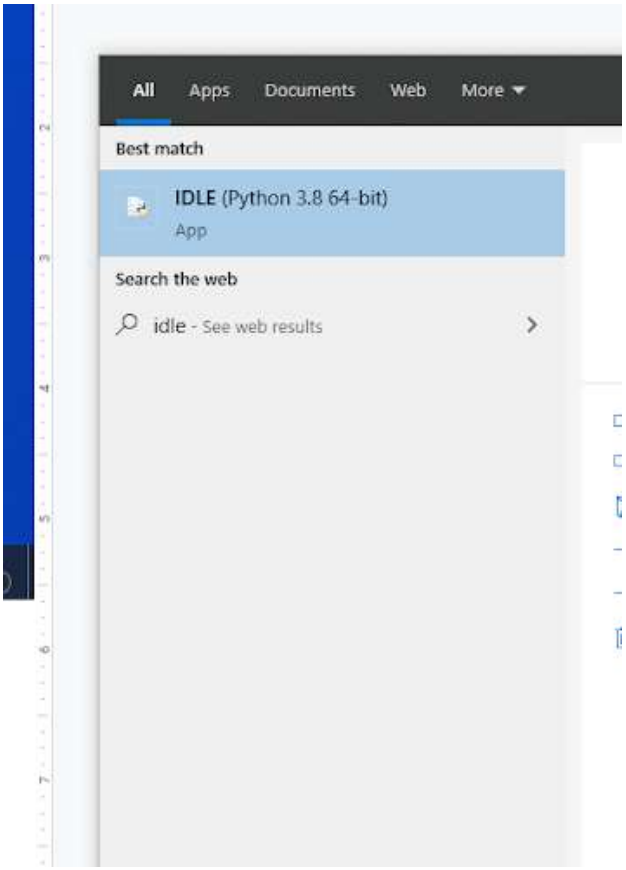
Chapter 10 Practice Directions

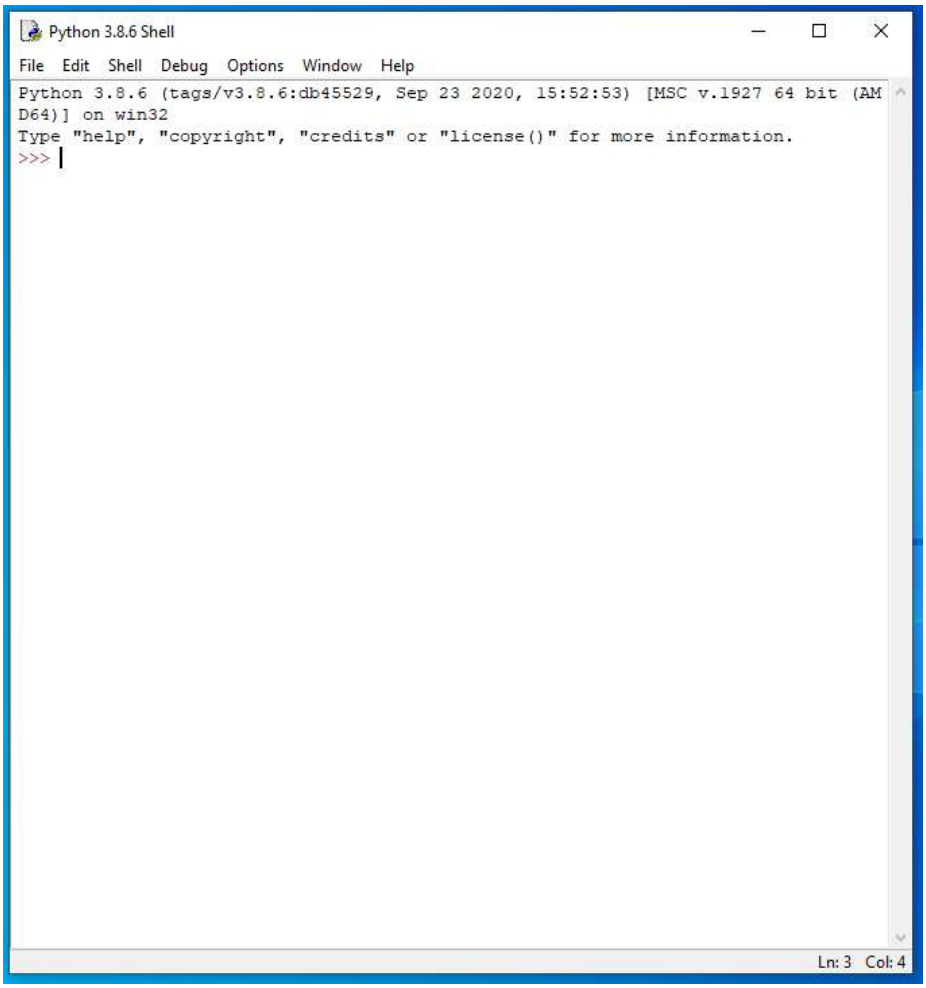
1. . Using your Windows button menu, find and launch your IDLE program.



