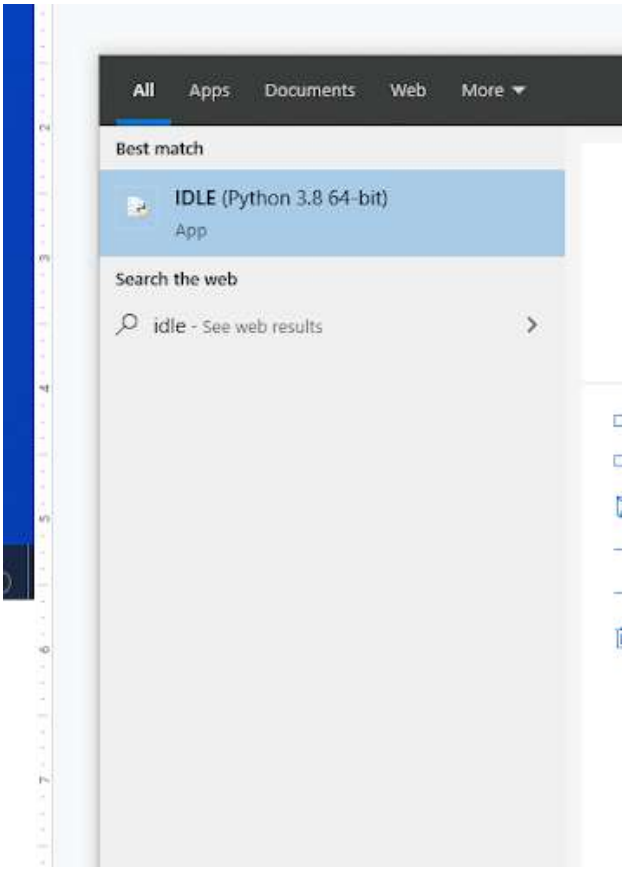
Chapter 6 Practice Directions

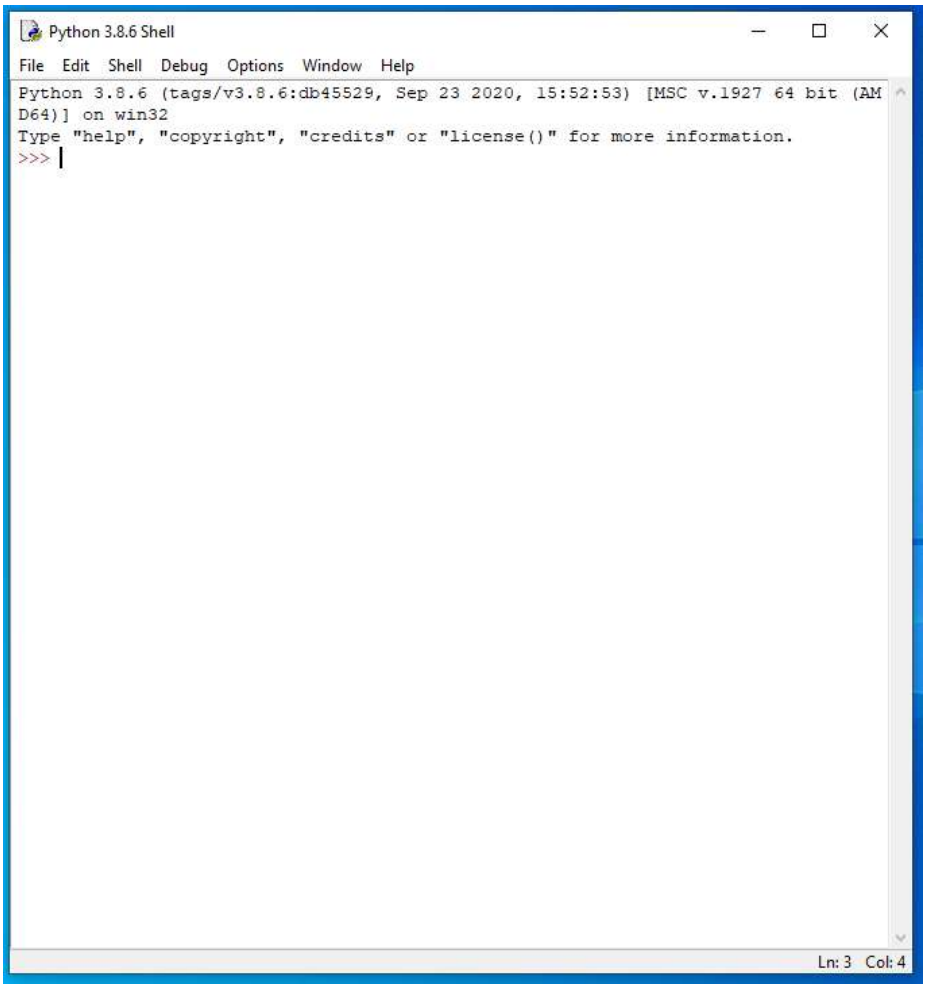
1. . Using your Windows button menu, find and launch your IDLE program.



