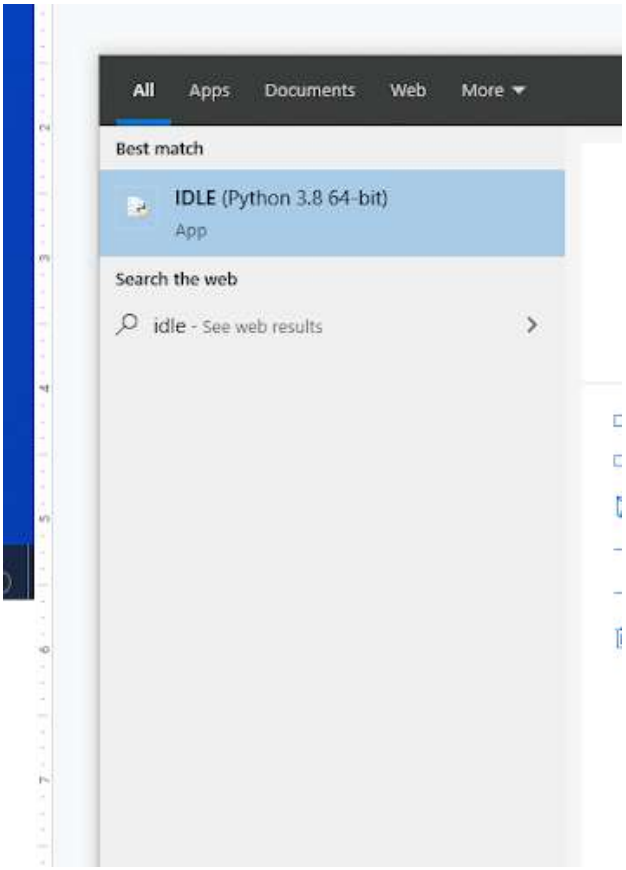
Chapter 4 Practice Directions

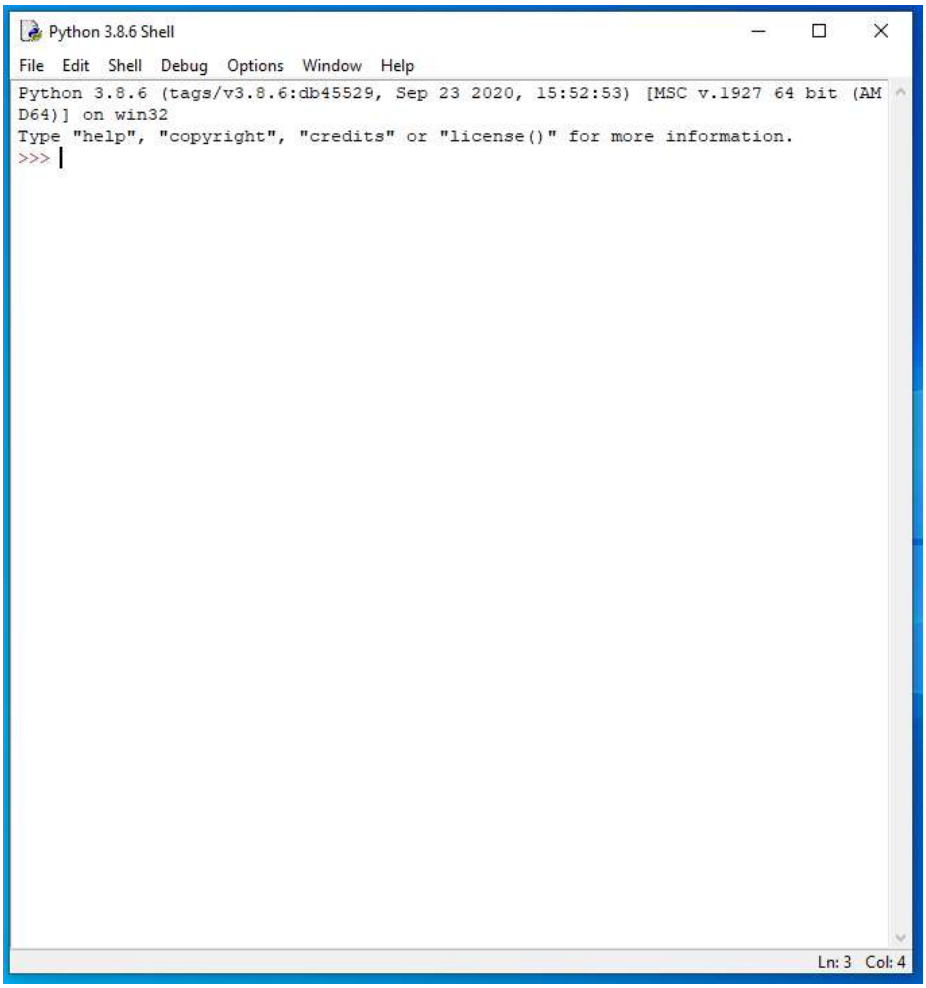
1. . Using your Windows button menu, find and launch your IDLE program.



