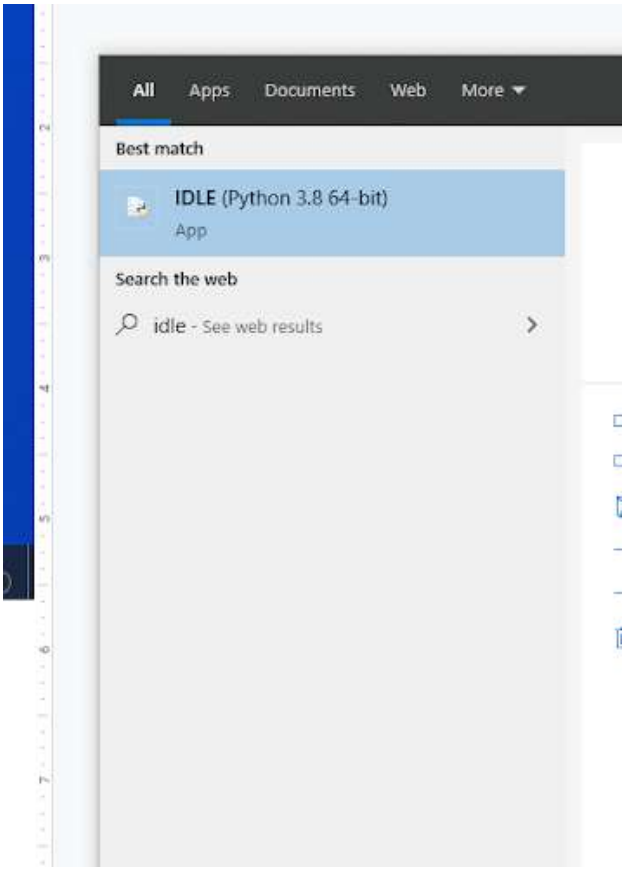
## Chapter 12 Practice Directions

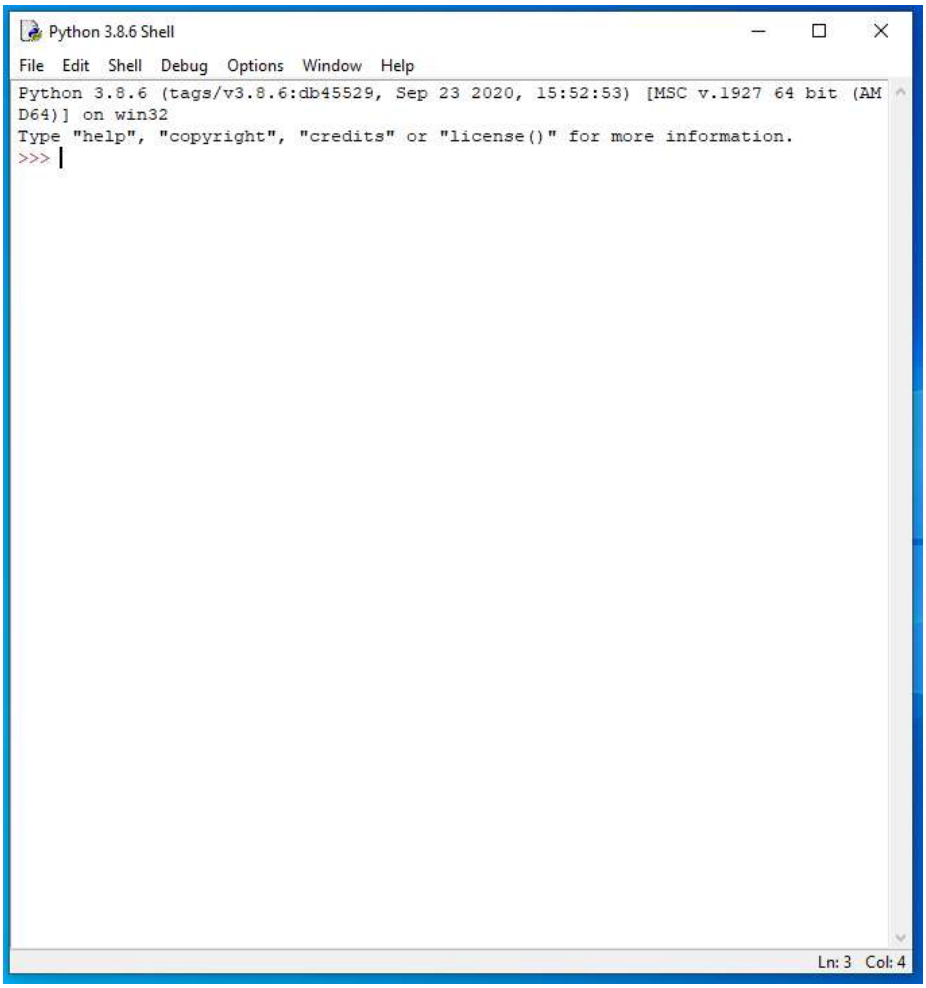
1. Using your Windows button menu, find and launch your IDLE program.




