacked with reference books"],
    10: [images.cabinet, images.half_shadow,
         "A small locker, for storing personal items"],
    11: [images.desk_computer, images.half_shadow,
         "A computer. Use it to run life support diagnostics"],
    12: [images.plant, images.plant_shadow, "A spaceberry plant, grown here"],
    13: [images.electrical1, images.half_shadow,
         "Electrical systems used for powering the space station"],
    14: [images.electrical2, images.half_shadow,
         "Electrical systems used for powering the space station"],
    15: [images.cactus, images.cactus_shadow, "Ouch! Careful on the cactus!"],
    16: [images.shrub, images.shrub_shadow,
         "A space lettuce. A bit limp, but amazing it's growing here!"],
    17: [images.pipe1, images.pipe1_shadow, "Water purification pipes"],
    18: [images.pipe2, images.pipe2_shadow,
         "Pipes for the life support systems"],
    19: [images.pipe3, images.pipe3_shadow,
         "Pipes for the life support systems"],
    20: [images.door, images.door_shadow, "Safety door. Opens automatically \
for astronauts in functioning spacesuits."],
    21: [images.door, images.door_shadow, "The airlock door. \
For safety reasons, it requires TWO person operation."],
    22: [images.door, images.door_shadow, "A locked door. It needs " \
         + PLAYER_NAME + "'s access card"],
    23: [images.door, images.door_shadow, "A locked door. It needs " \
         + FRIEND1_NAME + "'s access card"],
    24: [images.door, images.door_shadow, "A locked door. It needs " \
         + FRIEND2_NAME + "'s access card"],
    25: [images.door, images.door_shadow,
         "A locked door. It is opened from Main Mission Control"],
    26: [images.door, images.door_shadow,
         "A locked door in the engineering bay."],
    27: [images.map, images.full_shadow,
         "The screen says the crash site was Sector: " \
         + str(LANDER_SECTOR) + " // X: " + str(LANDER_X) + \
         " // Y: " + str(LANDER_Y)],
    28: [images.rock_large, images.rock_large_shadow,
         "A rock. Its coarse surface feels like a whetstone", "the rock"],
    29: [images.rock_small, images.rock_small_shadow,
         "A small but heavy piece of Martian rock"],
    30: [images.crater, None, "A crater in the planet surface"],
    31: [images.fence, None,
         "A fine gauze fence. It helps protect the station from dust storms"],
    32: [images.contraption, images.contraption_shadow,
         "One of the scientific experiments. It gently vibrates"],
    33: [images.robot_arm, images.robot_arm_shadow,
         "A robot arm, used for heavy lifting"],
    34: [images.toilet, images.half_shadow, "A sparkling clean toilet"],
    35: [images.sink, None, "A sink with running water", "the taps"],
    36: [images.globe, images.globe_shadow,
         "A giant globe of the planet. It gently glows from inside"],
    37: [images.science_lab_table, None,
         "A table of experiments, analyzing the planet soil and dust"],
    38: [images.vending_machine, images.full_shadow,
         "A vending machine. It requires a credit.", "the vending machine"],
    39: [images.floor_pad, None,
         "A pressure sensor to make sure nobody goes out alone."],
    40: [images.rescue_ship, images.rescue_ship_shadow, "A rescue ship!"],
    41: [images.mission_control_desk, images.mission_control_desk_shadow, \
         "Mission Control stations."],
    42: [images.button, images.button_shadow,
         "The button for opening the time-locked door in engineering."],
    43: [images.whiteboard, images.full_shadow,
         "The whiteboard is used in brainstorms and planning meetings."],
    44: [images.window, images.full_shadow,
         "The window provides a view out onto the planet surface."],
    45: [images.robot, images.robot_shadow, "A cleaning robot, turned off."],
    46: [images.robot2, images.robot2_shadow,
         "A planet surface exploration robot, awaiting set-up."],
    47: [images.rocket, images.rocket_shadow, "A one-person craft in repair"],
    48: [images.tomic_floor, None, "Toxic floor - do not walk on!"],
```

