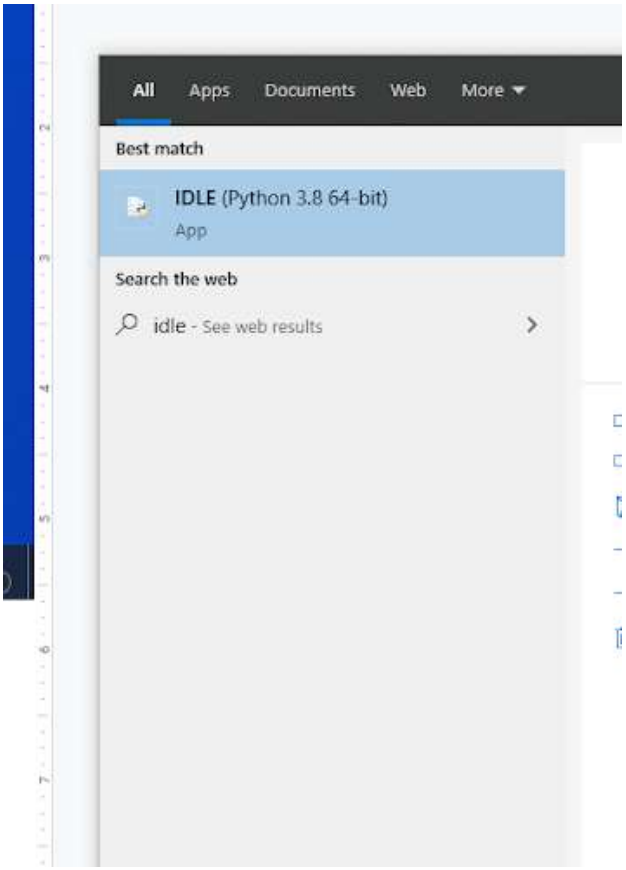
Chapter 13 Practice Directions

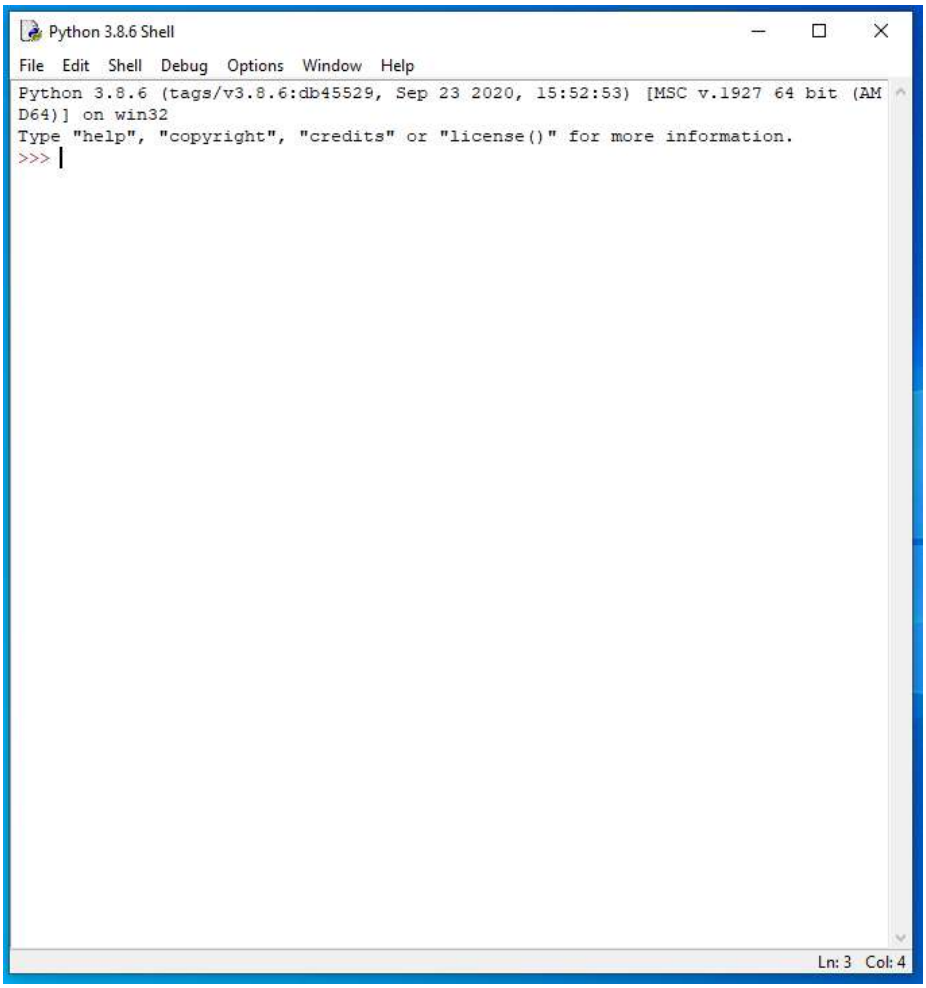
1. Using your Windows button menu, find and launch your IDLE program.