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.

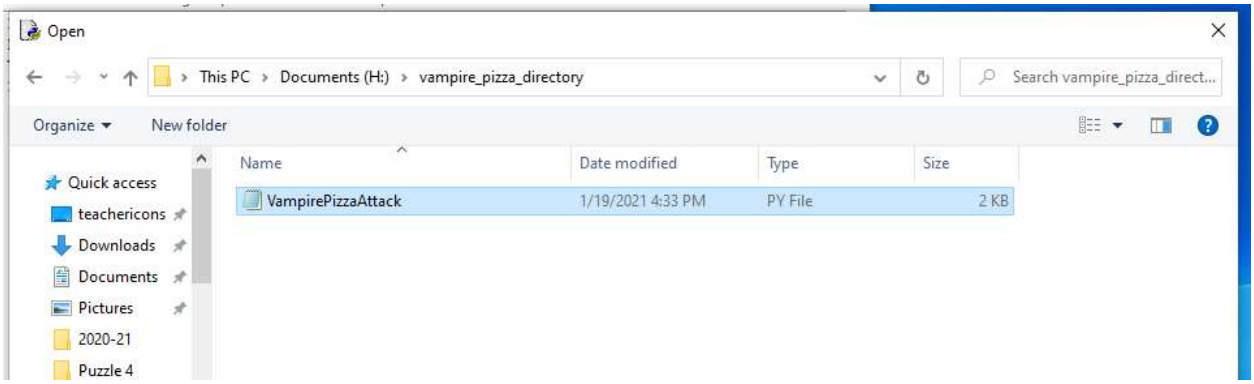
2. Your IDLE window should look something like this once it has launched.:



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.

3. Go to File > Open and then browse to find your VampirePizzaAttack file that we created last chapter and open it.



4. Your Python file and code from last chapter will open up.

5. We will now begin to code the next 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.

6. Click at the end of Line 3.

```
#Import Libraries
import pygame
from pygame import *
#Initialize pygame
pygame.init()
```

7. Press ENTER.

8. Type the code that you see on Line 4 of the screenshot below, then press ENTER:

```
1  #Import Libraries
2  import pygame
3  from pygame import *
4  from random import randint
5
6  #Initialize pygame
7  pygame.init()
```

   The code on Line 4 above imports the randint function from the random module. The randint function will allow us to let the computer choose a random integer from a certain range of numbers. In other words, it lets us randomize numbers within a program.

9. Click at the end of Line 22.

```
#Define colors
WHITE = (255, 255, 255)

#-----------------------------------------------------------------
#Load assets
```

10. Press ENTER twice.

11. Type the code that you see on Lines 24 – 25 of the screenshot below, pressing ENTER afterwards to add a blank line at the end.

```
21 #Define colors
22 WHITE = (255, 255, 255)
23
24 #Set up rates
25 SPAWNRATE = 360
26
27 #-----------------------------------------------------------------
28 #Load assets
```

    Line 24 contains another comment describing what the lines of code in this block do.

    Line 25 creates a constant variable called SPAWNRATE and sets its value to be equal to 360.