```
    49: [images.drone, None, "A delivery drone"],
    50: [images.energy_ball, None, "An energy ball - dangerous!"],
    51: [images.energy_ball2, None, "An energy ball - dangerous!"],
    52: [images.computer, images.computer_shadow,
         "A computer workstation, for managing space station systems."],
    53: [images.clipboard, None,
         "A clipboard. Someone has doodled on it.", "the clipboard"],
    54: [images.bubble_gum, None,
         "A piece of sticky bubble gum. Spaceberry flavour.", "bubble gum"],
    55: [images.yoyo, None, "A toy made of fine, strong string and plastic. \
Used for antigrav experiments.", PLAYER_NAME + "'s yoyo"],
    56: [images.thread, None,
         "A piece of fine, sewing string", "a piece of string"],
    57: [images.needle, None,
         "A sharp needle from a cactus plant", "a cactus needle"],
    58: [images.threaded_needle, None,
         "A cactus needle, wearing a length of string", "needle and string"],
    59: [images.canister, None,
         "The air canister has a leak.", "a leaky air canister"],
    60: [images.canister, None,
         "It looks like the seal will hold!", "a sealed air canister"],
    61: [images.mirror, None,
         "The mirror throws a circle of light on the walls.", "a mirror"],
    62: [images.bin_empty, None,
         "A rarely used bin, made of light plastic", "a bin"],
    63: [images.bin_full, None,
         "A heavy bin full of water", "a bin full of water"],
    64: [images.rags, None,
         "An oily rag. Pick it up by one corner if you must!", "an oily rag"],
    65: [images.hammer, None,
         "A hammer. Maybe good for cracking things open...", "a hammer"],
    66: [images.spoon, None, "A large serving spoon", "a spoon"],
    67: [images.food_pouch, None,
         "A dehydrated food pouch. It needs water.", "a dry food pack"],
    68: [images.food, None,
         "A food pouch. Use it to get 100% energy.", "ready-to-eat food"],
    69: [images.book, None, "The book has the words 'Don't Panic' on the \
cover in large, friendly letters", "a book"],
    70: [images.mp3_player, None,
         "An MP3 player, with all the latest tunes", "an MP3 player"],
    71: [images.lander, None, "The Poodle, a small space exploration craft. \
Its black box has a radio sealed inside.", "the Poodle lander"],
    72: [images.radio, None, "A radio communications system, from the \
Poodle", "a communications radio"],
    73: [images.gps_module, None, "A GPS Module", "a GPS module"],
    74: [images.positioning_system, None, "Part of a positioning system. \
Needs a GPS module.", "a positioning interface"],
    75: [images.positioning_system, None,
         "A working positioning system", "a positioning computer"],
    76: [images.scissors, None, "Scissors. They're too blunt to cut \
anything. Can you sharpen them?", "blunt scissors"],
    77: [images.scissors, None,
         "Razor-sharp scissors. Careful!", "sharpened scissors"],
    78: [images.credit, None,
         "A small coin for the station's vending systems",
         "a station credit"],
    79: [images.access_card, None,
         "This access card belongs to " + PLAYER_NAME, "an access card"],
    80: [images.access_card, None,
         "This access card belongs to " + FRIEND1_NAME, "an access card"],
    81: [images.access_card, None,
         "This access card belongs to " + FRIEND2_NAME, "an access card"]
    }

items_player_may_carry = list(range(53, 82))
# Numbers below are for floor, pressure pad, soil, toxic floor.
items_player_may_stand_on = items_player_may_carry + [0, 39, 2, 48]


