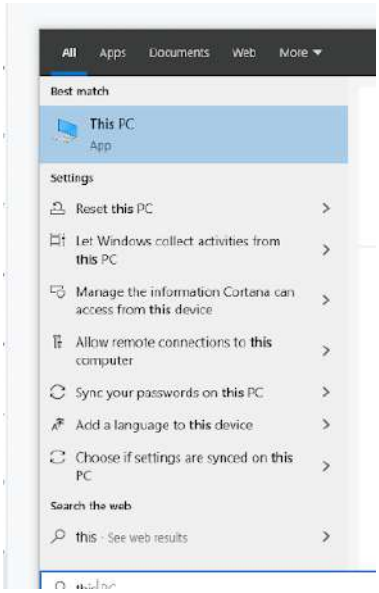
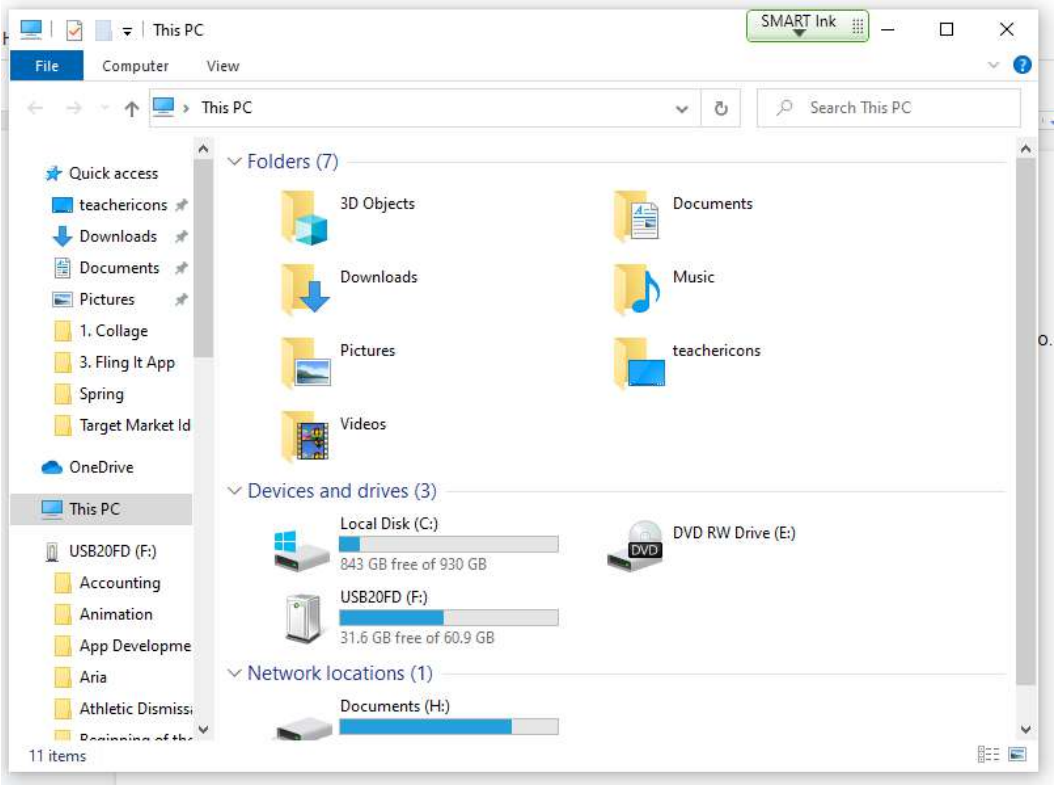
Coding a Plants vs Zombies Game in Pygame

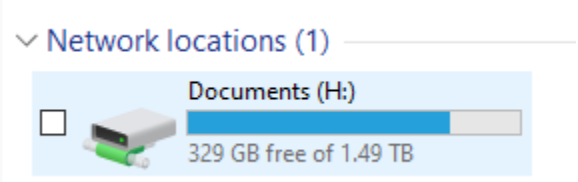
1. First, we need to create a folder in our student directory to save all of our Python files to. Click your Windows button in the bottom left corner and search for “This PC”.



2. Click on “This PC” to pull up your Windows Explorer window.

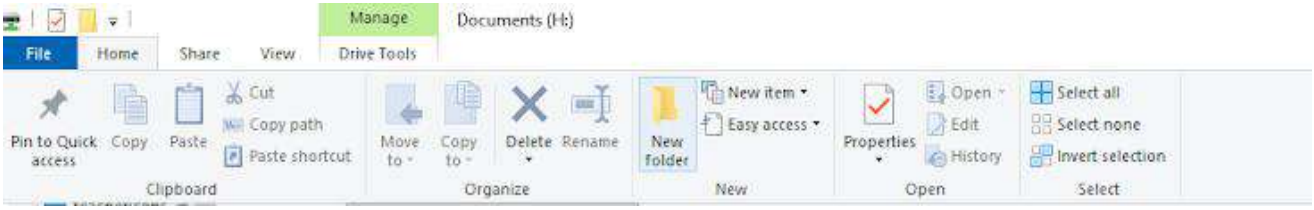


3. Locate your student drive in the menu. Most students have an H: drive. However, some students have a V: drive. You will see your username attached to your student drive. If you don't have a student drive, please let me know. This probably means that we need to have your student drives pulled over from the Monroe server.

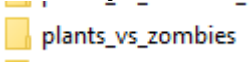


4. Double-click on your H: or V: drive to open it.

5. Click on Home > New Folder to create a new folder.

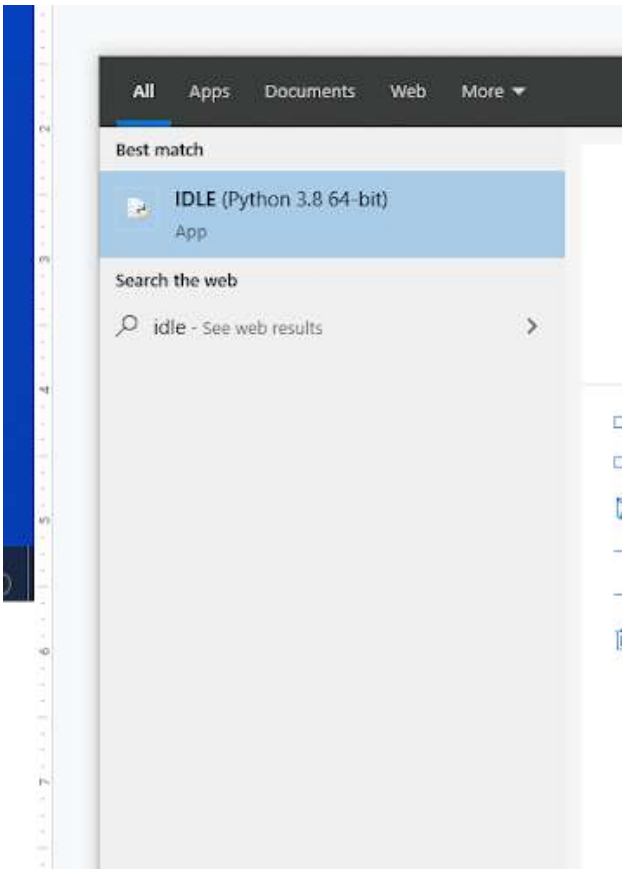


6. Rename your folder, "plants_vs_zombies" and then press ENTER.



7. You can close out of your Windows Explorer window after you have made your folder.

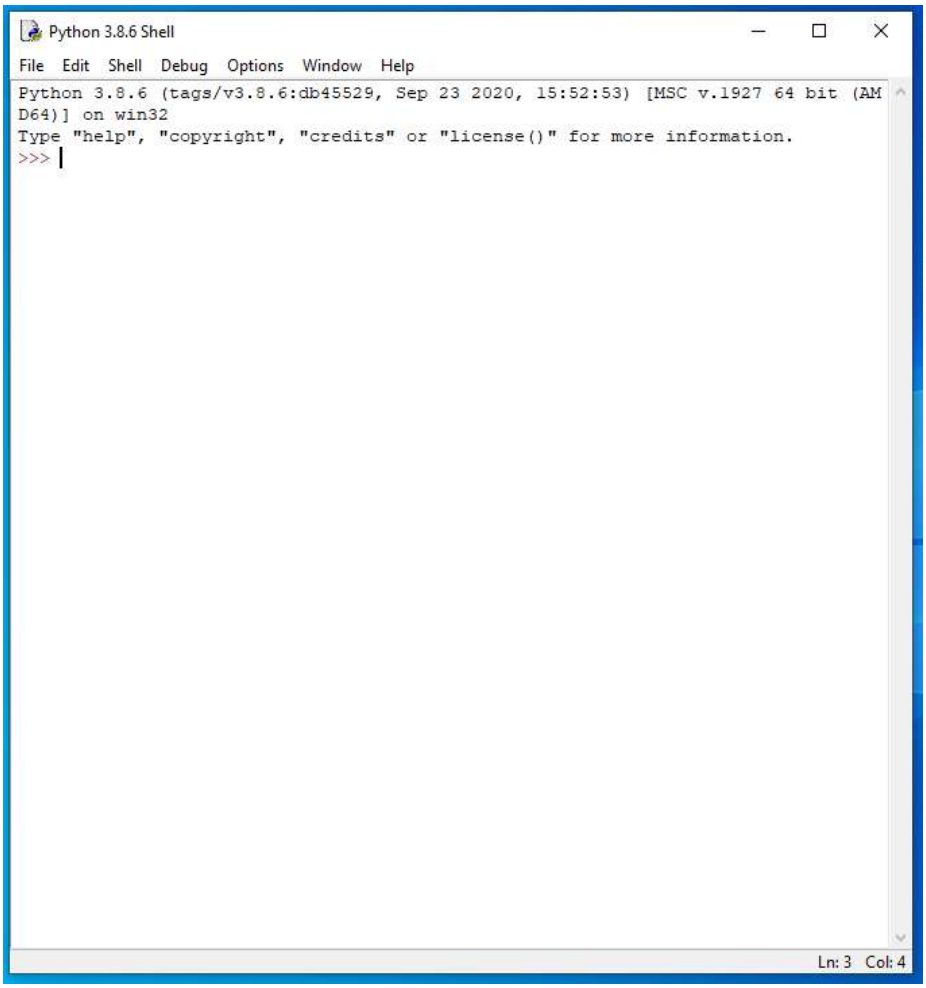
8. Using your Windows button menu, find and launch your IDLE program.



