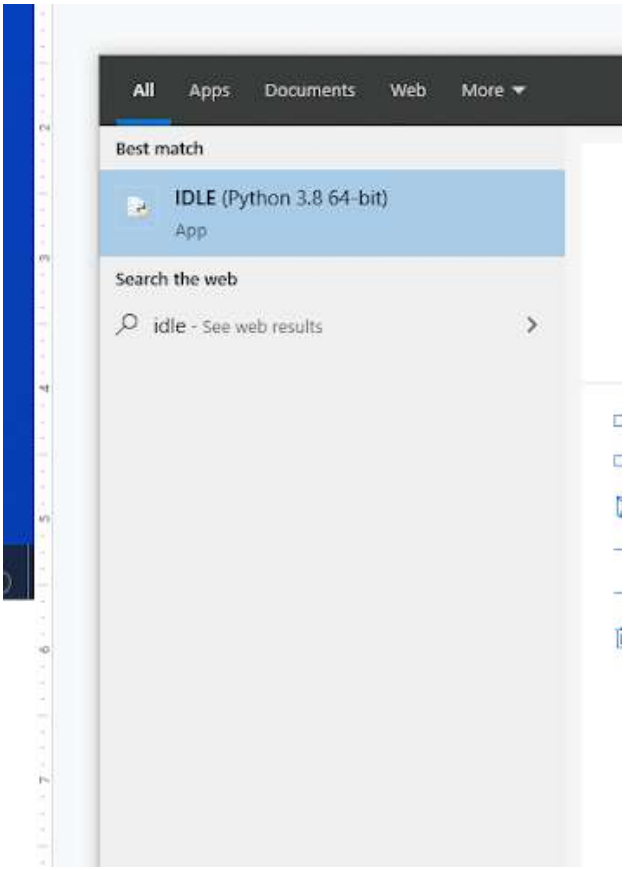
Chapter 8 Practice Directions

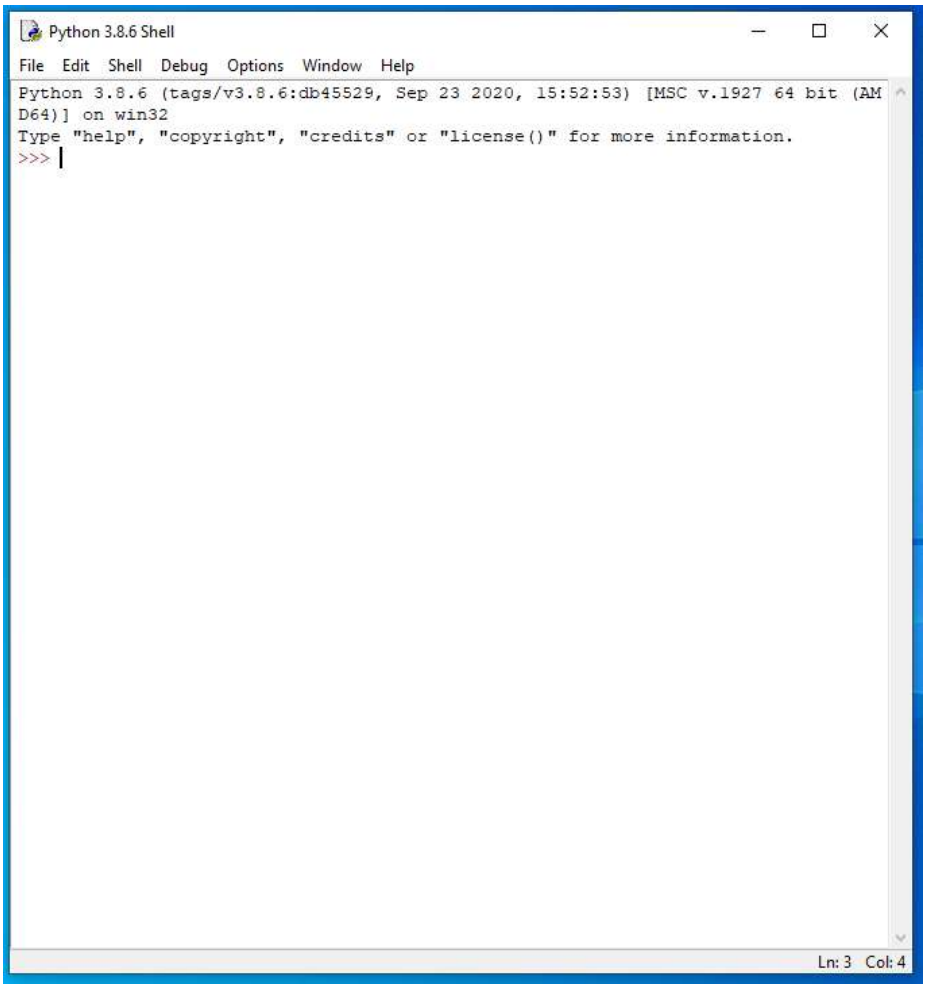
1. . Using your Windows button menu, find and launch your IDLE program.