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.

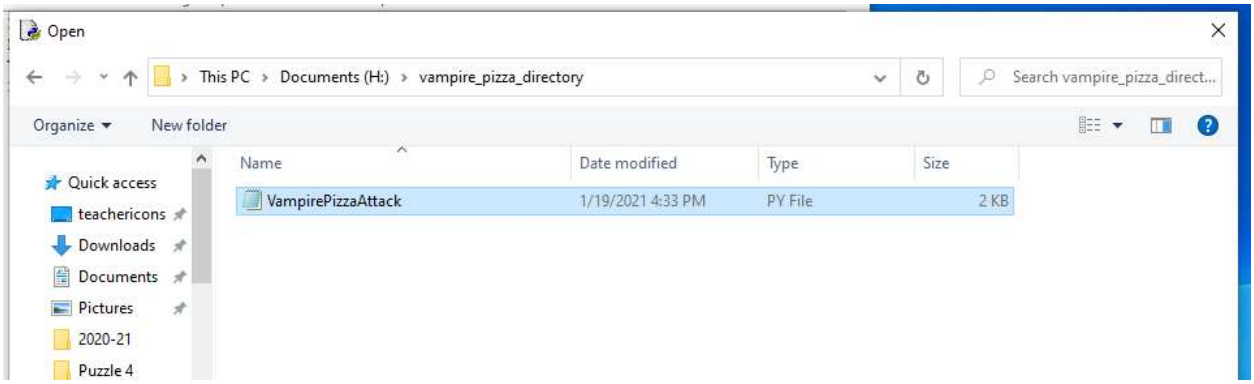
2. Your IDLE window should look something like this once it has launched.:



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 13 after the code to set the display window resolution.

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

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

7. Press ENTER twice.

8. Type the code that you see on Lines 15 – 17 of the screenshot below:

```
13 WINDOW_RES = (WINDOW_WIDTH, WINDOW_HEIGHT)
14
15 #Define the tile parameters
16 WIDTH = 100
17 HEIGHT = 100
```

In our game, we want to be able to know what parts of the tile background our vampire pizzas are passing over. That way, we can make some parts safe and other parts traps for the pizzas. To do this, there will be two parts of our grid: A part that we can see and a part that we cannot see.

We are going to create grid lines that overlay the background image. This is the part that we can see. Then, we will create a layer of the grid that makes it interactive. This is the part of the grid that is important because it is the part that the other game components, like pizzas, can interact with. This will let the player set parts of it as safe while using other parts of the grid as trap zones.

In the code above, we begin the process of creating an overlay grid for our game.

Line 15 contains a comment descripting what Lines 16 and 17 will do.

Line 16 creates a constant variable called WIDTH and sets it to be equal to 100 pixels.

Line 17 creates a constant variable called HEIGHT and sets it to be equal to 100 pixels.

We want the tiles for our grid to be squares that are 100 pixels high x 100 pixels wide.

9. Press ENTER twice.

10. Type the code that you see on Lines 19 – 20 below.

```
15 #Define the tile parameters
16 WIDTH = 100
17 HEIGHT = 100
18
19 #Define colors
20 WHITE = (255, 255, 255)
```

Line 19 contains a comment describing what the next line of code will do.