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.

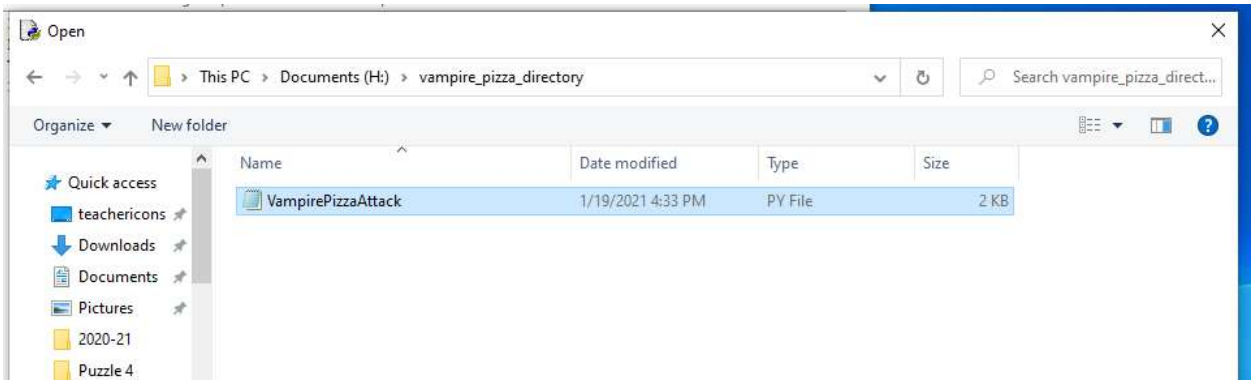
2. Your IDLE window should look something like this once it has launched.:



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 29, as shown in the screenshot below.

```
#Set up rates
SPAWNRATE = 360
FRAMERATE = 60
```

7. Press ENTER twice.

8. Type the code that you see on Lines 31 – 34 of the screenshot below.

```
#Set up rates
SPAWNRATE = 360
FRAMERATE = 60

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

The code on Line 31 contains a comment.

The code on Lines 32 – 34 creates three more constant variables and sets their values. These variables contain starting values for the game. The player will start with 15 pizza bucks.

The BUCK_RATE variable will store the number of game loops that should run for the player to earn pizza bucks. We set this value to 120, for now.

Finally, the STARTING_BUCK_BOOSTER variable will store the number of pizza bucks the player should earn each time the set amount of game loops runs. Since we have set our BUCK_RATE variable to 120 and the STARTING_BUCK_BOOSTER variable to 1, the player will receive 1 pizza buck every time the game loops 120 times.

9. Click in the parentheses on Line 75, as shown in the screenshot below.

```
#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))

    #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))
```

10. Update the code on Line 75 to match the screenshot below.

```
#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))
```

This modification will add the “counters” argument to the update function. Whenever the update function is called, the programmer will now have to specify which counters to update.

11. Click at the end of Line 78, as shown in the screenshot below.

```
#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))
```

12. Press ENTER twice.

13. Click your backspace button until you are all the way at the left margin.

14. Type the code that you see on Lines 80 – 88 of the screenshot below.