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.

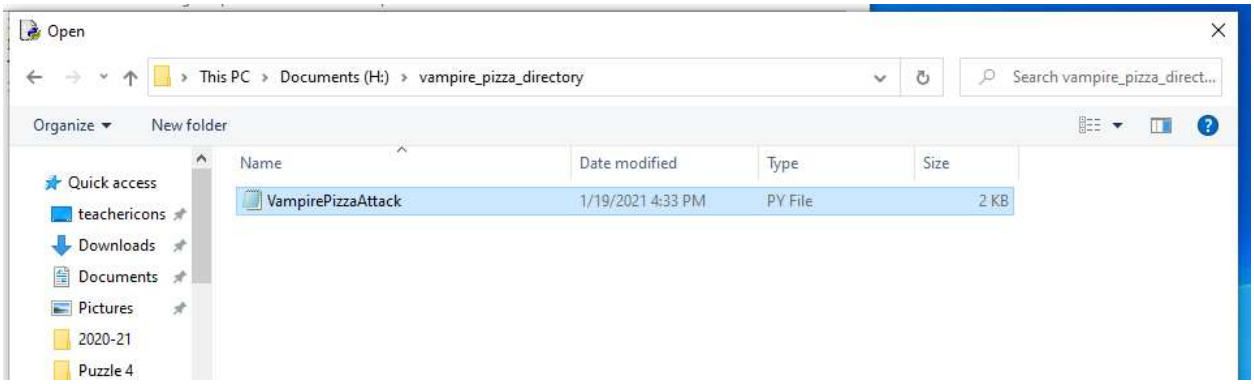
2. Your IDLE window should look something like this once it has launched.:



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 73.

```
    #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(BACKGROUND, (self.rect.x, self.rect.y), self.rect)
        self.rect.x -= self.speed
        game_window.blit(self.image, (self.rect.x, self.rect.y))
```

7. Press ENTER twice.

8. Type the code that you see on Lines 75 - 79 of the screenshot below, then press ENTER:

```
75  #Create a class of sprites. Each tile has an invisible interactive field attached to it which is a sprite in this class.
76  class BackgroundTile(sprite.Sprite):
77      def __init__(self):
78          super().__init__()
79          self.effect = False
```

   The code on Lines 75 contains a comment.

   Line 76 creates another subclass called BackgroundTile. The subclass will create sprite objects based off of the Sprite superclass that comes preprogrammed in Pygame.

   Line 77 contains the code that tells the program you are going to define how the new sprites generated under the BackgroundTile subclass will be set up.

   Line 7 states that you want this subclass to keep all of the default behaviors and settings from the Sprite superclass.

   Line 80 creates an instance attribute for each new sprite created under this class called effect. The value of the effect instance attribute is set to False. The effect attribute will change to true if a tile has a trap applies to it to trap vampire pizzas.

9. Click at the end of Line 90, as shown in the screenshot below.

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

10. Press ENTER twice.

11. Type the code that you see on Lines 92 – 93 of the screenshot below.

```
89  #------------------------------------------------------------
90  #Initialize and draw the background grid
91
92  #Create an empty list to hold the tile grid
93  tile_grid=[]
```

Line 92 contains a comment.

Line 93 creates a list variable called tile_grid and sets its value to be an empty list. Lists are denoted by using square brackets [ ] at the beginning and end of the list. Items within a list are separated by commas. List variables allow you to make lists of many different items. Each item in that list can be referenced using the list variable time and the item's index number (position within the list).

12. Click at the end of Line 96, as shown in the screenshot below.

```
#Populate the background grid
for row in range(6):
```

13. Press ENTER.

14. Type the code that you see on Lines 97 – 98 of the screenshot below.

```
95  #Populate the background grid
96  for row in range(6):
97      row_of_tiles = []
98      tile_grid.append(row_of_tiles)
99      for column in range(11):
```

Line 97 creates another list variable called row_of_tiles and sets its value to be an empty list.

Line 98 will append the value of the row_of_tiles list variable into the tile_grid list variable that we created earlier on Line 93. In other words, the tile_grid list variable will be made up of a bunch of smaller lists that list the tiles in each row.

15. Click at the end of Line 99, as shown in the screenshot below:

```
#Populate the background grid
for row in range(6):
    row_of_tiles = []
    tile_grid.append(row_of_tiles)
    for column in range(11):
```

16. Press ENTER.

17. Type the code that you see on Lines 100 – 102 of the screenshot below.

```
99      for column in range(11):
100         new_tile = BackgroundTile()
101         new_tile.rect = pygame.Rect(WIDTH * column, HEIGHT * row, WIDTH, HEIGHT)
102         row_of_tiles.append(new_tile)
103         draw.rect(BACKGROUND, tile_color, (WIDTH*column, HEIGHT*row, WIDTH, HEIGHT), 1)
```

Line 100 creates a new variable called new_tile and assigns that variable to a new sprite object instance created using the BackgroundTile class.