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.

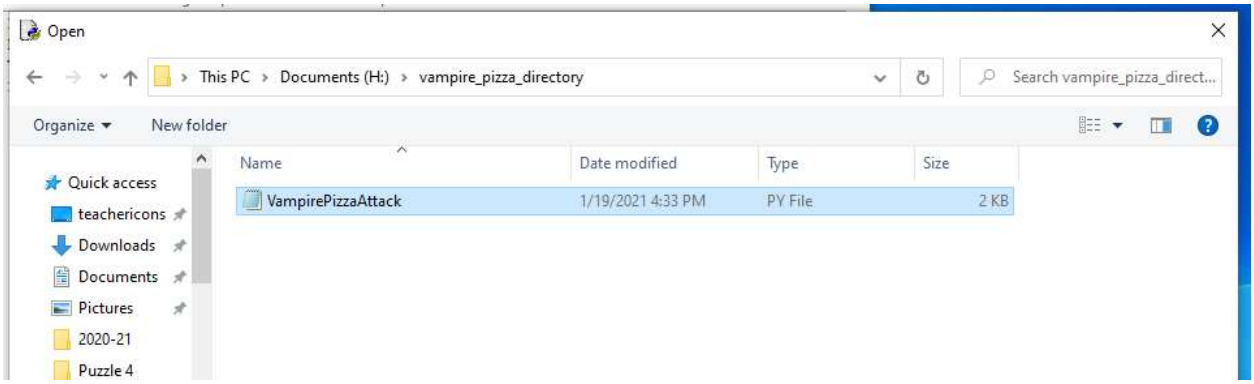
2. Your IDLE window should look something like this once it has launched.:




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 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 86, as shown in the screenshot below.

```
#-------------------------------------------------------------
#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 = REG_SPEED
        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))
```

7. Press ENTER.

8. Type the code that you see on Line 87 of the screenshot below. Ensure your indentation matches the indentation shown in the screenshot.

```
#------------------------------------------------------------
#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 = REG_SPEED
        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))
        self.health = 100
```

The code on Line 87 creates another instance attribute called self.health and sets its value to be 100.

9. Click at the end of Line 92, as shown in the screenshot below.

```
        self.health = 150

    #This function moves the enemies from right to left and destroys them after they've left the screen
    def update(self, game_window, counters):
        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))
```

10. Press ENTER.

11. Type the code that you see on Lines 93 – 96 of the screenshot below. Ensure that your indentation matches what is shown in the screenshot. You will also have to revise the game_window.blit… code that is already in your file to appear as shown in the screenshot.

```
    #This function moves the enemies from right to left and destroys them after they've left the screen
    def update(self, game_window, counters):
        game_window.blit(BACKGROUND, (self.rect.x, self.rect.y), self.rect)
        self.rect.x -= self.speed
        if self.health <= 0 or self.rect.x <= 100:
            self.kill()
        else:
            game_window.blit(self.image, (self.rect.x, self.rect.y))
```

The code on Line 93 creates an if function that checks the value of the object's health and rect.x position. If the object's health is less than or equal to 0 or the rect.x position is less than or equal to 100, the code on Line 94 runs. Line 94 will kill that particular class object.

If the health or the rect.x position are not less than or equal to 0 or less than or equal to 100, respectively, the code on Line 96 will run. This code, as we have previously discussed, will update the game window by blitting the vampire image at its new position.