```
            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))

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
```

Line 80 creates another class called Counters. The class is based off of the object base class.

Line 81 is blank.

Line 82 contains the __init__ method. As we have learned before, this method is always the first method used when creating a new class. This method sets up the rules for all objects created under this class. The __init__ method indicates that every object created under this class will need to have the attributes of pizza_bucks, buck_rate, and buck_booster. The values of these attributes for each instance of the Counters class can be different if the programmer chooses. The programmer will enter these values manually when the object is created.

Lines 83 – 88 contain attribute specifications for each Counters object created. For example, Line 83 creates a loop_count attribute for that particular object and sets the loop_count for that particular object to 0. Line 84 specifies the font to display for this particular counter object. Lines 85, 86, and 87 set the pizza_bucks, the buck_rate, and the buck_booster attributes for that counter object to be whatever the programmer tells it to be when the object is created. For example, if, when the programmer is creating the counter object, it is specified that the programmer wants the pizza_bucks to be 10, than the object created will assign itself the self.pizza_bucks value of 10.

Line 88 contains a .rect specification. We do not want this object to have a built in .rect right now, so we specify the .rect value as None. However, we may want to create a rect, move the rect, or specify the rect's position later, so it's important to provide that flexibility for our object now.

15. Press ENTER twice.

16. Type the code that you see on Lines 90 -92 of the screenshot below. Ensure that your indentation matches what is shown in the screenshot below.

```
        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

#Create a class of sprites. Each tile has an invisible interactive field attached to it which is a sprite in this class.
```

Line 90 creates the increment_bucks method. This method will take the self argument whenever it is called.

Line 91 begins an “if” loop. The increment_bucks method will check to see if the self.loop_count can be divided into the self.buck_rate without any leftover remainder. This means that the remainder is 0. If it is, Line 92 will run. If not, Line 92 will not run.

For example, if the self.loop_count value is 240 and the self.buck_rate is 120, then the loop_count value (240) divided by the buck_rate value (120) is 2 with no remainder leftover. This will evaluate to “True” for the if statement and Line 92 would run. In other words, for every 120 frames, the player will be granted whatever Line 92 specifies.

Line 92 adds the value of the self.pizza_bucks attribute for that particular class object to the value of the self.buck_booster for that particular class. This line will run every time the game loops a specified number of times.

17. Press ENTER twice.

18. Type the code that you see on Lines 94 – 101 of the screenshot below. Ensure your indentation matches the indentation shown in the screenshot.

```
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)
```

Line 94 creates a new method called draw_bucks. This method will take the self and the game_window arguments whenever it is called.

Line 95 starts another “if” function to evaluate if the self.bucks_rect variable is set to True. If it is, the rest of the code on Lines 96 – 101 will run. If it isn’t, than the code will not run.

As statement before, Line 96 will not run unless the self.bucks_rect variable is True.

If it is, Line 96 will blit the background image to the game window at the x and y location specified (the self.bucks_rect.x and self.bucks_rect.y). Since we want to display a new self.bucks number over the old self.bucks number, we are going to display the portion of the background image that can be found at the location of the self.bucks_rect element. We do not want it to redisplay the entire background image at the location of self.buck_rect.x and self.bucks_rect.y – we only want a portion of that background image displayed. The last number in the code will specify that we only want to redisplay that small section of background (in the self.bucks_rect spot) at the specified location. This code will allow us to cover the previous bucks value with the background image, essentially hiding it from view.

Remember, the .blit syntax is as follows: game_window.blit(SURFACE IT WILL BLIT, (LOCATION X, LOCATION Y OF BLIT), WHAT PART OF IMAGE TO DISPLAY).

Line 97 creates a variable called bucks_surf and sets its value to be the result of the display_font.render function. Essentially, the bucks_surf variable will be used to display text. This code will render the specified text using the font pizza-font.ttf. The rest of the code on Line 97 converts the self.pizza_bucks variable to a string value so that it can be displayed as text. It also sets the color of the text (the value of the WHITE variable) and the anti-aliasing specification for the text (True). When you set the anti-aliasing specification to True, the letters in the text will have smooth edges. If the anti-aliasing is set to False, the letters will appear to have jagged edges.

Line 98 updates the self.bucks_rect variable's value. It uses the bucks_surf.get_rect() function to create a rectangle for the text object in the bucks_surf variable.

Lines 99 and 100 set the values of the self.bucks_rect.x and self.bucks_rect.y variables to be a math formula, subtracting 50 from the WINDOW_WIDTH or WINDOW_HEIGHT variable values. This will place the self.bucks_rect object in the middle of the tile on the bottom-right corner.

The last line, Line 101, blits the bucks_surf image to the game_window at the location of the self.bucks_rect coordinates. This will display the new pizza bucks total in the game window.

19. Press ENTER twice.

20. Type the code that you see on Lines 103 – 106 of the screenshot below. Ensure your indentation matches the indentation shown in the screenshot.