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.

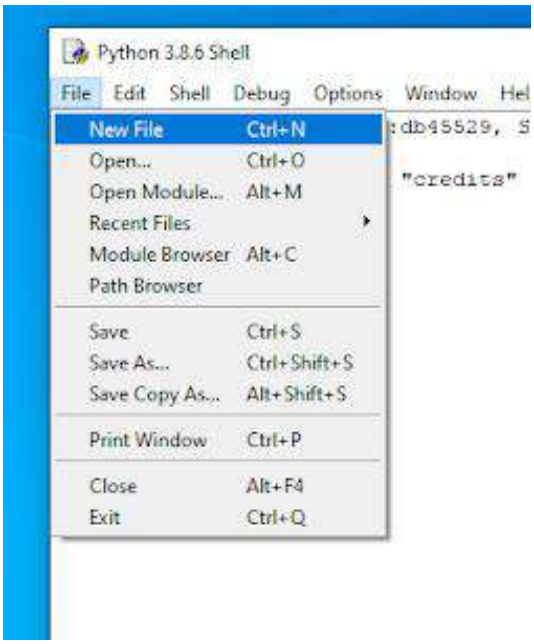
9. Your IDLE window should look something like this once it has launched.:



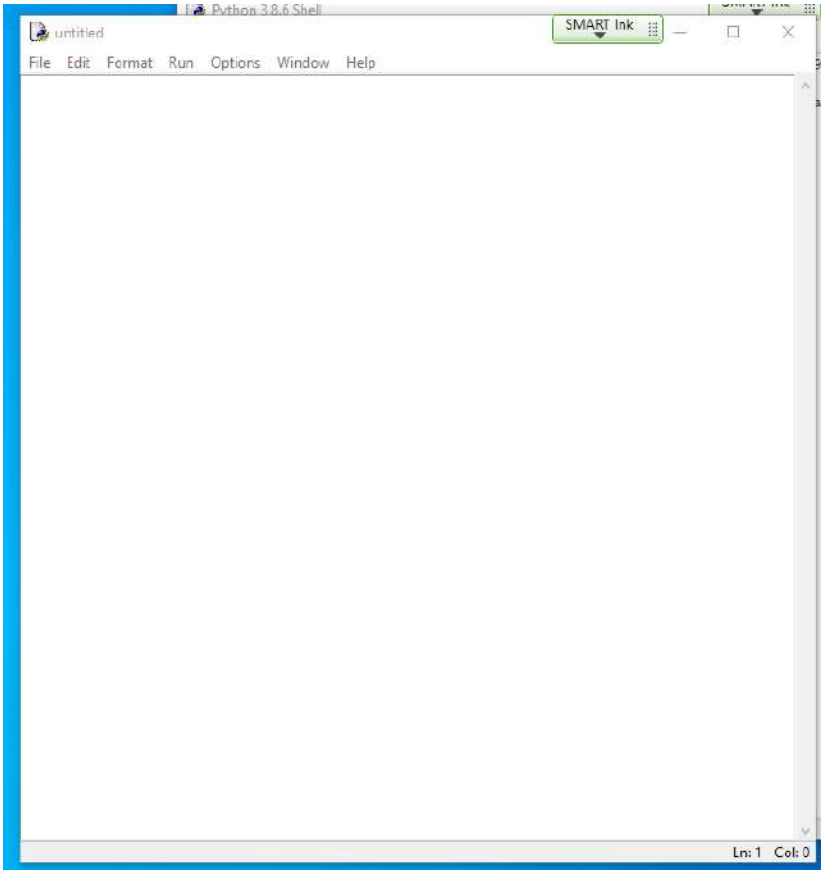
On Startup, IDLE will display the Python Shell, which can be used to give commands to the computer's operating system. Since we are viewing the shell through IDLE and not the actual command prompt window, the commands that we type into the Shell will not communicate directly with our operating system. 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.

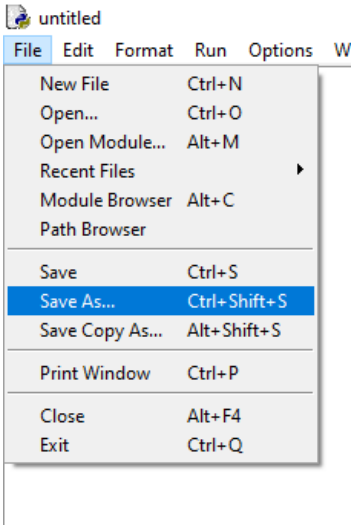
10. Go to File > New File to create a new Python file.



11. A blank Python file will open up. You can tell this apart from the Shell because the title bar will display the file name “untitled” right now because we haven’t saved it yet.

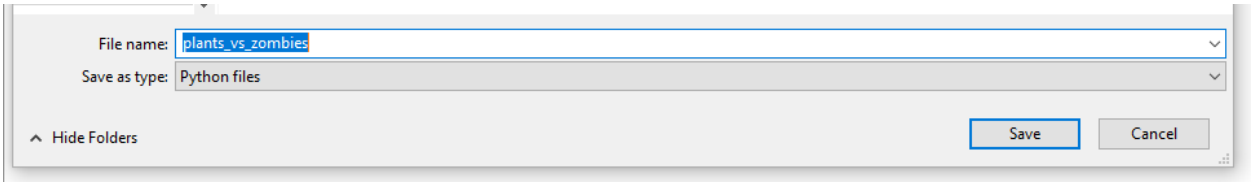


12. Go to File > Save As to save your file.



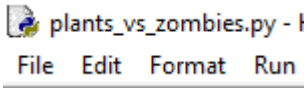
13. Navigate to your H: or V: drive and select the plants_vs_zombies folder that you created earlier. Double-click the folder to open it.

14. Type your new file name in the File Name box. Your new file name is plants_vs_zombies.py



15. Click the Save button.

16. You should now notice that your new Python file has been renamed. You will see the new name in the title bar along the top of the file.



17. We will now begin to code the first part of our game. I like to make my coding window larger so that I can see all of my code a bit better, but that is a personal decision. Remember as we move through these exercises that your spelling, capitalization, and indentation should match. If it doesn't, your program likely won't work.

18. Type the code you see on Lines 1 – 9 of the screenshot below.

```
1 from pygame.locals import *
2 import random
3 import pygame
4
5 WIDTH = 1000
6 HEIGHT = 600
7
8 backimg = pygame.image.load("background.png")
9 panelimg = pygame.image.load("panel.png")
```

Line 1 imports the local pygame module. The local pygame module consists of various constants used by Pygame throughout the execution of this program.

Line 2 imports the random module into the game.

Line 3 imports all pygame functions into the game.

Lines 5 and 6 create two new variables, WIDTH and HEIGHT, and set their values to be equal to 1000 and 600, respectively. These values are expected to change throughout the life of the program, so the variable names are capitalized to indicate that this is a constant variable. Now, future programmers looking at this code know they shouldn't mess with the values of these two variables.

Line 8 creates a variable called backimg. It's value is loaded with the picture, "background.png."

Line 9 creates a variable called panelimg. It's value is loaded with the picture, "panel.png."