12. Press ENTER twice.

13. Type the code that you see on Lines 98 – 102 of the screenshot below. Ensure your indentation matches what is shown in the screenshot.

```
    #This function moves the enemies from right to left and destroys them after they've left the screen
    def update(self, game_window, counters):
        game_window.blit(BACKGROUND, (self.rect.x, self.rect.y), self.rect)
        self.rect.x -= self.speed
        if self.health <= 0 or self.rect.x <= 100:
            self.kill()
        else:
            game_window.blit(self.image, (self.rect.x, self.rect.y))

    def attack(self, tile):
        if tile.trap == SLOW:
            self.speed = SLOW_SPEED
        if tile.trap == DAMAGE:
            self.health -= 1

class Counters (object):
```

Line 98 creates another method within the VampireSprite class. This method is called “attack” and it takes the tile argument whenever it is called.

Line 99 contains an if function that checks to see if that class object’s tile.trap attribute is set to “SLOW”. Remember, whenever the user uses a garlic trap, the trap_type will be set to SLOW. If the trap type on that tile is equal to SLOW, then the code on Line 100 executes to change the speed of the vampire sprite object to the value of the SLOW_SPEED variable.

Lines 101 and 102 repeat the process to check if the class object’s tile.trap attribute is set to DAMAGE (the user has used the pizza cutter trap). If that is the case, than the program will subtract one from the VampireSprite object’s health.

**14. Asdfhgaksdfhgasfd**

```
#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, rect):
        super().__init__()
        self.trap = None
        self.rect = rect
```

15. Click at the end of Line 156, as shown in the screenshot below.

```
#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
```

16. Delete the text on Line 156 and add the text shown in the screenshot below instead.

```
#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.trap = None
        self.rect = rect
```

17. Press ENTER twice.

18. Type the code you see on Lines 159 – 166 of the screenshot below. Ensure your indentation matches what is shown in the screenshot.

```
#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.trap = None
        self.rect = rect

class PlayTile(BackgroundTile):
    def set_trap(self, trap, counters):
        if bool(trap) and not bool(self.trap):
            counters.pizza_bucks -= trap.cost
            self.trap = trap
            if trap == EARN:
                counters.buck_booster += 1
        return None
```

Line 159 creates another class called PlayTile. This is a subclass based off of the BackgroundTile class. Our PlayTile objects will consist of the main game screen that the player can click on to set traps.

Line 160 creates the first method in the PlayTile subclass. This method is called set_trap and it will need the trap and counters arguments entered when it is called.

Line 161 contains an if function that checks to see if the boolean value of that tile's trap setting is true, which means that the tile was selected. The second portion of the if statement returns the opposite of the "bool" value for that particular background tile's trap setting. For example, if the tile has been selected, the boolean would evaluate to True. From there, if the self.trap tile is still equal to None, the boolean statement would

instead return a result of True, as opposed to False. So, if the tile was selected and it does not currently have a trap on it, the statement would evaluate to True.

If Line 161 evaluates to True, the code on Lines 162 – 165 will run. This code will subtract the cost of the trap from the player's pizza bucks, change the value of the self.trap object attribute to be "trap" instead of None, and, if the trap setting entered by the user is equal to EARN, it will add one to the player's buck_booster setting.

This method will also return the value of None. To make your functions return a value, you need to use the return statement seen on Line 166. In this case, the function will return the value of None when it is called. You can omit the return value of a function, as we have done before, or use a bare return without assigning a return value. In both cases, the return value will be done. However, it is syntactically correct to include a return value in all methods, as we have done here.

19. Press ENTER twice.

20. Type the code that you see on Lines 168 – 170 of the screenshot below. Ensure your indentation matches that shown in the screenshot below.

```
class PlayTile(BackgroundTile):
    def set_trap(self, trap, counters):
        if bool(trap) and not bool(self.trap):
            counters.pizza_bucks -= trap.cost
            self.trap = trap
            if trap == EARN:
                counters.buck_booster =+ 1
        return None

    def draw_trap(self, game_window, trap_applicator):
        if bool(self.trap):
            game_window.blit(self.trap.trap_img, (self.rect.x, self.rect.y))
```

Line 168 creates a new class method called draw_trap for the PlayTile class. This method will take the game_window and trap_applicator arguments when it is called.