################
##  SCENERY  ##
################

# Scenery describes objects that cannot move between rooms.
# room number: [[object number, y position, x position]...]
scenery = {
    26: [[39,6,2]],
    27: [[33,5,5], [33,1,1], [33,1,8], [47,8,2],
         [47,3,10], [47,5,0], [42,1,6]],
    28: [[27,0,3], [41,4,9], [41,6,7]],
    29: [[7,2,6], [6,2,8], [12,1,13], [44,0,1],
         [36,4,10], [10,1,1], [19,4,2], [17,4,4]],
    30: [[34,1,1], [35,1,9]],
    31: [[11,1,1], [15,1,9], [46,1,9]],
    32: [[48,2,2], [48,2,9], [48,2,4], [48,3,2], [48,3,9],
         [48,3,4], [48,4,2], [48,4,3], [48,4,4]],
    33: [[13,1,1], [13,1,9], [13,1,5], [13,1,10], [48,2,1],
         [48,2,7], [48,3,6], [48,3,3]],
    34: [[37,2,2], [32,6,7], [37,10,4], [28,6,3]],
    35: [[16,2,9], [16,2,2], [16,9,9], [16,9,6], [16,8,9], [16,8,2], [16,1,8],
         [16,1,3], [12,5,6], [12,9,6], [12,8,8],
         [15,4,6], [12,7,1], [12,7,11]],
    36: [[4,8,1], [9,1,7], [8,1,8], [8,1,9],
         [5,5,4], [6,5,7], [10,1,1], [12,1,2]],
    37: [[48,1,1], [48,3,2], [48,7,11], [48,5,2], [48,5,3],
         [48,7,2], [48,9,2], [48,9,3], [48,11,1], [48,11,2]],
    38: [[42,0,2], [6,2,2], [6,3,5], [6,4,7], [6,2,9], [45,1,10]],
    39: [[39,1,1], [7,9,4], [7,6,4], [6,3,6], [5,6,6],
         [6,5,9], [6,6,5], [35,1,11], [12,1,8], [12,1,9]],
    40: [[41,5,3], [41,5,7], [41,9,3], [41,9,7],
         [13,1,1], [13,1,3], [42,1,12]],
    41: [[4,3,1], [20,3,5], [4,5,1], [20,5,5], [4,7,1],
         [20,7,5], [12,1,1], [12,1,5]],
    44: [[46,4,3], [46,4,8], [18,1,1], [19,1,3],
         [19,1,5], [52,4,7], [24,1,8]],
    45: [[48,2,1], [48,2,2], [48,3,3], [48,3,4], [48,1,4], [48,1,1]],
    46: [[10,1,1], [4,1,2], [6,1,7], [9,1,8], [8,1,8], [6,4,9], [7,3,2]],
    47: [[5,1,1], [5,1,2], [10,1,9], [22,1,7], [5,1,5], [6,4,7], [4,1,8]],
    48: [[17,4,1], [17,4,2], [17,4,3], [17,4,4], [17,4,5], [17,4,6], [17,4,7],
         [17,5,1], [17,5,2], [17,5,3], [17,5,4],
         [17,6,5], [17,5,6], [17,5,7], [14,1,1]],
    49: [[14,2,2], [14,2,5], [7,5,1], [5,5,3], [48,3,3], [48,3,4]],
    50: [[45,4,8], [11,1,1], [19,1,8], [33,2,1], [46,4,6]]
    }
```

```
checksum = 0
check_counter = 0
for key, room_scenery_list in scenery.items():
    for scenery_item_list in room_scenery_list:
        checksum += (scenery_item_list[0] * key
                     + scenery_item_list[1] * (key + 1)
                     + scenery_item_list[2] * (key + 2))
        check_counter += 1
print(check_counter, "scenery items")
assert check_counter == 161, "Expected 161 scenery items"
assert checksum == 200095, "Error in scenery data"
print("Scenery checksum: " + str(checksum))

for room in range(1, 26):# Add random scenery in planet locations.
    if room != 13: # Skip room 13.
        scenery_item = random.choice([16, 28, 29, 30])
        scenery[room] = [[scenery_item, random.randint(2, 10),
                          random.randint(2, 10)]]

# Use loops to add fences to the planet surface rooms.
for room_coordinate in range(0, 13):
    for room_number in [1, 2, 3, 4, 5]: # Add top fence
        scenery[room_number] += [[31, 0, room_coordinate*]]
    for room_number in [1, 6, 11, 16, 21]: # Add left fence
        scenery[room_number] += [[31, room_coordinate, 0]]
    for room_number in [5, 10, 15, 20, 25]: # Add right fence
        scenery[room_number] += [[31, room_coordinate, 12]]

del scenery[21][-1] # Delete last fence panel in Room 21
del scenery[25][-1] # Delete last fence panel in Room 25