```
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)
```

Line 103 creates another method called update. This method takes the game_window argument when it is called.

Line 104 will increase the value of the self.loop_count variable by 1.

Line 105 will run the increment_bucks() method on that specific object.

Line 106 will run the draw_bucks method on that specific class object. The draw_bucks method requires the programmer to tell what window the method will run in as an argument. Since you want the method to run in the game_window, you enter that in the parentheses.

21. Click at the end of Line 118, as shown in the screenshot below.

```
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()
```

22. Press ENTER twice.

23. Type the code that you see on Line 120.

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

counters=Counters(STARTING_BUCKS, BUCK_RATE, STARTING_BUCK_BOOSTER)
```

This code creates a new variable called counters. Its value is set to be an instance of the Counters object class that you created earlier. This class will use the value of the STARTING_BUCKS, BUCK_RATE, AND STARTING_BUCK_BOOSTER variables as its arguments for setting up its instance/object attributes.

24. Click at the end of Line 199, shown in the screenshot below.

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

25. Modify the code as shown below.

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

    This modification will update each individual vampire sprite's counters. This is not relevant now, but will be next chapter.

26. Click at the end of the line (after the closing parentheses).

27. Press ENTER twice.

28. Type the code that you see on Lines 201 – 202.

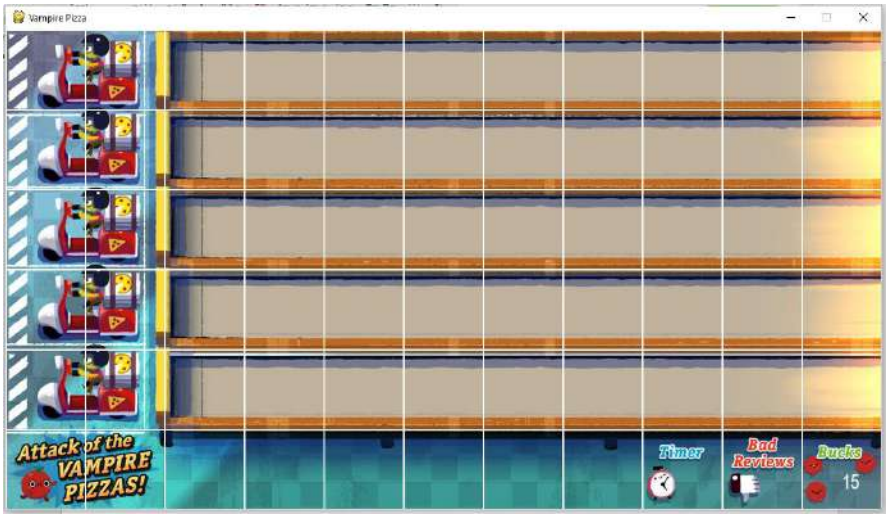
```
#-------------------------------------------------------------
    #Update display.
    for vampire in all_vampires:
        vampire.update(GAME_WINDOW, counters)

    #Update counters
    counters.update(GAME_WINDOW)

    display.update()

    #set the framerate
    clock.tick(FRAMERATE)
```

    Line 201 contains a comment.

    Line 202 will update the counters variable in the GAME_WINDOW. This update will run every frame.

29. File > Save to save your file.

30. Go to Run > Run Module to run your file. You should notice the game window load with a number that counts up in the bottom right corner. The number should increment by one every few seconds.


Vampire Pizza
Attack of the VAMPIRE PIZZAS!
Timer
Bad Reviews
Bucks
15

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))

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

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

    #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))

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)

 #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()

counters=Counters(STARTING_BUCKS, BUCK_RATE, STARTING_BUCK_BOOSTER)

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

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

    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()

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

    #Update counters
    counters.update(GAME_WINDOW)

    display.update()

    #set the framerate
    clock.tick(FRAMERATE)

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

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