19. Press ENTER twice.

20. Type the code you see on Lines 11 – 27 of the screenshot below.

```
8  backimg = pygame.image.load("background.png")
9  panelimg = pygame.image.load("panel.png")
10
11 class SF():
12     def __init__(self, x, y):
13         self.images = []
14         self.images.append(pygame.image.load('sf1.png'))
15         self.images.append(pygame.image.load('sf1.png'))
16         self.images.append(pygame.image.load('sf2.png'))
17         self.images.append(pygame.image.load('sf1.png'))
18         self.images.append(pygame.image.load('sf1.png'))
19         self.images.append(pygame.image.load('sf3.png'))
20         self.index = 0
21         self.image = self.images[self.index]
22         self.rect = pygame.Rect((x*80)+100, (y*80)+100, 65, 80)
23         self.health = 80
24         self.dropx = -10
25         self.dropy = -10
26         self.dropxv = 0
27         self.dropyv = 0
```

On Line 11, we declare a new class called SF. The class names always start with a capital letter.

On Line 12, we use the method def __init__ to create a new instance of the SF class. In other words, to create a new object that is part of the SF class. When we call the def __init__ method, we will give it the values of the x and y coordinate location we want the object to be generated at.

Line 13 creates a new list variable called self.images. The list is empty right now.

Lines 14 – 19 loads the sf1, sf2, and sf3 images into the program and append them to the self.images list that we created on Line 12. Notice that the sf1 image is in the list a total of four times, while the other two images, sf2 and sf3, are only in the list once.

Line 20 creates a new object attribute called self.index and set its value to 0.

Line 21 creates a new object attribute called self.image. Its value is set to the image in the list with an index value equal to the value of the self.index variable. Since we assigned the self.index variable to be 0 on Line 20, the first image that the self.image variable will be assigned would be sf1.png, or the first image in the self.images list.

Line 22 creates another object attribute called self.rect. It's value is equal to a rect object created at the x and y coordinate locations specified. The top left corner of the rect object will be located at (x*80)+100 and (y*80)+100. The rectangle will be 65 pixels long and 80 pixels tall.

Line 23 creates another object attribute called self.health and sets its value to 80.

Line 24 creates another object attribute called self.dropx and sets its initial value to -10.

Line 25 does the same thing for a new object attribute called self.dropy.

Line 26 creates another object attribute called self.dropxv and sets its value to 0.

Line 27 creates another object attribute called self.dropyv and sets its value to 0 as well.

Line 33 will copy the chosen balloon's image onto the new surface created in Line 31.

21. Press ENTER twice.

22. Type the code you see on Lines 29 – 40 of the screenshot below. Ensure your alignment matches what is shown in the screenshot.

```
11 class SF():
12     def __init__(self, x, y):
13         self.images = []
14         self.images.append(pygame.image.load('sf1.png'))
15         self.images.append(pygame.image.load('sf1.png'))
16         self.images.append(pygame.image.load('sf2.png'))
17         self.images.append(pygame.image.load('sf1.png'))
18         self.images.append(pygame.image.load('sf1.png'))
19         self.images.append(pygame.image.load('sf3.png'))
20         self.index = 0
21         self.image = self.images[self.index]
22         self.rect = pygame.Rect((x*80)+100, (y*80)+100, 65, 80)
23         self.health = 80
24         self.dropx = -10
25         self.dropy = -10
26         self.dropxv = 0
27         self.dropyv = 0
28 
29     def update(self):
30         self.index += 1
31         if self.index >= len(self.images):
32             self.index = 0
33         self.image = self.images[self.index]
34         for i in range (0,len(zombies)):
35             if (self.rect.colliderect(zombies[i].rect)):
36                 self.health -= 1
37                 if (self.health > 0):
38                     zombies[i].eating = True
39                 else:
40                     zombies[i].eating = False
41 
```

Line 29 creates another object method for the SF object class. This method is called update.

Line 30 increases the value of the self.index object attribute by 1.

Line 31 begins an “if” loop that will check to see if the value of the self.index variable is greater than or equal to the length of the self.images list. Remember the self.images list has 6 images in it, with index values ranging from 0 through 5. If the self.index variable is greater than or equal to 6 (because that is how many images are in the self.images list), Line 32 will run to change the value of the self.index variable back to 0.

Line 33 will update the value of the self.image variable to be a new picture. Remember we just updated the self.index variable. Line 33 will go back to the self.images list of images and reference the image at the new index position, and then assign that image to the self.image variable, overwriting the old image that was stored there.

Line 34 begins a “for” loop that runs for every item in the zombies list. We have not created this list yet, but we will in the future.

For each item in the zombies list, Line 35 will check to see if the rect of the SF class object collided with the rect for that particular zombie object in the zombies list. If it has, then the self.health variable for the SF object will be decreased by 1.

After decreasing the SF object’s health by 1, Line 37 will check to see that the self.health variable is still greater than 0. If it is, then the zombies.eating variable for that particular zombie object would be set to True. Otherwise, if the self.health variable is NOT greater than 0, the zombies.eating variable for that particular zombie object would be set to False.

23. Press ENTER twice.

24. Type the code you see on Lines 42 – 62 of the screenshot below. Ensure your indentation matches what is shown in the screenshot.

```
40                      zombies[i].eating = False
41
42      def action(self):
43          #if drop at panel, set off screen and add sun
44          if ((abs(self.dropx - 330) < 100) and (abs(self.dropy - 30) < 30)):
45              self.dropx = -10
46              self.dropy = -10
47              self.dropxv = 0
48              self.dropyv = 0
49              return 10
50          #else if active then move and draw
51          elif ((self.dropx != -10) and (self.dropy != -10)):
52              self.dropx += self.dropxv
53              self.dropy += self.dropyv
54              pygame.draw.rect(screen, (255,255,0), pygame.Rect(self.dropx,self.dropy,6,6))
55          #else maybe make a new one
56          else:
57              if (random.randint(0,80) == 0):
58                  self.dropx = self.rect.x + 40
59                  self.dropy = self.rect.y + 20
60                  self.dropxv = (380 - self.dropx) // 50
61                  self.dropyv = (30 - self.dropy) // 50
62          return 0
```

Line 42 creates a new class method called action.

Line 43 contains a comment.

Line 44 begins an “if” loop. This “if” functions checks whether the absolute value of self.dropx – 330 is less than 100 and if the absolute value of self.dropy – 30 is less than 30. The “abs” function returns the absolute value of a number. In other words, it will remove the negative sign if there is one.

If both of these conditions are true. The code on Lines 45 – 49 will run to change the value of the object’s self.dropx, self.dropy, self.dropxv, and self.dropyv variables. The “if” function will also return the value of 10.

Line 50 contains another comment.

Line 51 contains an “else if” function. This function will run if the first “if” function proves to be False. Line 51 checks to see if the self.dropx value is NOT equal to -10 and if the self.dropy value is NOT equal to -10.

If Line 51 is true, Lines 52 – 54 will run. Line 52 will increase the value of the self.dropx variable by the value of the self.dropxv variable. Line 53 will increase the value of the self.dropy variable by the value of the self.dropyv variable.

Line 54 draws a rectangle on the screen surface. The rectangle is filled with the color with the RGB values of 255, 255, 0 (yellow). It will be drawn at the location of the self.dropx and self.dropy coordinates and have a width and height of 6.

Line 55 contains another comment.

Line 56 begins an “else” loop. The code inside this block will execute if the if function on Line 44 and the elif function on Line 51 both return “False.”

Line 57 begins an “if” loop. This loop will run the random.randint function to generate a random number between 0 and 79 (the randint function includes everything up to, but not including, the highest number in the range of numbers). If the random number generated is equal to 0, Lines 58 – 61 will run to adjust the value of the dropx, dropy, dropxv and dropxy variables.

Line 62 will return the value of 0 to the game if the elif or else functions on Lines 51 and 62 run.

25. Press ENTER twice.

26. Type the code you see on Lines 64 – 65 of the screenshot below. Ensure your indentation matches what is shown in the screenshot.

```
61                  self.dropyv = (30 - self.dropy) // 50
62          return 0
63
64      def draw(self):
65          screen.blit(self.image, (self.rect.x,self.rect.y))
66
```

Line 64 creates another class method for the SF class. This method is called draw.

Line 65 blits the image in the self.image variable to the screen surface at the location of the self.rect.x and self.rect.y coordinates.

27. Press ENTER twice.

28. Backspace all the way back to the left margin.

29. Type the code you see on Lines 67 – 81 of the screenshot below. Ensure your indentation matches up with what is shown in the screenshot.

```
64      def draw(self):
65          screen.blit(self.image, (self.rect.x,self.rect.y))
66
67  class NUT(SF):
68      def __init__(self, x, y):
69          super().__init__(x, y)
70          self.images = []
71          self.images.append(pygame.image.load('nt1.png'))
72          self.images.append(pygame.image.load('nt1.png'))
73          self.images.append(pygame.image.load('nt1.png'))
74          self.images.append(pygame.image.load('nt1.png'))
75          self.images.append(pygame.image.load('nt1.png'))
76          self.images.append(pygame.image.load('nt1.png'))
77          self.images.append(pygame.image.load('nt1.png'))
78          self.images.append(pygame.image.load('nt1.png'))
79          self.images.append(pygame.image.load('nt1.png'))
80          self.images.append(pygame.image.load('nt2.png'))
81          self.health = 250
```

Line 67 declares another class called NUT. The NUT class will inherit the attributes of the SF class that you just created.