###############
## GAME 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(22, 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

    if current_room in scenery:
        for this_scenery in scenery[current_room]:
            scenery_number = this_scenery[0]
            scenery_y = this_scenery[1]
            scenery_x = this_scenery[2]
            room_map[scenery_y][scenery_x] = scenery_number

            image_here = objects[scenery_number][0]
            image_width = image_here.get_width()
            image_width_in_tiles = int(image_width / TILE_SIZE)

            for tile_number in range(1, image_width_in_tiles):
                room_map[scenery_y][scenery_x + tile_number] = 255
```

```
    center_y = int(HEIGHT / 2) # Center of game window
    center_x = int(WIDTH / 2)
    room_pixel_width = room_width * TILE_SIZE # Size of room in pixels
    room_pixel_height = room_height * TILE_SIZE
    top_left_x = center_x - 0.5 * room_pixel_width
    top_left_y = (center_y - 0.5 * room_pixel_height) + 110


###############
## GAME LOOP ##
###############

def start_room():
    show_text("You are here: " + room_name, 0)

def game_loop():
    global player_x, player_y, current_room
    global from_player_x, from_player_y
    global player_image, player_image_shadow
    global selected_item, item_carrying, energy
    global player_offset_x, player_offset_y
    global player_frame, player_direction

    if game_over:
        return

    if player_frame > 0:
        player_frame += 1
        time.sleep(0.05)
        if player_frame == 5:
            player_frame = 0
            player_offset_x = 0
            player_offset_y = 0

#save player's current position
    old_player_x = player_x
    old_player_y = player_y

# move if key is pressed
    if player_frame == 0:
        if keyboard.right:
            from_player_x = player_x
            from_player_y = player_y
            player_x += 1
            player_direction = "right"
            player_frame = 1
        elif keyboard.left: #elif stops player making diagonal movements
            from_player_x = player_x
            from_player_y = player_y
            player_x -= 1
            player_direction = "left"
            player_frame = 1
        elif keyboard.up:
            from_player_x = player_x
            from_player_y = player_y
            player_y -= 1
            player_direction = "up"
            player_frame = 1
        elif keyboard.down:
            from_player_x = player_x
            from_player_y = player_y
            player_y += 1
            player_direction = "down"
            player_frame = 1

# check for exiting the room
    if player_x == room_width: # through door on RIGHT
        #clock.unschedule(hazard_move)
        current_room += 1
        generate_map()
        player_x = 0 # enter at left
        player_y = int(room_height / 2) # enter at door
        player_frame = 0
        #start_room()
        return

    if player_x == -1: # through door on LEFT
        #clock.unschedule(hazard_move)
        current_room -= 1
        generate_map()
        player_x = room_width - 1  # enter at right
        player_y = int(room_height / 2) # enter at door
        player_frame = 0
        #start_room()
        return

    if player_y == room_height: # through door at BOTTOM
        #clock.unschedule(hazard_move)
        current_room += MAP_WIDTH
        generate_map()
        player_y = 0 # enter at top
        player_x = int(room_width / 2) # enter at door
        player_frame = 0
        #start_room()
        return

    if player_y == -1: # through door at TOP
        #clock.unschedule(hazard_move)
        current_room -= MAP_WIDTH
        generate_map()
        player_y = room_height - 1 # enter at bottom
        player_x = int(room_width / 2) # enter at door
        player_frame = 0
        #start_room()
        return



     # If the player is standing somewhere they shouldn't, move them back.
    if room_map[player_y][player_x] not in items_player_may_stand_on: #\
    #           or hazard_map[player_y][player_x] != 0:
        player_x = old_player_x
        player_y = old_player_y
        player_frame = 0
```

```
    if player_direction == "right" and player_frame > 0:
        player_offset_x = -1 + (0.25 * player_frame)
    if player_direction == "left" and player_frame > 0:
        player_offset_x = 1 - (0.25 * player_frame)
    if player_direction == "up" and player_frame > 0:
        player_offset_y = 1 - (0.25 * player_frame)
    if player_direction == "down" and player_frame > 0:
        player_offset_y = -1 + (0.25 * player_frame)