Line 20 creates a constant variable called WHITE. The variable is set to the RGB color code for white. RGB color codes for virtually any color can be found online. Each code contains numbers for the amount of red, green, and blue in any color shade. The

numbers range from 0 – 255. Since white is virtually the combination of all color shades, its RGB code is 255, 255, 255.

11. Press ENTER twice.

12. Locate the scale settings of the pizza_surf image on Line 39.

```
33
34  #Set up the enemy image
35  #Load the image into the program
36  pizza_img = image.load('vampire.png')
37  #Convert the image to a surface
38  pizza_surf = Surface.convert_alpha(pizza_img)
39  VAMPIRE_PIZZA= transform.scale(pizza_surf, (100, 100))
```

13. Change the scale settings of 100, 100 to WIDTH and HEIGHT so that the pizza image is scale to fit exactly into a grid tile.

```
33
34  #Set up the enemy image
35  #Load the image into the program
36  pizza_img = image.load('vampire.png')
37  #Convert the image to a surface
38  pizza_surf = Surface.convert_alpha(pizza_img)
39  VAMPIRE_PIZZA= transform.scale(pizza_surf, (WIDTH, HEIGHT))
```

14. Click at the end of Line 39 and press ENTER twice.

15. Type the code that you see on Lines 41 – 45 of the screenshot below.

```
39  VAMPIRE_PIZZA= transform.scale(pizza_surf, (WIDTH, HEIGHT))
40
41  #-------------------------------------------------------------
42  #Initialize and draw the background grid
43
44  #Define the color of the grid outline
45  tile_color = WHITE
```

Line 41 contains another code separator.

Lines 42 and 44 contains comments describing what the following lines of code will do.

Line 45 creates a variable called tile_color and sets its value to be equal to the value of the WHITE constant variable, which contains the RGB color code for white. In other words, we create a variable called tile_color and sets its value to white.

16. Press ENTER twice.

17. Type the code that you see on Lines 47 – 50 of the screenshot below. Remember that indentation is important.

```
41  #-------------------------------------------------------------
42  #Initialize and draw the background grid
43
44  #Define the color of the grid outline
45  tile_color = WHITE
46
47  #Populate the background grid
48  for row in range(6):
49      for column in range(11):
50          draw.rect(BACKGROUND, tile_color, (WIDTH*column, HEIGHT*row, WIDTH, HEIGHT), 1)
```

Line 47 contains a comment describing what the following lines of code will do.

Line 48 creates a “for” loop. The lines of code under this loop will loop 6 times since our board has 6 rows. The variable of “row” in this loop is essentially just a placeholder. Line 48 is basically saying, “for **code** in range(6):”. It doesn’t matter what placeholder you put as the name of the variable in your loop code. However, whatever variable name you use will be referenced later in the loop, so its good practice to name it something that makes sense.

Line 49 creates a nested for loop. This loop will loop 11 times since each row on our game board will have 11 columns.

Line 50 contains the draw.rect function, which will draw our rectangles that make up our tile grid. The first argument in the draw.rect function specified the surface that we want the rectangle to draw on. In this case, we want the grid rectangle to draw on the BACKGROUND surface.

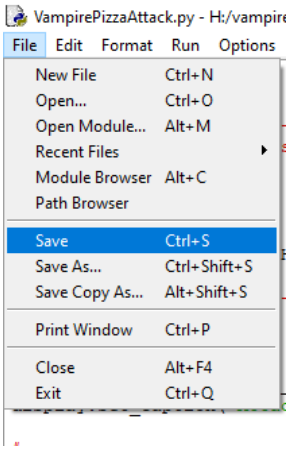
The second argument in the draw_rect function contains the color that we would like the rectangle outline to be. In this case, we want our rectangle outline to be set to the tile_color variable, which is white.

The third argument specifies the x and y coordinate locations along with the width and the height of the rectangle. The x coordinate is set to the width variable (which is 100) times the column that the loop is currently in. So, if the loop is working on generating a tile for the third column, it would take the width variable (100) times 3 (for the third column) and start generating the rectangle at the x location of 300. The y location coordinate is set to the height variable (which is also 100) times the current row that is being generated. If the loop is working on row 5, it would take the row number (5) times the height (100) to generate the rectangle at the y location of 500.