Line 169 checks to see if the self.trap attribute for that particular object is set to anything. Remember, boolean values can either be true or false. If the self.trap attribute has anything assigned to it other than None, than this line will return the value of True. Remember in our previous method, the set_trap method, we changed the self.trap attribute to say "trap" if the player has selected a tile and that tile does not already have a trap applied to it.

If the self.trap attribute is is found to have a value (or found to be True), Line 170 runs to blit the trap_img picture at the location of the PlayTile's rect.x and rect.y coordinates.

21. Press ENTER twice.

22. Type the code that you see on Lines 172 - 181 of the screenshot below. Ensure your indentation matches what is shown in the screenshot.

```
class ButtonTile(BackgroundTile):
    def set_trap (self, trap, counters):
        if counters.pizza_bucks >= self.trap.cost:
            return self.trap
        return trap

    def draw_trap (self, game_window, trap_applicator):
        if bool(trap_applicator.selected):
            if trap_applicator.selected == self.trap:
                game_window.blit(self.trap.trap_img, (self.rect.x, self.rect.y))
```

Line 172 creates another subclass of the BackgroundTile. This subclass is called ButtonTile. For your reference, a Button Tile will be used to create buttons on the game window that allow a player to select a trap to apply.

Line 173 creates the first method of this subclass. This method is called set_trap. It will require the trap and the counters argument when it is called.

Line 174 starts an “if” statement that will check whether the pizza.bucks founter is greater than or equal to the cost of the trap. If this test returns the value of “True”, than this function will return the result of the self.trap class object attribute. Otherwise, it will return the value “trap”.

Line 177 is blank.

Line 178 creates another method in this subclass. This method is called draw_trap and will require the game_window and trap_applicator arguments when it is called.

Line 179 checks to see if the self.selected attribute for that tile is set to True. If it is, it will then check to see if its self.selected value is equal to self.trap (Line 180).

If both these conditions are true, than the draw.rect function on Line 181 will run to draw a rectangle in the game window at the location of that button tile. The RGB values of 238, 190, 47 corresponds to a yellow/gold color. The rectangle will be the size of the WIDTH and HEIGHT variables. The border of the rectangle will be 5 pixels.

23. Press ENTER twice.

24. Type the code you see on Lines 183 – 190 of the screenshot below. Ensure your indentation matches that of the screenshot.

```
class InactiveTile(BackgroundTile):
    #Do nothing if clicked
    def set_trap(self, trap, counters):
        return None

    #Do not display anything
    def draw_trap(self, game_window, trap_applicator):
        pass
```

Line 183 creates another subclass called InactiveTile. This subclass is based off of the BackgroundTile class. For your reference, inactive tiles are tiles around the outside of the grid that do not function as either buttons the player can use to select traps or as tiles on the actual game grid that the pizza sprites will move across. They are the ouside tiles that do not do anything.

Line 184 contains a comment.

Line 185 creates the set_trap method for the InactiveTile subclass. This method will return None. This means that the tile will not be able to have a trap set on it.

Line 186 is blank and Line 187 contains another comment.

Line 188 creates another method in this subclass. This method is called draw_trap. This method will use the “pass” keyword to tell the program to do nothing.

25. Select Lines 220 – 223, as shown in the screenshot below.

```
#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))
```

26. Hit your BACKSPACE key twice to delete the four lines that are selected and move your insertion point up to the end of the previous line.

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

27. Press ENTER. Your insertion point should be indented.

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

```

28. Type the code you see on Lines 220 - 231 of the screenshot below. Ensure your indentation matches what is shown in the screenshot.