###############
##  DISPLAY  ##
###############

def draw_image(image, y, x):
    screen.blit(
        image,
        (top_left_x + (x * TILE_SIZE),
         top_left_y + (y * TILE_SIZE) - image.get_height())
        )

def draw_shadow(image, y, x):
    screen.blit(
        image,
        (top_left_x + (x * TILE_SIZE),
         top_left_y + (y * TILE_SIZE))
        )

def draw_player():
    player_image = PLAYER[player_direction][player_frame]
    draw_image(player_image, player_y + player_offset_y,
               player_x + player_offset_x)
    player_image_shadow = PLAYER_SHADOW[player_direction][player_frame]
    draw_shadow(player_image_shadow, player_y + player_offset_y,
                player_x + player_offset_x)

def draw():
    if game_over:
        return

    # Clear the game arena area.
    box = Rect((0, 150), (800, 600))
    screen.draw.filled_rect(box, RED)
    box = Rect ((0, 0), (800, top_left_y + (room_height - 1)*30))
    screen.surface.set_clip(box)
    floor_type = get_floor_type()

    for y in range(room_height): # Lay down floor tiles, then items on floor.
        for x in range(room_width):
            draw_image(objects[floor_type][0], y, x)
            # Next line enables shadows to fall on top of objects on floor
            if room_map[y][x] in items_player_may_stand_on:
                draw_image(objects[room_map[y][x]][0], y, x)

     # Pressure pad in room 26 is added here, so props can go on top of it.
    if current_room == 26:
        draw_image(objects[39][0], 5, 2)
        image_on_pad = room_map[8][2]
        if image_on_pad > 0:
            draw_image(objects[image_on_pad][0], 8, 2)

    for y in range(room_height):
        for x in range(room_width):
            item_here = room_map[y][x]
            # Player cannot walk on 255; it marks spaces used by wide objects.
            if item_here not in items_player_may_stand_on + [255]:
                image = objects[item_here][0]

                if (current_room in outdoor_rooms
                    and y == room_height - 1
                    and room_map[y][x] == 1) or \
                    (current_room not in outdoor_rooms
                    and y == room_height - 1
                    and room_map[y][x] == 1
                    and x > 0
                    and x < room_width - 1):
                    # Add transparent wall image in the front row.
                    image = PILLARS[wall_transparency_frame]

                draw_image(image, y, x)

                if objects[item_here][1] is not None: # If object has a shadow
                    shadow_image = objects[item_here][1]
                    # if shadow might need horizontal tiling
                    if shadow_image in [images.half_shadow,
                                        images.full_shadow]:
                        shadow_width = int(image.get_width() / TILE_SIZE)
                        # Use shadow across width of object.
                        for z in range(0, shadow_width):
                            draw_shadow(shadow_image, y, x+z)
                    else:
                        draw_shadow(shadow_image, y, x)

        if (player_y == y):
            draw_player()

    screen.surface.set_clip(None)

def adjust_wall_transparency():
    global wall_transparency_frame

    if (player_y == room_height - 2
        and room_map[room_height - 1][player_x] == 1
        and wall_transparency_frame < 4):
        wall_transparency_frame += 1 # Fade wall out.

    if ((player_y < room_height - 2
            or room_map[room_height - 1][player_x] != 1)
            and wall_transparency_frame > 0):
        wall_transparency_frame -= 1 # Fade wall in.

def show_text(text_to_show, line_number):
    if game_over:
        return
    text_lines = [15, 50]
    box = Rect((0, text_lines[line_number]), (800, 35))
    screen.draw.filled_rect(box, BLACK)
    screen.draw.text(text_to_show,
                     (20, text_lines[line_number]), color="GREEN")



###############
##  START    ##
###############

generate_map()
clock.schedule_interval(game_loop, 0.03)
clock.schedule_interval(adjust_wall_transparency, 0.05)
```