12. Modify the display caption on Line 32 to say, “Vampire Pizza”

```
#-------------------------------------------------------------
#Load assets

#Create window
GAME_WINDOW = display.set_mode(WINDOW_RES)
display.set_caption('Vampire Pizza')
```

13. Add a set of parentheses around the “WINDOW_RES” specification in the scale settings for the BACKGROUND variable on Line 37. We are just cleaning our code up a bit. The scale would have worked fine without doing this, but since the resolution contains two different numbers (the height and the width in pixels), it is good form to put the variable in parentheses to make sure that both of those numbers are recognized by the computer.

```
display.set_caption('Vampire Pizza')

#Set up the background image
background_img = image.load('restaurant.jpg')
background_surf = Surface.convert_alpha(background_img)
BACKGROUND = transform.scale(background_surf, (WINDOW_RES))
```

14. Click at the end of Line 44, as shown in the screenshot below.

```
#Set up the enemy image
#Load the image into the program
pizza_img = image.load('vampire.png')
#Convert the image to a surface
pizza_surf = Surface.convert_alpha(pizza_img)
VAMPIRE_PIZZA= transform.scale(pizza_surf, (WIDTH, HEIGHT))

#-------------------------------------------------------------
```

15. Press ENTER twice.

16. Type the code that you see on Lines 46 – 49 of the screenshot below.

```
44 VAMPIRE_PIZZA= transform.scale(pizza_surf, (WIDTH, HEIGHT))
45
46 #-------------------------------------------------------------
47 #Set up classes
48 #Create an enemy class
49 class VampireSprite(sprite.Sprite):
```

Line 46 contains a comment separator.

Lines 47 and 48 contains comments describing what the next section of code will do.

Line 49 creates a new class called VampireSprite. The VampireSprite class is a new class of sprites. Whatever we create under this class will be a sprite. So, the VampireSprite class is based off of the Sprite superclass, and any of the elements under it will be of the sprite type.

In Pygame, a sprite can function as a class of objects that can have special features built in. Essentially, we are creating a VampireSprite class based off of the Sprite superclass type, and any class object generated within the VampireSprite class will be a sprite object.

17. Press ENTER twice.

18. Type the code that you see on Lines 51 – 59 of the screenshot below.

```
46  #------------------------------------------------------------
47  #Set up classes
48  #Create an enemy class
49  class VampireSprite(sprite.Sprite):
50
51      #This function creates an instance of the enemy
52      def __init__(self):
53          super().__init__()
54          self.speed = 2
55          self.lane = randint (0, 4)
56          all_vampires.add(self)
57          self.image = VAMPIRE_PIZZA.copy()
58          y = 50 + self.lane * 100
59          self.rect = self.image.get_rect(center = (1100, y))
```

The code on Line 51 contains a comment describing what the next block of code does.

The code on Line 52 contains a def __init__(self): method. Remember, we learned last chapter that the __init__ method creates rules for setting up (or initiating) new instances of VampireSprite class objects. In this case, each sprite generated until the VampireSprite class will be a new vampire pizza sprite. The __init__ method only takes one default argument, which is self. Essentially, you are telling any newly created objects under the VampireClass to follow the rules in the code for setting themselves up in the program. The object “self” is a placeholder we use when we want to create an attribute that works with every instance of a class.

Line 53 contains a super().__init__() method. The super() method tells the program that we want to use all of the built-in rules for the superclass of the sprite. In this case, the superclass is Sprite. Anything that we add afterwards in the rest of the code is in addition to the default rules that are programmed in the superclass.

Line 54 creates a new variable called self.speed and sets the speed of all sprites in this class to 2.

Line 55 creates another variable called self.lane and uses the randint function to generate a random integer between the numbers 0 and 4.

Line 56 adds the newly generated sprite object to the all_vampires group that we will create later.

Line 57 creates the self.image variable. The self.image variable’s value will be a copy of the VAMPIRE_PIZZA image. You want to copy the image because you will want more than one vampire pizza sprite in your game at a time using that image. If you didn’t copy it, then whenever new sprites generate, the older sprites would lose their image on the

screen and become invisible. Creating a new copy of the image on the screen for each new sprite would avoid this program.