```
#Populate the background grid
for row in range(6):
    row_of_tiles = []
    tile_grid.append(row_of_tiles)
    for column in range(11):
        tile_rect = Rect(WIDTH * column, HEIGHT * row, WIDTH, HEIGHT)
        if column <= 1:
            new_tile = InactiveTile(tile_rect)
        else:
            if row == 5:
                if 2 <= column <= 4:
                    new_tile = ButtonTile(tile_rect)
                    new_tile.trap = [SLOW, DAMAGE, EARN][column - 2]
                else:
                    new_tile = InactiveTile(tile_rect)
            else:
                new_tile = PlayTile(tile_rect)
```

The code on Line 220 creates a new variable called tile_rect. This variable is set to be the result of the Rect function. The first two numbers in the parentheses after the Rect function specify the x and y location for the top left corner of the rectangle. The second two numbers in the parentheses specify the rectangle's width and height. Since this is found in the "for" loop that will loop 11 times, and that "for" loop is nested until another loop that repeats itself 6 times, a new tile_rect location will be calculated every time a background grid tile is generated. The tiles are generated from by row from left to right and from top to bottom.

Line 221 begins an if function that checks whether the current column iteration the “for” loop is on is less than or equal to 1. In other words, the program will check to see if it is still working on populating the background grid in the first or second column. Remember, column numbers and row numbers start at 0.

If this is the case, Line 222 runs. This line will create a new variable called new_tile and set the variable’s value to be an instance of the InactiveTile class object at the location stored in the tile_rect variable.

If the column number is not less than or equal to 1, Line 222 will be skipped and the program will move down to the code on Line 223.

Line 224 checks to see if the current row iteration is row 5, meaning it is the 6$^{th}$ row from the top (or the bottom row in your grid). If this is the case, the program will then check to see if the column index number is 2, 3, or 4 (Line 225).

If the program determines that it is iterating on the row with an index of 5 and a column index of 2, 3, or 4, Line 226 will run. This will create a variable called new_tile and set its value to be a ButtonTile class object using the tile_rect location that was calculated earlier. Line 226 will set the new_tile.trap attribute to either SLOW, DAMAGE, or EARN.

To determine which tile gets assigned which trap, it takes a bit of deduction. Remember, list items are numbered by their index values. Therefore, in the first list on Line 227, the SLOW trap has an index value of 0 because it is the first item in the list. The DAMAGE trap has an index value of 1 since it is the second in the list. The EARN trap has an index value of 2 since it is the third in the list.

In the second set of brackets, you see that the current column index value is found and then the program subtracts 2 from that number.

Remember we are working in the tiles with column indexes of 2, 3, and 4 that are in row 5 of our grid. So, if you take the column index number 2 and subtract 2 from it, you will get 0. This zero number corresponds to the list item with a 0 inxex, which is the SLOW item. This means that the first new_tile.trap setting will be SLOW and will be set to the tile with a column index value of 2 and a row index value of 5.

The same process would be repeated to assign the tile in column index 3 the DAMAGE trap and column index 4 the EARN trap.

In summary, Line 227 will assign the appropriate tile traps to the appropriate tiles based on which column the tiles are in.

Lines 228 and 229 will execute if the row index is equal to 5 but the column index is NOT 2, 3, or 4. Line 229 will create more instances of the InactiveTile class objects at the tile_rect location specified.

Line 230 will execute if the column index is not less than or equal to 1 and the row is not equal to 5.

Line 231 will create a PlayTile class object at the location calculated for the tile_rect variable. Again, remember this loop runs every time a new tile is generated. So, the tile_rect position will be different each time the loop runs.

29. Press ENTER.

30. Type the code shown on Lines 232 – 234 of the screenshot below. Ensure your indentation matches what you see in the screenshot.