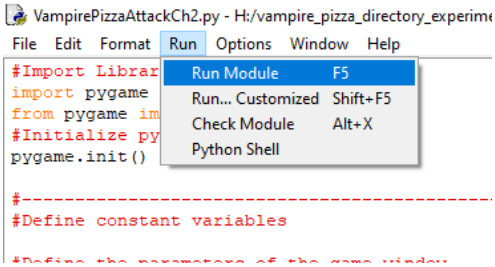
The WIDTH and the HEIGHT variables in the third argument specify how large the rectangle will be (its width and its height).

The last argument specified the thickness of the rectangle border, which is set to 1 pixel.

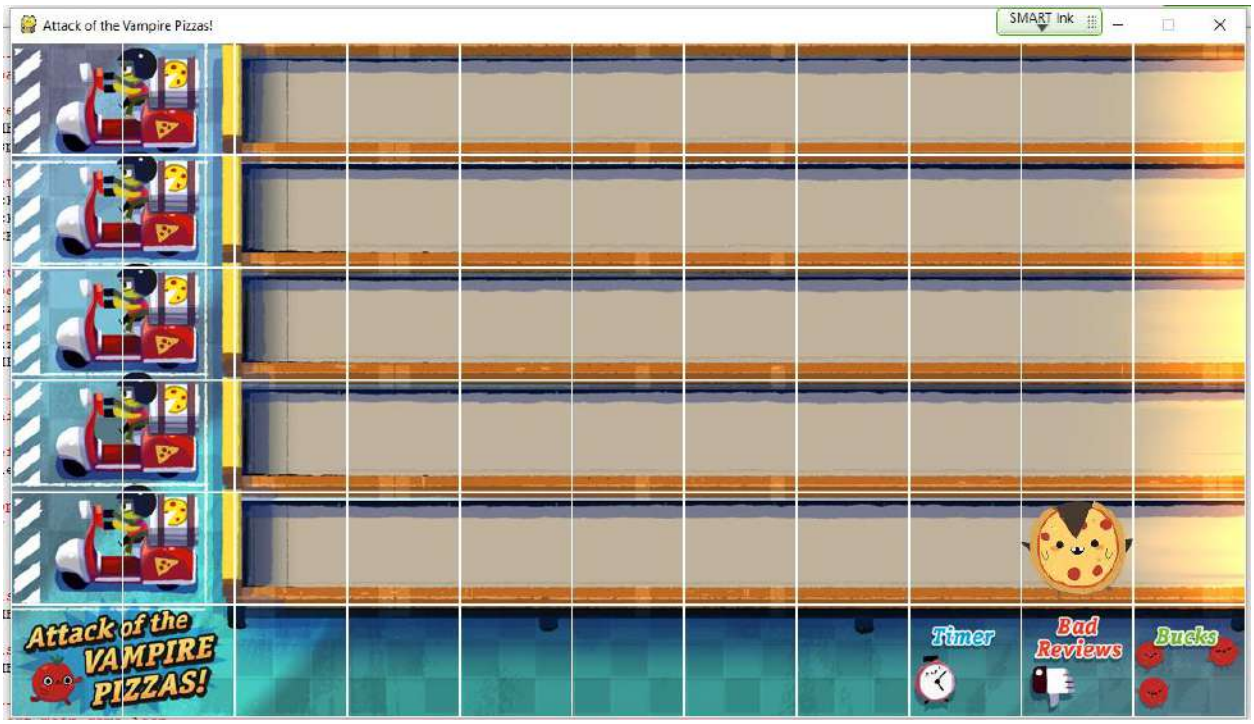
18. Go to File > Save. Before we test our game, we need to save it.



19. Now, go to Run > Run Module.



20. 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.



21. 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 *
#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)

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

#Create window
GAME_WINDOW = display.set_mode(WINDOW_RES)
display.set_caption('Attack of the Vampire Pizzas!')

#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))

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

```
#-------------------------------------------------------------
#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))

#Display the enemy image on the screen
GAME_WINDOW.blit(VAMPIRE_PIZZA, (900, 400))

#-------------------------------------------------------------
#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

#-------------------------------------------------------------
    #Update display.
    display.update()

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

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