Line 58 creates a y variable and sets its value to be whatever coordinate position it needs to have to be in the middle of the selected lane. Remember, the lane is a random number between 0 and 4. The math function for finding the y-coordinate was mainly generated by trial and error to see where different numbers generated the sprite at.

Line 59 creates a self.rect variable. This line creates a rect (rectangle) for each sprite and places the rectangle just off the right side of the screen in the correct lane it is supposed to be on. The rectangle is then filled with the self.image picture, which is the VAMPIRE_PIZZA image. All sprites on Pygame need a rect and a surface to move on. Unless otherwise specified, every rect will generate at the x- and y-coordinates of 0, 0 and place the sprite and any other elements (the sprite's picture, etc.) inside of it. To change the position the rectangle generates at, you would need to manually enter x- and y-coordinates for each rect, which is what we have done at the end of Line 59 in parentheses. The end of Line 59 tells the rect to center around the x-coordinate of 1100 (just off the screen) and the y coordinate of whatever we calculated in Line 58 (the value of the y variable).

19. Press ENTER twice.

20. Type the code that you see on Lines 61 – 63 of the screenshot below.

```
56          all_vampires.add(self)
57          self.image = VAMPIRE_PIZZA.copy()
58          y = 50 + self.lane * 100
59          self.rect = self.image.get_rect(center = (1100, y))
60
61      #This function moves the enemies from right to left and destroys them after they've left the screen
62      def update(self, game_window):
63          game_window.blit(self.image, (self.rect.x, self.rect.y))
```

Line 61 creates a comment describing what the code below it does.

Line 62 creates another method called update. This method will use the self and the game_window arguments to run.

Line 63 blits the new sprite to the game_window, meaning that it reloads the game_window to display the new sprite object instance that was just created. When it reloads, the newest sprite will display using its image and the x- and y-coordinates of the sprite's rect.

21. Press ENTER twice.

22. Type the code that you see on Lines 65 – 69 of the screenshot below.

```
62      def update(self, game_window):
63          game_window.blit(self.image, (self.rect.x, self.rect.y))
64
65  #--------------------------------------------------------------
66  #Create class instances
67
68  #create a sprite group for all VampireSprite instances
69  all_vampires = sprite.Group()
```

Lines 65 - 68 create comments describing what the code in the following block does.

Line 69 creates the all_vampires variable and sets its value to be equal to a group of sprites. This group will hold all instances of the VampireSprite class because each time a new instance is generated, it adds itself to the group (using the .add method on Line 56.

23. Remove the two lines highlighted in gray in the screenshot below. They appear on Lines 84 – 85 of your code. Just backspace them or delete the two lines to get rid of the code.

```
78      for column in range(11):
79          draw.rect(BACKGROUND, tile_color, (WIDTH*column, HEIGHT*row, WIDTH, HEIGHT), 1)
80
81  #Display the background image to the screen
82  GAME_WINDOW.blit(BACKGROUND, (0, 0))
83
84  #Display the enemy image on the screen
85  GAME_WINDOW.blit(VAMPIRE_PIZZA, (900, 400))
```

24. Click at the end of Line 98.

```
        #Exit loop on quit
        if event.type == QUIT:
            game_running = False
```

25. Press ENTER twice.

26. Type the code that you see on Lines 100 - 103 Of the screenshot below.

```
98              game_running = False
99
100 #-------------------------------------------------------------
101 #Create VampireSprite instances
102     if randint (1, SPAWNRATE) == 1:
103         VampireSprite()
```

Line 101 contains a comment describing what the code does.

Lines 102 – 103 contain an if function that checks to determine if a random number generated using the randint function is equal to 1. The random number generated will be between 1 and the value of the SPAWNRATE variable, which is currently 360. If the number generated is 1, it will generate a new instance of VampireSprite.