```
#Populate the background grid
for row in range(6):
    row_of_tiles = []
    tile_grid.append(row_of_tiles)
    for column in range(11):
        tile_rect = Rect(WIDTH * column, HEIGHT * row, WIDTH, HEIGHT)
        if column <= 1:
            new_tile = InactiveTile(tile_rect)
        else:
            if row == 5:
                if 2 <= column <= 4:
                    new_tile = ButtonTile(tile_rect)
                    new_tile.trap = [SLOW, DAMAGE, EARN][column - 2]
                else:
                    new_tile = InactiveTile(tile_rect)
        row_of_tiles.append(new_tile)
        if row == 5 and 2 <= column <= 4:
            BACKGROUND.blit(new_tile.trap.trap_img, (new_tile.rect.x, new_tile.rect.y))
```

Line 232 appends the new_tile that was created to the row_of_tiles list.

Line 233 contains an if statement that checks to see if the row index is5 and if the column index is 2, 3, or 4.

Line 234 blits the trap image that matches the trap assigned to the new_tile at the x and y location of the new_tile.

31. Press ENTER.

32. Type the code your see on Lines 235 - 237 of the screenshot below. Ensure your indentation matches what is 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):
        tile_rect = Rect(WIDTH * column, HEIGHT * row, WIDTH, HEIGHT)
        if column <= 1:
            new_tile = InactiveTile(tile_rect)
        else:
            if row == 5:
                if 2 <= column <= 4:
                    new_tile = ButtonTile(tile_rect)
                    new_tile.trap = [SLOW, DAMAGE, EARN][column - 2]
                else:
                    new_tile = InactiveTile(tile_rect)
        row_of_tiles.append(new_tile)
        if row == 5 and 2 <= column <= 4:
            BACKGROUND.blit(new_tile.trap.trap_img, (new_tile.rect.x, new_tile.rect.y))
        if column != 0 and row != 5:
            if column !=1:
                draw.rect(BACKGROUND, tile_color, (WIDTH * column, HEIGHT * row, WIDTH, HEIGHT), 1)
```

Lines 235 - 237 check to see if the column index DOES NOT equal 0 or 1 and if the row index DOES NOT equal 5. If both of these conditions are true, Line 235 will run the draw.rect function to draw a rectangle on the background image using the tile_color specified (currently, it is set to white), the width and the height specified, and the location of the x and y coordinates for the top left corner of the rectangle (the first two numbers in the parentheses). The last number in the parentheses, 1, will be the border width in pixels.

33. Select Lines 288 – 293, as shown in the screenshot below.

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

    for vampire in all_vampires:
        tile_row = tile_grid[vampire.rect.y //100]
        vampire_left_side_x = vampire.rect.x // 100
        vampire_right_side_x = (vampire.rect.x + vampire.rect.width) // 100
        if -1 < vampire_left_side_x < 10:
            left_tile = tile_row[vampire_left_side_x]
        else:
            left_tile = None
        if -1 < vampire_right_side_x < 10:
            right_tile_wall = tile_row[vampire_right_side_x]
        else:
            right_tile = None
        if bool(left_tile) and left_tile.effect:
            vampire_speed = SLOW_SPEED
        if bool(right_tile) and right_tile.x != left_tile.x and right_tile.effect:
            vampire.speed = SLOW_SPEED
        if vampire.rect.x <= 0:
            vampire.kill()
```

34. Click your BACKSPACE key twice to delete the selected lines and move your insertion point to the end of the previous line.

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

    for vampire in all_vampires:
        tile_row = tile_grid[vampire.rect.y //100]
        vampire_left_side_x = vampire.rect.x // 100
        vampire_right_side_x = (vampire.rect.x + vampire.rect.width) // 100
        if -1 < vampire_left_side_x < 10:
            left_tile = tile_row[vampire_left_side_x]
        else:
            left_tile = None
        if -1 < vampire_right_side_x < 10:
            right_tile_wall = tile_row[vampire_right_side_x]
        else:
            right_tile_wall = None
```

35. Press ENTER then BACKSPACE again. Your insertion point should be indented one indentation.

36. Type the code you see on Lines 288 – 292 of the screenshot below. Ensure your indentation matches what is shown in the screenshot.