Line 68 uses the def __init__ method to create a new instance of the NUT class. In other words, to create a new object that is part of the NUT class. When we call the def __init__ method, we give it the values of the x and y coordinate location we want the object to use when the program creates it.

Line 68 uses the super().__init__ method to export the x and y coordinates the object has taken in (when it was created) to the super class, which is, in this case, the SF class. The super class is the class that the object has inherited its properties from. Any changes to object attributes that occur in the NUT class will overwrite the values established by the SF superclass.

Line 70 creates a blank self.images list variable, and Lines 71 – 80 load and append the images to the self.images list. This class will overwrite the frames that were loaded into the self.images list SF class on Lines 14 – 19 of the SF class.

Line 81 creates the self.health object attribute/variable and sets its value to be equal to 250.

30. Press ENTER twice.

31. Type the code you see on Lines 83 – 84 of the screenshot below. Ensure the indentation matches up with what is shown in the screenshot.

```
67 class NUT(SF):
68     def __init__(self, x, y):
69         super().__init__(x, y)
70         self.images = []
71         self.images.append(pygame.image.load('ntl.png'))
72         self.images.append(pygame.image.load('ntl.png'))
73         self.images.append(pygame.image.load('ntl.png'))
74         self.images.append(pygame.image.load('ntl.png'))
75         self.images.append(pygame.image.load('ntl.png'))
76         self.images.append(pygame.image.load('ntl.png'))
77         self.images.append(pygame.image.load('ntl.png'))
78         self.images.append(pygame.image.load('ntl.png'))
79         self.images.append(pygame.image.load('ntl.png'))
80         self.images.append(pygame.image.load('nt2.png'))
81         self.health = 250
82
83     def action(self):
84         return 0
85
```

Line 83 creates another method for the NUT class object. This method is called action.

Line 84 will return the value of 0 for the method.

32. Press ENTER twice.

33. Backspace until your insertion point is at the left margin.

34. Type the code you see on Lines 86 – 99 of the screenshot below.

```
83      def action(self):
84          return 0
85
86  class SHOOT(SF):
87      def __init__(self, x, y):
88          super().__init__(x, y)
89          self.images = []
90          self.images.append(pygame.image.load('sh1.png'))
91          self.images.append(pygame.image.load('sh1.png'))
92          self.images.append(pygame.image.load('sh1.png'))
93          self.images.append(pygame.image.load('sh1.png'))
94          self.images.append(pygame.image.load('sh1.png'))
95          self.images.append(pygame.image.load('sh1.png'))
96          self.images.append(pygame.image.load('sh1.png'))
97          self.images.append(pygame.image.load('sh2.png'))
98          self.health = 80
99          self.reload = 40
```

Lines 86 – 99 are very similar to the code you created for the NUT class on Lines 67 – 81. The class created on Line 86 is called SHOOT. It inherits the attributes of the SF class.

Line 87 uses the def __init__ method to initialize a new SHOOT class object.

Lines 89 – 98 are exactly like what you created in the NUT class. You load and append images to the self.images list to overwrite the images created by the SF class object. Line 98 assigns a different self.health value than the NUT class has. Remember, objects in the NUT class have a self.health of 250. Objects in the SHOOT class have a self.health value of 80.

Line 99 creates a new class attribute called self.reload. The value for the self.reload attribute is set to 40.

35. Press ENTER twice.

36. Type the code you see on Lines 101 – 107 of the screenshot below. Ensure your indentation matches what is shown in the screenshot.

```
86  class SHOOT(SF):
87      def __init__(self, x, y):
88          super().__init__(x, y)
89          self.images = []
90          self.images.append(pygame.image.load('sh1.png'))
91          self.images.append(pygame.image.load('sh1.png'))
92          self.images.append(pygame.image.load('sh1.png'))
93          self.images.append(pygame.image.load('sh1.png'))
94          self.images.append(pygame.image.load('sh1.png'))
95          self.images.append(pygame.image.load('sh1.png'))
96          self.images.append(pygame.image.load('sh1.png'))
97          self.images.append(pygame.image.load('sh2.png'))
98          self.health = 80
99          self.reload = 40
100
101     def action(self):
102         self.reload -= 1
103         if ((self.reload <= 0) and (zombie_on_row(self.rect.y))):
104             self.reload = 40
105             boom = BULLET(self.rect.x+50, self.rect.y+25)
106             bullets.append(boom)
107         return 0
```

Line 101 creates a new class method called action.

Whenever this method runs, Line 102 will subtract 1 from the value of the self.reload variable.

Line 103 begins an “if” statement that checks to see if the value of the self.reload variable is less than 0 AND if there is a zombie on the current row by checking the rect’s y coordinate position and determining which row the zombie is on. It should be noted that we haven’t created the zombie_on_row method yet for this game.

If Line 103 proves to be true, the self.reload variable will be set to 40 on Line 104. Line 105 creates the boom variable and sets its value to be a new BULLET class object (which we haven’t coded yet) at the location of self.rect.x+50 and self.rect.y+25.

Line 106 will append the current bullet being generated to the bullets list, which we also haven’t created yet.

Line 107 will return the value of 0 for the method.

37. Press ENTER twice.

38. Backspace all the way back until your insertion point is at the left margin.

39. Type the code you see on Lines 109 – 116 of the screenshot below.

```
107         return 0
108
109 class ZOMBIE(SF):
110     def __init__(self, x, y):
111         super().__init__(x, y)
112         self.image = pygame.image.load('zombie1.png')
113         self.rect = pygame.Rect((x*80)+100, (y*80)+100, 50, 80)
114         self.pause = 15
115         self.health = 20
116         self.eating = False
```

Line 109 creates another class named ZOMBIE. The ZOMBIE class will inherit the SF superclass attributes.

Line 110 uses the def __init__ method to create a new instance of the ZOMBIE class object. This method requires the x and y coordinates to be entered when it is called.

Line 111 will pass the x and y coordinates the object took in on Line 110 to the superclass (SF) and override those attribute values in the superclass.

Line 112 will override the self.image variable in the SF class by loading the zombie1.png image into the program.

Line 113 will override the self.rect variable from the SF class by changing the rect size and position to the coordinates specified in the parentheses on Line 113. The only real change this makes is that the width (50) is different.

Line 114 creates a class attribute called self.pause and sets its value to 15.

Line 115 overrides the self.health class attribute from the SF class with a new value of 20.

Line 116 creates the self.eating class attribute and sets its value to True.

40. Press ENTER twice.

41. Type the code you see on Lines 118 – 123 of the screenshot below. Ensure your indentation matches up with what is shown in the screenshot.

```
109 class ZOMBIE(SF):
110     def __init__(self, x, y):
111         super().__init__(x, y)
112         self.image = pygame.image.load('zombie1.png')
113         self.rect = pygame.Rect((x*80)+100, (y*80)+100, 50, 80)
114         self.pause = 15
115         self.health = 20
116         self.eating = False
117 
118     def update(self):
119         self.pause -= 1
120         if (self.pause <= 0):
121             self.pause = 15
122             if (self.eating == False):
123                 self.rect.move_ip(-5,0)
```

Line 118 creates another class method for the ZOMBIE class called update.

Whenever this method is called, Line 119 decreases the self.pause value by 1.

We are going to use the self.pause attribute to indicate the time between the zombie's steps. The self.eating attribute designates whether the zombie is eating anything or not.

Line 120 begins an "if" loop to check to see if the self.pause attribute is less than or equal to 0. If it is, Line 121 will change the self.pause attributes value to 15.

After changing the self.pause attribute's value, another "if" loop begins on Line 122 to check to see if the self.eating attribute's value is False. If the value is False, Line 123 uses the rect.move_ip method to move the zombie class object five pixels to the left on the x-axis.

The rect.move_ip method stands for rect.move in place. This will move the entire object, including all images that are blitting inside that particular rect. Remember, pygame rect objects are just rectangles that have a position and size. To move the image along with the rect, we need to use the move_ip function. If we only wanted to move the rect itself, the rect.move() function would work. The alternative to the move_ip function is to re-blit, or re-draw, the image inside the rect each time the rect moves.

42. Press ENTER twice.

43. Backspace your insertion point all the way to the left margin.

44. Type the code you see on Lines 125 – 128 of the screenshot below. Ensure your indentation matches up with what is shown in the screenshot.

```
122             if (self.eating == False):
123                 self.rect.move_ip(-5,0)
124
125 class BULLET:
126     #constructor
127     def __init__(self, x, y):
128         self.rect = pygame.Rect(x,y,10,10)
129
```

Line 125 creates a new class called BULLET. Notice that this class DOES NOT inherit any attributes from the SF class as the previous few classes have.

Line 126 contains a comment.

Line 127 uses the def __init__ method to create a new instance of a BULLET class object. When this method is called, it will take in the x and y values from the program.

Line 128 creates a new self.rect attribute. The attribute's value is set to be equal to a new pygame rect object generated at the location of x and y with a height and width of 10.