Line 101 creates a new_tile.rect attribute for the new object that was created and sets its value to be an invisible rectangle using the width and height variables and the column and row numbers.

Line 102 appends the new_tile created to the row_of_tiles list that you created on Line 97.

18. Click at the end of Line 122, as shown in the screenshot below.

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

19. Press ENTER twice.

20. Type the code that you see on Lines 104 – 109 of the screenshot below:

```
102                 game_running = False
103
104             #Set up the background tiles to respond to a mouse click
105             elif event.type == pygame.MOUSEBUTTONDOWN:
106                 x, y = pygame.mouse.get_pos()
107                 tile_grid[y//100][x//100].effect = True
108                 print(x, y)
109                 print('You clicked me!')
```

Line 104 contains a comment.

Line 105 creates an elif statement that checks if an event has happened. Elif stands for "else if." If the specific event it is checking for happens, the code in the block will execute. If the event doesn't occur, the code will be skipped. In this case, the "elif" statement is testing for whether the user has clicked the mouse button (the MOUSEBUTTONDOWN event).

The code in Lines 106 – 109 will execute if the user presses the mouse button.

Line 106 will create variables called x and y and assign their values to the x and y coordinate positions of wherever the user has clicked.

Line 107 will find the background tile that is at the location where the mouse button was clicked. It will change the effect attribute of that particular tile to True, meaning that the user has now placed a trap at that particular location. The "//" symbols tell the program to divide.

Lines 108 and 109 contain print statements that will print the values of the x and y variable and also a statement saying "You clicked me!".

21. Click at the end of Line 114, as shown in the screenshot below.

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

22. Press ENTER twice.

23. Type the code that you see on Lines 116 – 121 of the screenshot below.

```
116 #------------------------------------------------------------
117 #Set up collision detection
118     #draw a background grid
119     for tile_row in tile_grid:
120         for tile in tile_row:
121             GAME_WINDOW.blit(BACKGROUND, (tile.rect.x, tile.rect.y), tile.rect)
```

    Line 116 contains a comment separator.

    Lines 117 and 118 contain more comments.

    Line 119 starts a “for” loop what will loop for each item/list in the tile_grid list variable. Remember, the tile_grid list variable is made up of smaller lists from the row_of_tiles list variable.

    Line 120 will loop for each tile in each of the the tile_row lists.

    Line 121 will blit the background grid to the specified location. It will blit the BACKGROUND image to the location of the tile.rect.x and tile.rect.y variables. The blitted image will take up the area specified by the tile.rect specifications.

24. Press ENTER.

25. Go to File > Save to save your project.

26. Go to Run > Run Module. You shouldn’t see much happening since we haven’t programmed the print statements to display on the screen or the traps to load. However, in the Python Shell, you should see the print statements pop up with the x and y coordinates of whatever tile you clicked, along with the “You clicked me!” print statement.

27. You may now close out of the game window and IDLE.

Final Code:

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

#Initialize pygame
pygame.init()

#set up clock
clock = time.Clock()

#-----------------------------------------------------------------
#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
FRAMERATE = 60

#-----------------------------------------------------------------
#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(BACKGROUND, (self.rect.x, self.rect.y), self.rect)
        self.rect.x -= self.speed
        game_window.blit(self.image, (self.rect.x, self.rect.y))

 #Create a class of sprites. Each tile has an invisible interactive field attached to it which is a sprite in this class.
class BackgroundTile(sprite.Sprite):
    def __init__(self):
        super().__init__()
        self.effect = False

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

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

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

```
#Initialize and draw the background grid

#Create an empty list to hold the tile grid
tile_grid=[]

#Define the color of the grid outline
tile_color = WHITE

#Populate the background grid
for row in range(6):
    row_of_tiles = []
    tile_grid.append(row_of_tiles)
    for column in range(11):
        new_tile = BackgroundTile()
        new_tile.rect = pygame.Rect(WIDTH * column, HEIGHT * row, WIDTH, HEIGHT)
        row_of_tiles.append(new_tile)
        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

        #Set up the background tiles to respond to a mouse click
```

```
        elif event.type == pygame.MOUSEBUTTONDOWN:
            x, y = pygame.mouse.get_pos()
            tile_grid[y//100][x//100].effect = True
            print(x, y)
            print('You clicked me!')

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

#----------------------------------------------------------------
#Set up collision detection
    #draw a background grid
    for tile_row in tile_grid:
        for tile in tile_row:
            GAME_WINDOW.blit(BACKGROUND, (tile.rect.x, tile.rect.y), tile.rect)

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

    display.update()

    #set the framerate
    clock.tick(FRAMERATE)

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

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