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

    for vampire in all_vampires:
        tile_row = tile_grid[vampire.rect.y //100]
        vampire_left_side_x = vampire.rect.x // 100
        vampire_right_side_x = (vampire.rect.x + vampire.rect.width) // 100
        if -1 < vampire_left_side_x < 10:
            left_tile = tile_row[vampire_left_side_x]
        else:
            left_tile = None
        if -1 < vampire_right_side_x < 10:
            right_tile_wall = tile_row[vampire_right_side_x]
        else:
            right_tile_wall = None
        if bool(left_tile):
            vampire.attack(left_tile)
        if bool(right_tile_wall):
            if right_tile_wall != left_tile:
                vampire.attack(right_tile_wall)
```

Previously in our game loop, every time a VampireSprite collided with a trap, it was set to slow down. But now we have three different types of traps that all do different things. To account for this, we need to tell the program to apply the effect of each tile.

Line 288 will check to see if there is a value for the left_tile attribute. Remember the left_tile variable specifies that, if there is a tile on the grid, what column that tile is in. The value of the left_tile variable will be None if there is no vampire sprite on the grid. Likewise, the value of the right_tile grid will check to see if the right side of the sprite is on the grid and, if so, where it is at. So, if the boolean function on Line 286 will return true if the left_tile variable has a value, meaning that the left side of the vampire sprite is on the grid.

If Line 289 returns the value of True, the attack function that you created in your vampire sprite class will run, which applies the appropriate trap to the appropriate tile.

If Line 288 returns the value of False, Line 289 will be skipped and the program will move down to Line 290. Line 290 checks to see if the right_tile variable has a value, meaning that the right side of the tile is on the grid. If this is true, then the program will double-check to make sure that the right_tile and the left_tile variable do not equal each other. If they don’t, then Line 292 will run to run the attack method from your Vampire Sprite class in the right_tile and apply the appropriate trap.

37. Click at the end of Line 297, as shown in the screenshot below.

```
            if bool(right_tile_wall):
                if right_tile_wall != left_tile:
                    vampire.attack(right_tile_wall)