45. Press ENTER twice.

46. Type the code you see on Lines 130 – 137. Ensure your indentation matches up with what is shown in the screenshot below.

```
125 class BULLET:
126     #constructor
127     def __init__(self, x, y):
128         self.rect = pygame.Rect(x,y,10,10)
129
130     #move the bullet
131     def move(self):
132         self.rect.move_ip(15,0)
133         for i in range(0, len(zombies)):
134             if (self.rect.colliderect(zombies[i].rect)):
135                 zombies[i].health -= 2
136                 self.rect.x = WIDTH
137                 return
```

Line 130 contains another comment.

Line 131 creates another class method called move.

Line 132 will move the bullet rect object (and image) by 15 pixels along the x axis. (towards the right – positive x direction).

Line 133 begins a “for” loop that will loop for each item in the zombies list. We have not initialized this list yet.

For each item in the zombies list, Line 134 will check to see if the bullet’s rect has collided with that particular zombie’s rect object.

If this is true (meaning the bullet has collided with the zombie), the zombie’s health attribute will be decreased by 2 (Line 135).

Line 136 will change the value of the bullet’s rect.x coordinate to be equal to the WIDTH variable. Essentially, the bullet will disappear from the screen.

Line 137 will return the value of “None” to the program. In this case, the return statement means the code will go no further and will not continue on in that block of code. If Line 137 is executed, the return statement functions similar to a “break” statement.

47. Press ENTER twice.

48. Type the code you see on Lines 139 – 147 of the screenshot below. Ensure your indentation matches what is shown in the screenshot.

```
125 class BULLET:
126     #constructor
127     def __init__(self, x, y):
128         self.rect = pygame.Rect(x,y,10,10)
129 
130     #move the bullet
131     def move(self):
132         self.rect.move_ip(15,0)
133         for i in range(0, len(zombies)):
134             if (self.rect.colliderect(zombies[i].rect)):
135                 zombies[i].health -= 2
136                 self.rect.x = WIDTH
137                 return
138 
139     #draw the bullet
140     def draw(self):
141         pygame.draw.rect(screen, (255,255,255), pygame.Rect(self.rect.x,self.rect.y,6,6))
142 
143 def zombie_on_row(row):
144     for i in range (0,len(zombies)):
145         if (zombies[i].rect.y == row):
146             return True
147     return False
```

Line 139 contains another comment.

Line 140 creates another BULLET class method called draw. This method will draw the bullet class object's rectangle on the screen in white (with an RGB color of 255, 255, 255). The rect will be drawn at the location of the self.rect.x and self.rect.y coordinates and have a width and height of 6 (Line 141).

Line 142 contains a blank line.

Line 143 creates another method called zombie_on_row. We called this method earlier in the game, but haven't created it yet. This method will need the "row" parameter entered into it whenever it is called.

Line 144 will loop for each item in the zombies list. Again, we haven't created the zombies list yet.

Line 145 will check to see if each zombie's rect.y position is equal to the value of the row. If it is, the function will return the value of True (Line 146). If not, it will return the value of False (Line 147). This method will be used to check to see if the zombie rect object is on that row.

49. Press ENTER twice.

50. Backspace all the way to the left margin.

51. Type the code you see on Lines 149 – 153 of the screenshot below.

```
143 def zombie_on_row(row):
144     for i in range (0,len(zombies)):
145         if (zombies[i].rect.y == row):
146             return True
147     return False
148
149 pygame.init()
150 screen = pygame.display.set_mode((WIDTH, HEIGHT))
151 pygame.display.set_caption("Plants vs Zombies")
152 font = pygame.font.SysFont("comicsansms", 16)
153 clock = pygame.time.Clock()
```

Line 149 initializes pygame.

Line 150 creates a variable called screen. The display.set_mode will create a window on the user's screen with a width and height equal to the values specified in the WIDTH and HEIGHT variables. The screen is going to be our game window.

Line 151 sets the caption for the game window to be "Plants vs Zombies."

Line 152 creates a new variable called font. The font variable's value is set to a new font object that uses the Comic Sans MS font in size 16. This font should be installed on your system.

Line 153 creates a variable called clock. The clock variable's value is set to be a new Clock() object.

52. Press ENTER twice.

53. Type the code you see on Lines 155 – 157 of the screenshot below.

```
149 pygame.init()
150 screen = pygame.display.set_mode((WIDTH, HEIGHT))
151 pygame.display.set_caption("Plants vs Zombies")
152 font = pygame.font.SysFont("comicsansms", 16)
153 clock = pygame.time.Clock()
154
155 items = []
156 zombies = []
157 bullets = []
```

Lines 155 – 157 create three new list variables: items, zombies, and bullets. All of the list variables are set to be empty.

54. Press ENTER twice.

55. Type the code you see on Lines 159 – 161 of the screenshot below.

```
149 pygame.init()
150 screen = pygame.display.set_mode((WIDTH, HEIGHT))
151 pygame.display.set_caption("Plants vs Zombies")
152 font = pygame.font.SysFont("comicsansms", 16)
153 clock = pygame.time.Clock()
154
155 items = []
156 zombies = []
157 bullets = []
158
159 sun = 50
160 score = 0
161 spawn = 20
```

Line 159 creates the sun variable and sets its value to be equal to 50. This is the amount of sunlight in the game.

Line 160 creates the score variable and sets its value to be equal to 0.

Line 161 creates the spawn variable and sets its value to be equal to 20. This means that a new zombie will spawn every 20 game cycles.

56. Press ENTER twice.

57. Type the code you see on Lines 163 – 171 of the screenshot below.

```
159 sun = 50
160 score = 0
161 spawn = 20
162
163 #make a sunflower off the grid so player always has some sun
164 flower = SF(-1,5)
165 items.append(flower)
166
167 item_selected = 0
168 item1cost = 30
169 item2cost = 40
170 item3cost = 60
171 empty_squares = [[0 for j in range(6)] for i in range(10)]
```

Line 163 contains a comment.

Line 164 creates the flower variable. The variable's value is set to be a new SF class object with the x and y coordinates of -1 and 5.

Line 165 will append the flower SF object to the items list.

Line 166 is blank.

Line 167 creates a new variable called item_selected and sets its value to 0.

Line 168 creates a new variable called item1cost and sets its value to 30.

Line 169 creates a new variable called item2cost and sets its value to 40.

Line 170 creates a new variable called item3cost and sets its value to 60.

Line 171 creates a new variable called empty_squares creates a 2D list full of zero values that is 6 by 10. This list will be used for our play area. The number 0 will represent that the square is empty and the number 1 will represent that the square has something on it. Whenever something moves onto a square, it will change that squares value in the empty_squares list to 1.

58. Press ENTER twice.

59. Type the code you see on Lines 173 – 191 of the screenshot below. Ensure your indentation matches what is shown in the screenshot.

```
171 empty_squares = [[0 for j in range(6)] for i in range(10)]
172
173 #main loop
174 running = True
175 game_over = False
176 while running == True:
177     clock.tick(10)
178     #get ESCAPE to quit
179     for e in pygame.event.get():
180             if (e.type == pygame.QUIT):
181                 running = False
182             if ((e.type == KEYDOWN) and (e.key == K_ESCAPE)):
183                 running = False
184             if ((e.type == KEYDOWN) and (e.key == K_0)):
185                 item_selected = 0
186             if ((e.type == KEYDOWN) and (e.key == K_1) and (sun >= item1cost)):
187                 item_selected = 1
188             if ((e.type == KEYDOWN) and (e.key == K_2) and (sun >= item2cost)):
189                 item_selected = 2
190             if ((e.type == KEYDOWN) and (e.key == K_3) and (sun >= item3cost)):
191                 item_selected = 3
```

Line 173 contains a comment.

Lines 174 and 175 create two variables called running and game_over. The running variable's value is set to True and the game_over variable's value is set to False.

Line 176 begins a "while" loop that will run while the running variable is equal to True. Since the running variable will be set to True for the entirety of the game and will be changed to False when the game is over, this loop of code will run throughout the whole game.

Line 177 sets the frame rate for the game to 10. This is a slower frame rate than most games. Most games run around 40-60 frames per second. We are settings ours at 10. The primary reason for this is that we don't have a whole lot of moving parts in our game so the user won't likely notice that the game is running slower.

Line 178 contains another comment.

Lines 179 – 191 begin a "for" loop that will check each even in the event queue list against several conditions.

Line 180 begins by checking whether the "x" or "Close" button has been clicked on the game window. If so, Like 181 changes the running variable to False.

If Line 180 proves to be False, then the program continues checking whether certain keys are being currently pressed on the keyboard.

Line 182 checks to see if the user is holding down the ESC key. If either of these conditions are true, Lines 181 and 183 will change the value of the running variable to False and end the game loop.

Line 184 checks to see if the user is holding down the zero (0) key. If this is true, Line 185 will set the value of the item_selected variable to be 0.

Line 186 checks to see if the user is holding down the 1 key. It also checks to see if the value of the sun variable is greater than or equal to the item1cost variable's value. If this is true, Line 187 will set the value of the item_selected variable to be 1.