27. Click after your “#Update display.” comment and press ENTER.

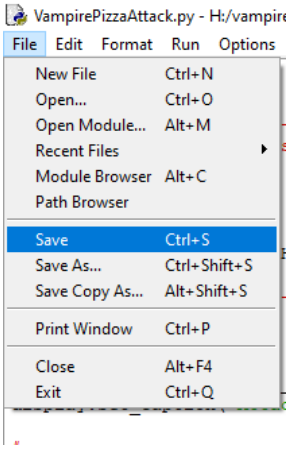
```
#-------------------------------------------------------------
    #Update display.

    display.update()

#Close main game loop
#-------------------------------------------------------------
```

28. Type the code that you see on Lines 107 - 109 of the screenshot below.

```
105 #-------------------------------------------------------------
106     #Update display.
107     for vampire in all_vampires:
108         vampire.update(GAME_WINDOW)
109
110     display.update()
111
112 #Close main game loop
113 #-------------------------------------------------------------
```

Line 107 begins a “for” loop that will update the location of each vampire pizza sprite once every iteration of the game loop runs. In other words, this loop will go through each sprite in the all_vampires group and update its location in the game window.

29. Go to File > Save. Before we test our game, we need to save it.



30. Now, go to Run > Run Module.

31. After giving your game window time to load, you should see your background image in your game window with a grid of white lines overlaid on it. You should also see pizza vampire sprite images start to generate on the right-hand size of your screen.

32. You can close out of the Python file. You can also close out of the Python Shell if you still have it open.

Final Code:

```
#Import Libraries
import pygame
from pygame import *
from random import randint

#Initialize pygame
pygame.init()

#-----------------------------------------------------------------
#Define constant variables

#Define the parameters of the game window
WINDOW_WIDTH = 1100
WINDOW_HEIGHT = 600
WINDOW_RES = (WINDOW_WIDTH, WINDOW_HEIGHT)

#Define the tile parameters
WIDTH = 100
HEIGHT = 100

#Define colors
WHITE = (255, 255, 255)

#Set up rates
SPAWNRATE = 360

#-----------------------------------------------------------------
#Load assets

#Create window
GAME_WINDOW = display.set_mode(WINDOW_RES)
display.set_caption('Vampire Pizza')

#Set up the background image
background_img = image.load('restaurant.jpg')
background_surf = Surface.convert_alpha(background_img)
BACKGROUND = transform.scale(background_surf, (WINDOW_RES))
```

```
#Set up the enemy image
#Load the image into the program
pizza_img = image.load('vampire.png')
#Convert the image to a surface
pizza_surf = Surface.convert_alpha(pizza_img)
VAMPIRE_PIZZA= transform.scale(pizza_surf, (WIDTH, HEIGHT))

#-------------------------------------------------------------
#Set up classes
#Create an enemy class
class VampireSprite(sprite.Sprite):

    #This function creates an instance of the enemy
    def __init__(self):
        super().__init__()
        self.speed = 2
        self.lane = randint (0, 4)
        all_vampires.add(self)
        self.image = VAMPIRE_PIZZA.copy()
        y = 50 + self.lane * 100
        self.rect = self.image.get_rect(center = (1100, y))

    #This function moves the enemies from right to left and destroys them after they've left the screen
    def update(self, game_window):
        game_window.blit(self.image, (self.rect.x, self.rect.y))

#-------------------------------------------------------------
#Create class instances

#create a sprite group for all VampireSprite instances
all_vampires = sprite.Group()

#-------------------------------------------------------------
#Initialize and draw the background grid

#Define the color of the grid outline
tile_color = WHITE

#Populate the background grid
for row in range(6):
    for column in range(11):
        draw.rect(BACKGROUND, tile_color, (WIDTH*column, HEIGHT*row, WIDTH, HEIGHT), 1)
```

```
#Display the background image to the screen
GAME_WINDOW.blit(BACKGROUND, (0, 0))

#------------------------------------------------------------
#Start main game loop

#Game loop
game_running = True
while game_running:

#------------------------------------------------------------
#Check for events

    #Checking for and handling events
    for event in pygame.event.get():
        #Exit loop on quit
        if event.type == QUIT:
            game_running = False

#------------------------------------------------------------
#Create VampireSprite instances
    if randint (1, SPAWNRATE) == 1:
        VampireSprite()

#------------------------------------------------------------
    #Update display.
    for vampire in all_vampires:
        vampire.update(GAME_WINDOW)

    display.update()

#Close main game loop
#------------------------------------------------------------

#Clean up game
pygame.quit()
```