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

38. Press ENTER twice.

39. Type the code you see on Lines 299 – 301 of the screenshot below. Ensure your indentation matches what is shown.

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

        for tile_row in tile_grid:
            for tile in tile_row:
                tile.draw_trap(GAME_WINDOW, trap_applicator)
```

The code above will update the traps by redrawing them on all of the tiles using the draw_trap function you defined earlier.

**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

#Set up counters
STARTING_BUCKS = 15
BUCK_RATE = 120
STARTING_BUCK_BOOSTER = 1

#Define speeds
REG_SPEED = 2
SLOW_SPEED = 1

#---------------------------------------------------------------
#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))

#tile trap images
garlic_img = image.load('garlic.png')
garlic_surf = Surface.convert(garlic_img)
GARLIC = transform.scale(garlic_surf, (WIDTH, HEIGHT))
GARLIC.set_alpha(127)
cutter_img = image.load('pizzacutter.png')
cutter_surf = Surface.convert(cutter_img)
CUTTER = transform.scale(cutter_surf, (WIDTH, HEIGHT))
CUTTER.set_alpha(127)
pepperoni_img = image.load('pepperoni.png')
pepperoni_surf = Surface.convert(pepperoni_img)
PEPPERONI = transform.scale(pepperoni_surf, (WIDTH, HEIGHT))
PEPPERONI.set_alpha(127)

#-------------------------------------------------------------
#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 = REG_SPEED
        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))
        self.health = 100

    #This function moves the enemies from right to left and destroys them after they've left the screen
    def update(self, game_window, counters):
        game_window.blit(BACKGROUND, (self.rect.x, self.rect.y), self.rect)
        self.rect.x -= self.speed
        if self.health <= 0 or self.rect.x <= 100:
            self.kill()
        else:
            game_window.blit(self.image, (self.rect.x, self.rect.y))
```

```
    def attack(self, tile):
        if tile.trap == SLOW:
            self.speed = SLOW_SPEED
        if tile.trap == DAMAGE:
            self.health -= 1

class Counters (object):

    def __init__(self, pizza_bucks, buck_rate, buck_booster):
        self.loop_count = 0
        self.display_font = pygame.font.SysFont("Arial", 25)
        self.pizza_bucks = pizza_bucks
        self.buck_rate = buck_rate
        self.buck_booster = buck_booster
        self.bucks_rect = None

    def increment_bucks(self):
        if self.loop_count % self.buck_rate == 0:
            self.pizza_bucks += self.buck_booster

    def draw_bucks(self, game_window):
        if bool(self.bucks_rect):
            game_window.blit(BACKGROUND, (self.bucks_rect.x, self.bucks_rect.y), self.bucks_rect)
        bucks_surf = self.display_font.render(str(self.pizza_bucks), True, WHITE)
        self.bucks_rect= bucks_surf.get_rect()
        self.bucks_rect.x = WINDOW_WIDTH - 50
        self.bucks_rect.y = WINDOW_HEIGHT - 50
        game_window.blit(bucks_surf, self.bucks_rect)

    def update (self, game_window):
        self.loop_count += 1
        self.increment_bucks()
        self.draw_bucks(game_window)

#Set up the different kinds of traps
class Trap(object):

    def __init__(self, trap_kind, cost, trap_img):
        self.trap_kind = trap_kind
        self.cost = cost
        self.trap_img = trap_img

class TrapApplicator(object):

    def __init__(self):
        self.selected = None

    def select_trap (self, trap):
        if trap.cost <= counters.pizza_bucks:
            self.selected = trap

    def select_tile(self, tile, counters):
        self.selected = tile.set_trap(self.selected, counters)
```

```
 #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, rect):
        super().__init__()
        self.trap = None
        self.rect = rect

class PlayTile(BackgroundTile):
    def set_trap(self, trap, counters):
        if bool(trap) and not bool(self.trap):
            counters.pizza_bucks -= trap.cost
            self.trap = trap
            if trap == EARN:
                counters.buck_booster += 1
        return None

    def draw_trap(self, game_window, trap_applicator):
        if bool(self.trap):
            game_window.blit(self.trap.trap_img, (self.rect.x, self.rect.y))

class ButtonTile(BackgroundTile):
    def set_trap (self, trap, counters):
        if counters.pizza_bucks >= self.trap.cost:
            return self.trap
        return trap

    def draw_trap (self, game_window, trap_applicator):
        if bool(trap_applicator.selected):
            if trap_applicator.selected == self.trap:
               game_window.blit(self.trap.trap_img, (self.rect.x, self.rect.y))

class InactiveTile(BackgroundTile):
    #Do nothing if clicked
    def set_trap(self, trap, counters):
        return None

    #Do not display anything
    def draw_trap(self, game_window, trap_applicator):
        pass

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

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

counters=Counters(STARTING_BUCKS, BUCK_RATE, STARTING_BUCK_BOOSTER)

SLOW = Trap('SLOW', 5, GARLIC)
DAMAGE = Trap('DAMAGE', 3, CUTTER)
EARN = Trap('EARN', 7, PEPPERONI)

trap_applicator = TrapApplicator()

#-------------------------------------------------------------
#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):
        tile_rect = Rect(WIDTH * column, HEIGHT * row, WIDTH, HEIGHT)
        if column <= 1:
            new_tile = InactiveTile(tile_rect)
        else:
            if row == 5:
                if 2 <= column <= 4:
                    new_tile = ButtonTile(tile_rect)
                    new_tile.trap = [SLOW, DAMAGE, EARN][column - 2]
                else:
                    new_tile = InactiveTile(tile_rect)
            else:
                new_tile = PlayTile(tile_rect)
        row_of_tiles.append(new_tile)
        if row == 5 and 2 <= column <= 4:
            BACKGROUND.blit(new_tile.trap.trap_img, (new_tile.rect.x, new_tile.rect.y))
        if column != 0 and row != 5:
            if column !=1:
                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()
            trap_applicator.select_tile(tile_grid[y//100][x//100], counters)

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

```
#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:
            if bool(tile.trap):
                GAME_WINDOW.blit(BACKGROUND, (tile.rect.x, tile.rect.y), tile.rect)

    for vampire in all_vampires:
        tile_row = tile_grid[vampire.rect.y //100]
        vampire_left_side_x = vampire.rect.x // 100
        vampire_right_side_x = (vampire.rect.x + vampire.rect.width) // 100
        if -1 < vampire_left_side_x < 10:
            left_tile = tile_row[vampire_left_side_x]
        else:
            left_tile = None
        if -1 < vampire_right_side_x < 10:
            right_tile_wall = tile_row[vampire_right_side_x]
        else:
            right_tile_wall = None
        if bool(left_tile):
            vampire.attack(left_tile)
        if bool(right_tile_wall):
            if right_tile_wall != left_tile:
                vampire.attack(right_tile_wall)

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

    for tile_row in tile_grid:
        for tile in tile_row:
            tile.draw_trap(GAME_WINDOW, trap_applicator)

    #Update counters
    counters.update(GAME_WINDOW)

    display.update()

    #set the framerate
    clock.tick(FRAMERATE)

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

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