Line 188 checks to see if the user is holding down the 2 key. It also checks to see if the value of the sun variable is greater than or equal to the item2cost variable's value. If this is true, Line 189 will set the value of the item_selected variable to be 2.

Line 190 checks to see if the user is holding down the 3 key. It also checks to see if the value of the sun variable is greater than or equal to the item3cost variable's value. If this is true, Line 191 will set the value of the item_selected variable to be 3.

60. Press ENTER.

61. Type the code you see on Lines 192 – 210 of the screenshot below. Ensure your indentation matches what is shown in the screenshot.

```
176 while running == True:
177     clock.tick(10)
178     #get ESCAPE to quit
179     for e in pygame.event.get():
180             if (e.type == pygame.QUIT):
181                 running = False
182             if ((e.type == KEYDOWN) and (e.key == K_ESCAPE)):
183                 running = False
184             if ((e.type == KEYDOWN) and (e.key == K_0)):
185                 item_selected = 0
186             if ((e.type == KEYDOWN) and (e.key == K_1) and (sun >= item1cost)):
187                 item_selected = 1
188             if ((e.type == KEYDOWN) and (e.key == K_2) and (sun >= item2cost)):
189                 item_selected = 2
190             if ((e.type == KEYDOWN) and (e.key == K_3) and (sun >= item3cost)):
191                 item_selected = 3
192             if e.type == pygame.MOUSEBUTTONDOWN:
193                 if (item_selected != 0):
194                     (mx,my) = pygame.mouse.get_pos()
195                     if ((mx > 100) and (my > 100) and (mx < 900) and (my < 580)):
196                         mx = ((mx - 100) // 80)
197                         my = ((my - 100) // 80)
198                         if (empty_squares[mx][my] == 0):
199                             empty_squares[mx][my] = 1
200                             if (item_selected == 1):
201                                 tmp = SF(mx, my)
202                                 sun -= item1cost
203                             elif (item_selected == 2):
204                                 tmp = SHOOT(mx, my)
205                                 sun -= item2cost
206                             elif (item_selected == 3):
207                                 tmp = NUT(mx, my)
208                                 sun -= item3cost
209                             items.append(tmp)
210                             item_selected = 0
```

Line 192 continues to check for events in the event queue. Line 192 now checks to see if the user has clicked the mouse. If so, the rest of the code under this block will run.

If the user has clicked the mouse, the game needs to determine what the user has actually clicked on the screen.

Line 193 will check to see if the item_selected variable is NOT equal to 0. If this is true, then a new list using the mx and my (mouse x and mouse y) values that we will assign is created using the mouse cursor's coordinate positions at the location of the click. This list will be used to identify exactly what the user has clicked.

Line 195 will check to see if the mouse cursor's x coordinate is greater than 100 and the y coordinate is greater than 100. It will also check to see if the mouse cursor's x coordinate is less than 900 and y coordinate is less than 580. This line checks to see if the user has actually clicked inside the play area for the game.

If this is true, then a new value for the mx and my variables will be calculated on Lines 196 and 197. This will find the number of the row and column the user has clicked in. Remember our game board is laid out in actual rows and columns. We need to know exactly where the user clicked.

Line 198 will check for the mx and my coordinate values in the empty_squares list are 0, meaning that the squares are empty. If they are, line 199 will change the value of the empty_squares list to 1, meaning that the square is no longer marked as empty in the empty_squares list.

Line 200 will check to see if the item selected by the user is equal to 1. If it is, than a variable called tmp will be created on Line 201. This variable will generate a new SF object at the location of mx and my.

Line 202 will decrease the sun variable by the amount of the item1cost.

Line 203 – 208 will perform the same type of behavior to generate SHOOT and NUT objects at the proper location if the user has selected them.

Line 209 appends the tmp object that was generated to the items list.

Line 210 resets the item_selected value to 0 to get ready for another selection.

62. Press ENTER twice.

63. Type the code you see on Lines 212 – 216 of the screenshot below. Ensure your indentation matches what is shown in the screenshot.

```
176 while running == True:
177     clock.tick(10)
178     #get ESCAPE to quit
179     for e in pygame.event.get():
180             if (e.type == pygame.QUIT):
181                 running = False
182             if ((e.type == KEYDOWN) and (e.key == K_ESCAPE)):
183                 running = False
184             if ((e.type == KEYDOWN) and (e.key == K_0)):
185                 item_selected = 0
186             if ((e.type == KEYDOWN) and (e.key == K_1) and (sun >= item1cost)):
187                 item_selected = 1
188             if ((e.type == KEYDOWN) and (e.key == K_2) and (sun >= item2cost)):
189                 item_selected = 2
190             if ((e.type == KEYDOWN) and (e.key == K_3) and (sun >= item3cost)):
191                 item_selected = 3
192             if e.type == pygame.MOUSEBUTTONDOWN:
193                 if (item_selected != 0):
194                     (mx,my) = pygame.mouse.get_pos()
195                     if ((mx > 100) and (my > 100) and (mx < 900) and (my < 580)):
196                         mx = ((mx - 100) // 80)
197                         my = ((my - 100) // 80)
198                         if (empty_squares[mx][my] == 0):
199                             empty_squares[mx][my] = 1
200                             if (item_selected == 1):
201                                 tmp = SF(mx, my)
202                                 sun -= item1cost
203                             elif (item_selected == 2):
204                                 tmp = SHOOT(mx, my)
205                                 sun -= item2cost
206                             elif (item_selected == 3):
207                                 tmp = NUT(mx, my)
208                                 sun -= item3cost
209                             items.append(tmp)
210                             item_selected = 0
211
212     spawn -= 1
213     if (spawn <= 0):
214         zombie = ZOMBIE(9,random.randint(0,5))
215         zombies.append(zombie)
216         spawn = 150 + random.randint(0,50)
```

Line 212 decreases the value of the spawn variable we created on Line 161 by 1.

Line 213 will then check to see if the new value of the spawn variable is less than or equal to 0. If this is true, Lines 214 - 216 will run.

Line 214 creates a new ZOMBIE class object and assigns it the name, “zombie.” The ZOMBIE object will generate at the x coordinate of 9 and a random y coordinate between 0 and 4. This means that the zombie object will always generate in the 9th column but will randomize its row.

Line 215 will append the new zombie object to the zombies list.

Line 216 will reset the spawn delay time for the zombie somewhere between 150 and 200 game cycles.

64. Press ENTER twice.

65. Type the code you see on Lines 218 – 224 of the screenshot below. Ensure that your indentation matches the indentation shown in the screenshot below.

```
212     spawn -= 1
213     if (spawn <= 0):
214         zombie = ZOMBIE(9,random.randint(0,5))
215         zombies.append(zombie)
216         spawn = 150 + random.randint(0,50)
217
218     suntext = font.render("Sun: " +str(sun), True, (0,0,0))
219     scoretext = font.render("Score: " +str(score), True, (0,0,0))
220     screen.fill((0,0,0))
221     screen.blit(backimg, (100,100))
222     screen.blit(panelimg, (325,10))
223     screen.blit(suntext,(330, 20))
224     screen.blit(scoretext,(330, 50))
```

Line 218 creates the suntext variable. The value of the suntext variable is set to be a new rendered font object. The font will say, “Sun: “ plus a string of text that tells the player the value of the sun variable (converting the sun integer variable to string format so it can be displayed in a text string). Anti-alising is turned on (True) meaning that the letters will appear smooth. The font color is black (RGB value of 0, 0, 0).

Line 219 does the same behavior to create a new font object that displays the score on the screen.

Line 220 fills the screen (game window) with black.

Line 221 blits the picture stored in the backimg variable at the location of 100, 100 on the screen (game window).

Line 222 blits the picture stored in the panelimg variable at the location of 325, 10 on the screen (game window).

Line 223 blits the suntext font object onto the screen at the location of 330, 20.

Line 224 blits the scoretext font object onto the screen at the location of 330, 50.

66. Press ENTER.

67. Type the code you see on Lines 225 – 235 of the screenshot below. Ensure your indentation matches what is shown in the screenshot.

```
218     suntext = font.render("Sun: " +str(sun), True, (0,0,0))
219     scoretext = font.render("Score: " +str(score), True, (0,0,0))
220     screen.fill((0,0,0))
221     screen.blit(backimg, (100,100))
222     screen.blit(panelimg, (325,10))
223     screen.blit(suntext,(330, 20))
224     screen.blit(scoretext,(330, 50))
225     #update, draw, action
226     for i in range (0,len(items)):
227         items[i].update()
228         items[i].draw()
229         sun += items[i].action()
230     #clean up
231     for i in range (0,len(items)):
232         if (items[i].health <= 0):
233             empty_squares[(items[i].rect.x-100)//80][(items[i].rect.y-100)//80] = 0
234             items.pop(i)
235             break
```

Line 225 contains a comment.

Line 226 begins a “for” loop that will loop for every item in the items list. This list contains all of the items that have been generated in our game.

Line 227 will update the items in the game (running the update method that we created for each class object).

Line 228 will redraw all of the items onto the screen at the appropriate location.

Line 229 calls the action method that we created. A sunflower object returns a value of 10 when it reaches the panel, while the NUT and SHOOT class objects return a value of 0. This value will be added to the sun variable’s value to increase the sunshine by that amount.

68. Press ENTER.

69. Type the code you see on Lines 230 – 244 of the screenshot below. Ensure your indentation matches the indentation shown in the screenshot.

```
230     #clean up
231     for i in range (0,len(items)):
232         if (items[i].health <= 0):
233             empty_squares[(items[i].rect.x-100)//80][(items[i].rect.y-100)//80] = 0
234             items.pop(i)
235             break
236     #move draw
237     for i in range (0,len(bullets)):
238         bullets[i].move()
239         bullets[i].draw()
240     #clean up
241     for i in range (0,len(bullets)):
242         if (bullets[i].rect.x >= WIDTH):
243             bullets.pop(i)
244             break
```

Line 230 contains a comment.

Line 231 begins another “for” loop that loops through every item in the items list.

Line 232 checks to see if that particular item’s health attribute is less than or equal to 0. If this is true, then Lines 233 – 235 run.

Line 233 changes the value for that item in the empty_squares list to 0, meaning that the square is not empty again.

Line 234 will run the items.pop method to remove that particular class object from the list of items in the game, meaning that the next time the game updates, the object will not be in the items list and will not be drawn onto the screen. The object will disappear from the game window on the next game loop.

Line 235 breaks out of that particular game loop.

Line 236 contains another comment.

Line 237 creates another “for” loop that will loop for each item in the bullets list.

Lines 238 and 239 will move and redraw each bullet object using the move and draw BULLET class methods that we created earlier.

Line 240 contains another comment.

Line 241 will again loop for every item in the bullets list.

Line 242 will check to see if the bullet rect's x position is greater than or equal to the width of the game window (the WIDTH variable). If this is true, the bullet is now off the screen and there is no reason to keep calculating its position since the user can't see it.

Line 243 will remove that particular bullet object from the bullets list.

Line 244 will break out of that particular code loop in the program and move down to the next loop of code.

70. Type the code you see on Lines 245 – 253 of the screenshot below. Ensure your indentation matches what is shown in the screenshot.

```
230     #clean up
231     for i in range (0,len(items)):
232         if (items[i].health <= 0):
233             empty_squares[(items[i].rect.x-100)//80][(items[i].rect.y-100)//80] =
234             items.pop(i)
235             break
236     #move draw
237     for i in range (0,len(bullets)):
238         bullets[i].move()
239         bullets[i].draw()
240     #clean up
241     for i in range (0,len(bullets)):
242         if (bullets[i].rect.x >= WIDTH):
243             bullets.pop(i)
244             break
245     #update, draw
246     for i in range (0,len(zombies)):
247         zombies[i].update()
248         zombies[i].draw()
249         if (zombies[i].rect.x < 100):
250             running = False
251             game_over = True
252             overtext = font.render("GAME OVER", True, (255,255,255))
253             screen.blit(overtext,(40, 40))
```

Line 245 contains another comment.

Line 246 creates a "for" loop that will loop for every item in the zombies list.

Lines 247 and 248 will run the ZOMBIE class object's update and draw methods that we programmed earlier for each item in the list.

Line 249 begins an "if" function that checks to see if the zombie objects rect.x position is less than 100.

If the rect.x position is less than 100, the running variable will be set to False (Line 250), meaning that the game will be over. The zombie object has made it all the way across the game window and the player has lost.

Line 251 will change the value of the game_over variable to true.

Line 252 creates a new variable called overtext. This variable's value is set to be a new font object. The font object will say, "GAME OVER." It's anti-aliasing will be turned on (True), meaning that the text will appear smooth. It will be displayed in white text (RGB code of 255, 255, 255).

Line 253 will blit the overtext font object onto the screen (game window) at the location of 40, 40.

71. Press ENTER.

72. Type the code you see on Lines 254 – 259 of the screenshot below. Ensure your indentation matches the indentation shown in the screenshot.

```
245     #update, draw
246     for i in range (0,len(zombies)):
247         zombies[i].update()
248         zombies[i].draw()
249         if (zombies[i].rect.x < 100):
250             running = False
251             game_over = True
252             overtext = font.render("GAME OVER", True, (255,255,255))
253             screen.blit(overtext,(40, 40))
254     #clean up
255     for i in range (0,len(zombies)):
256         if (zombies[i].health <= 0):
257             zombies.pop(i)
258             score += 50
259             break
```

Line 254 contains a comment.

Line 255 begins another “for” loop that will loop for every item in the zombies list.

Line 256 will check to see if the zombie object’s health attribute value is less than 0. If it is, Lines 257 – 259 will run.

Line 257 will remove that particular zombie class object from the zombies list.

Line 258 will increase the value of the score variable by 50.

Line 259 will break out of the main game loop.

73. Press ENTER.

74. Type the code you see on Lines 260 – 271 of the screenshot below. Ensure your indentation matches the indentation shown in the screenshot.

```
254     #clean up
255     for i in range (0,len(zombies)):
256         if (zombies[i].health <= 0):
257             zombies.pop(i)
258             score += 50
259             break
260     #mouse pointer
261     if (item_selected != 0):
262         (mx,my) = pygame.mouse.get_pos()
263         if ((mx > 100) and (my > 100) and (mx < 900) and (my < 580)):
264             mx = ((mx-100) // 80)
265             my = ((my-100) // 80)
266             if (empty_squares[mx][my] == 0):
267                 mx = (mx * 80) + 100
268                 my = (my * 80) + 100
269                 pygame.draw.rect(screen, (255,255,255), pygame.Rect(mx,my,80,80))
270
271     pygame.display.flip()
272
```

Line 260 contains another comment.

This code of blocks looks similar to something we have done before. However, it will end with something slightly different. This code will draw the actual mouse cursor on the screen.

Line 261 begins an “if” function that checks to see if the item_selected is NOT equal to 0.

If this is true (meaning the item selected is 1, 2, or 3), then a new list using the mx and my (mouse x and mouse y) values that we will assign is created using the mouse cursor’s coordinate positions at the location of the click. This list will be used to identify exactly what the user has clicked.

Line 263 will check to see if the mouse cursor’s x coordinate is greater than 100 and the y coordinate is greater than 100. It will also check to see if the mouse cursor’s x coordinate is less than 900 and y coordinate is less than 580. This line checks to see if the user has actually clicked inside the play area for the game.

If this is true, then a new value for the mx and my variables will be calculated on Lines 264 and 265. This will find the number of the row and column the user has clicked in. Remember our game board is laid out in actual rows and columns. We need to know exactly where the user clicked.

Line 266 will check to see that the square that was selected is empty. If it is, then we will adjust the value of the mx and my variable coordinates and draw a white rectangle at that position (Lines 267 – 269). The rectangle will be drawn at the selected position and have a width and height of 80.

Line 270 is empty.

Line 271 will use the display.flip command to redraw all objects on the screen. The display.flip() command will update the contents of the entire display. In other words, it will redraw all of the sprites and background on your display.

We did run the draw() methods before, and these items redrew the objects at a new position. However, we never updated the entire display to reflect the newly redrawn positions. Line 271 does that.

75. Press ENTER twice.

76. Type the code you see on Lines 273 – 283. Ensure your indentation matches the indentation shown in the screenshot.

```
271     pygame.display.flip()
272
273 #game_over is only true if zombies win, so wait for a quit
274 while game_over == True:
275     clock.tick(10)
276     #get ESCAPE to quit
277     for e in pygame.event.get():
278             if (e.type == pygame.QUIT):
279                 game_over = False
280             if ((e.type == KEYDOWN) and (e.key == K_ESCAPE)):
281                 game_over = False
282
283 pygame.quit()
```

Line 273 contains another comment.

Line 274 begins the next game loop. This loop will run when the game_over variable is changed to True, meaning that the game has ended. However, the game is only over if the zombies have won. The game_over variable only changes to True once the zombies have won. This loop will wait for the user to quit the game by clicking the close button or pressing the ESC key.

Line 275 will set the frame rate for the game to 10 frames per second.

Line 276 contains another comment.

Line 277 checks all of the events in the game's event queue.

Line 278 will check to see if that particular event is a QUIT event, meaning the player has closed out of the game using the "X" button. If this is true, it will change the game_over variable to False.

Line 280 will check to see if the user is currently holding down the ESC key on their keyboard. If this is true, Line 281 will also change the game_over variable to False.

Line 282 is blank.

Line 283 executes the pygame.quit() function to stop all of the scripts currently running in the game.

77. That is it! You have programmed your game!

Full Code:

```
from pygame.locals import *
import random
import pygame

WIDTH = 1000
HEIGHT = 600

backimg = pygame.image.load("Background.png")
panelimg = pygame.image.load("panel.png")

class SF():
    def __init__(self, x, y):
        self.images = []
        self.images.append(pygame.image.load('sf1.png'))
        self.images.append(pygame.image.load('sf2.png'))
        self.images.append(pygame.image.load('sf2.png'))
        self.images.append(pygame.image.load('sf1.png'))
        self.images.append(pygame.image.load('sf1.png'))
        self.images.append(pygame.image.load('sf3.png'))
        self.index = 0
        self.image = self.images[self.index]
        self.rect = pygame.Rect((x*80)+200, (y*80)+100, 65, 80)
        self.health = 80
        self.dropx = -10
        self.dropy = -10
        self.dropxv = 0
        self.dropyv = 0

    def update(self):
        self.index += 1
        if self.index >= len(self.images):
            self.index = 0
        self.image = self.images[self.index]
        for i in range (0,len(zombies)):
            if (self.rect.colliderect(zombies[i].rect)):
                self.health -= 1
                if (self.health > 0):
                    zombies[i].eating = True
                else:
                    zombies[i].eating = False

    def action(self):
        #if drop at panel, set off screen and add sun
        if ((abs(self.dropx - 830) < 100) and (abs(self.dropy - 30) < 80)):
            self.dropx = -10
            self.dropy = -10
            self.dropxv = 0
            self.dropyv = 0
            return 10
        #else if active then move and draw
        elif ((self.dropx != -10) and (self.dropy != -10)):
            self.dropx += self.dropxv
            self.dropy += self.dropyv
            pygame.draw.rect(screen, (255,255,0), pygame.Rect(self.dropx,self.dropy,6,6))
        #else maybe make a new one
        else:

            if (random.randint(0,80) == 0):
                self.dropx = self.rect.x + 40
                self.dropy = self.rect.y + 20
                self.dropxv = (860 - self.dropx) // 50
                self.dropyv = (80 - self.dropy) // 50
        return 0

    def draw(self):
        screen.blit(self.image, (self.rect.x,self.rect.y))

class NUT(SF):
    def __init__(self, x, y):
        super().__init__(x, y)
        self.images = []
        self.images.append(pygame.image.load('nt1.png'))
        self.images.append(pygame.image.load('nt1.png'))
        self.images.append(pygame.image.load('nt1.png'))
        self.images.append(pygame.image.load('nt1.png'))
        self.images.append(pygame.image.load('nt1.png'))
        self.images.append(pygame.image.load('nt1.png'))
        self.images.append(pygame.image.load('nt1.png'))
        self.images.append(pygame.image.load('nt1.png'))
        self.images.append(pygame.image.load('nt1.png'))
        self.health = 250

    def action(self):
        return 0

class SHOOT(SF):
    def __init__(self, x, y):
        super().__init__(x, y)
        self.images = []
        self.images.append(pygame.image.load('sh1.png'))
        self.images.append(pygame.image.load('sh1.png'))
        self.images.append(pygame.image.load('sh1.png'))
        self.images.append(pygame.image.load('sh1.png'))
        self.images.append(pygame.image.load('sh1.png'))
        self.images.append(pygame.image.load('sh1.png'))
        self.images.append(pygame.image.load('sh1.png'))
        self.images.append(pygame.image.load('sh2.png'))
        self.health = 80
        self.reload = 40

    def action(self):
        self.reload -= 1
        if ((self.reload <= 0) and (zombie_on_row(self.rect.y))):
            self.reload = 40
            boom = BULLET(self.rect.x+50, self.rect.y+25)
            bullets.append(boom)
        return 0

class ZOMBIE(SF):
    def __init__(self, x, y):
        super().__init__(x, y)
        self.image = pygame.image.load('zombie1.png')
        self.rect = pygame.Rect((x*80)+190, (y*80)+100, 50, 80)
        self.pause = 15
        self.health = 20
        self.eating = False

    def update(self):
        self.pause -= 1
        if (self.pause <= 0):
            self.pause = 15
            if (self.eating == False):
                self.rect.move_ip(-5,0)

class BULLET:
    #constructor
    def __init__(self, x, y):
        self.rect = pygame.Rect(x,y,10,10)

    #move the bullet
    def move(self):
        self.rect.move_ip(15,0)
        for i in range(0, len(zombies)):
            if (self.rect.colliderect(zombies[i].rect)):
                zombies[i].health -= 2
                self.rect.x = WIDTH
                return

    #draw the bullet
    def draw(self):
        pygame.draw.rect(screen, (255,255,255), pygame.Rect(self.rect.x,self.rect.y,6,6))

def zombie_on_row(row):
    for i in range (0,len(zombies)):
        if (zombies[i].rect.y == row):
            return True
    return False

pygame.init()
screen = pygame.display.set_mode((WIDTH, HEIGHT))
pygame.display.set_caption("Plants vs Zombies")
font = pygame.font.SysFont("Consolasmm", 16)
clock = pygame.time.Clock()

items = []
zombies = []
bullets = []
```

```
                --
sun = 50
score = 0
spawn = 20

#make a sunflower off the grid so player always has some sun
flower = SF(-1,5)
items.append(flower)

item_selected = 0
item1cost = 30
item2cost = 40
item3cost = 40
empty_squares = [[0 for j in range(8)] for i in range(10)]

#main loop
running = True
game_over = False
while running == True:
    clock.tick(10)
    #get ESCAPE to quit
    for e in pygame.event.get():
            if (e.type == pygame.QUIT):
                running = False
            if ((e.type == KEYDOWN) and (e.key == K_ESCAPE)):
                running = False
            if ((e.type == KEYDOWN) and (e.key == K_0)):
                item_selected = 0
            if ((e.type == KEYDOWN) and (e.key == K_1) and (sun >= item1cost)):
                item_selected = 1
            if ((e.type == KEYDOWN) and (e.key == K_2) and (sun >= item2cost)):
                item_selected = 2
            if ((e.type == KEYDOWN) and (e.key == K_3) and (sun >= item3cost)):
                item_selected = 3
            if e.type == pygame.MOUSEBUTTONDOWN:
                if (item_selected != 0):
                    (mx,my) = pygame.mouse.get_pos()
                    if ((mx > 100) and (my > 100) and (mx < 900) and (my < 580)):
                        mx = ((mx - 100) // 50)
                        my = ((my - 100) // 90)
                        if (empty_squares[mx][my] == 0):
                            empty_squares[mx][my] = 1
                            if (item_selected == 1):
                                tmp = SF(mx, my)
                                sun -= item1cost
                            elif (item_selected == 2):
                                tmp = SHOOT(mx, my)
                                sun -= item2cost
                            elif (item_selected == 3):
                                tmp = NUT(mx, my)
                                sun -= item3cost
                            items.append(tmp)
                            item_selected = 0

    spawn -= 1
    if (spawn <= 0):
        zombie = ZOMBIE(9,random.randint(0,8))
        zombies.append(zombie)
        spawn = 130 + random.randint(0,59)

    suntext = font.render("Sun: " +str(sun), True, (0,0,0))
    scoretext = font.render("Score: " +str(score), True, (0,0,0))
    screen.fill((0,0,0))
    screen.blit(backimg, (100,100))
    screen.blit(panelimg, (325,10))
    screen.blit(suntext,(330, 20))
    screen.blit(scoretext,(230, 50))
    #update, draw, action
    for i in range (0,len(items)):
        items[i].update()
        items[i].draw()
        sun += items[i].action()
    #clean up
    for i in range (0,len(items)):
        if (items[i].health <= 0):
            empty_squares[(items[i].rect.x-100)//50][(items[i].rect.y-100)//90] = 0
            items.pop(i)
            break
    #more draw
    for i in range (0,len(bullets)):
        bullets[i].move()
        bullets[i].draw()
    #clean up
    for i in range (0,len(bullets)):
        if (bullets[i].rect.x >= WIDTH):
            bullets.pop(i)
            break
    #update, draw
    for i in range (0,len(zombies)):
        zombies[i].update()
        zombies[i].draw()
        if (zombies[i].rect.x < 100):
            running = False
            game_over = True
            overtext = font.render("GAME OVER", True, (255,255,255))
            screen.blit(overtext,(40, 40))
    #clean up
    for i in range (0,len(zombies)):
        if (zombies[i].health <= 0):
            zombies.pop(i)
            score += 50
            break
    #mouse pointer
    if (item_selected != 0):
        (mx,my) = pygame.mouse.get_pos()
        if ((mx > 100) and (my > 100) and (mx < 900) and (my < 580)):
            mx = ((mx-100) // 50)
            my = ((my-100) // 90)
            if (empty_squares[mx][my] == 0):
                mx = (mx * 50) + 100
                my = (my * 90) + 100
                pygame.draw.rect(screen, (255,255,255), pygame.Rect(mx,my,50,90))

    pygame.display.flip()


#game_over is only true if zombies win, so wait for a quit
while game_over == True:
    clock.tick(10)
    #get ESCAPE to quit
    for e in pygame.event.get():
            if (e.type == pygame.QUIT):
                game_over = False
            if ((e.type == KEYDOWN) and (e.key == K_ESCAPE)):
                game_over = False

pygame.quit()
```