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.

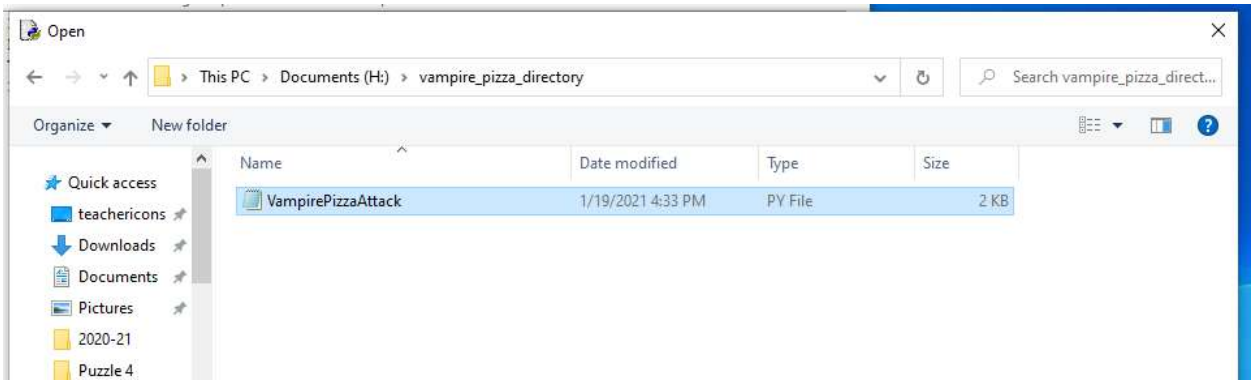
2. Your IDLE window should look something like this once it has launched.:



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 34, as shown in the screenshot below.

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

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

7. Press ENTER.

8. Type the code you see on Lines 35 – 36 of the screenshot below.

```
#Set up counters
STARTING_BUCKS = 15
BUCK_RATE = 120
STARTING_BUCK_BOOSTER = 1
MAX_BAD_REVIEWS = 3
WIN_TIME = FRAMERATE * 60 * 3
```

Line 35 creates a constant variable called MAX_BAD_REVIEWS. The variable's value is set to 3.

Line 36 creates another constant variable called WIN_TIME. This variable will represent the number of frames the game runs through before the time runs out. We want to start out with our game running for 3 minutes. The variable's value is set to be the value of the FRAMERATE variable (currently set to 60), times 60, times 3. Remember the framerate is how many frames are generated per second. So, 60 frames per second times 60 is 3,600 frames. This is the number of times the game loop runs in one minute. Multiple 3,600 by 3 and you get 10,800 frames, which is the number of times the game loop runs in three minutes.

If you want to change the time on the clock later on, just change the number 3 to the number of minutes that you want the game to last.

9. Click at the end of Line 96, 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
        if self.health <= 0 or self.rect.x <= 100:
            self.kill()
```

10. Press ENTER.

11. Type the code you see on Lines 97 – 98 of the screenshot below. Ensure your indentation matches the indentation 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()
        if self.rect.x <= 100:
            counters.bad_reviews += 1
    else:
        game_window.blit(self.image, (self.rect.x, self.rect.y))
```

Line 97 creates another “if” function that will check to see if the x-coordinate of the Vampire Sprite class object is less than or equal to 100. If it is, that means that the vampire sprite has reached the end of the grid and has eaten or destroyed a pizza of yours.

If this happens, customers will leave you bad reviews. The code on Line 98 will increase the bad_reviews counter by 1.

12. Click at the end of Line 110, as shown in the screenshot below.

```
            if tile.trap == DAMAGE:
                self.health -= 1

class Counters (object):

    def __init__(self, pizza_bucks, buck_rate, buck_booster):
```

13. Modify the arguments inside the def __init__ function parentheses to match the screenshot below.

```
            if tile.trap == DAMAGE:
                self.health -= 1

class Counters (object):

    def __init__(self, pizza_bucks, buck_rate, buck_booster, timer):
```

The code that you entered above includes the timer to the setup of your game counters so that the program will measure the amount of time passing.

14. Click at the end of Line 116, as shown in the screenshot below.

```
class Counters (object):

    def __init__(self, pizza_bucks, buck_rate, buck_booster, timer):
        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
```

15. Press ENTER.

16. Type the code you see on Lines 117 – 120 of the screenshot below.

```
class Counters (object):

    def __init__(self, pizza_bucks, buck_rate, buck_booster, timer):
        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
        self.timer = timer
        self.timer_rect = None
        self.bad_reviews = 0
        self.bad_reviews_rect = None

    def increment_bucks(self):
```

Lines 117 – 120 all set up class object attributes for the Counters class: self.timer, self.time_rect, self.bad_reviews, and self.bad_reviews_rect. Whenever a new Counters class object is generated, it will generate with these attributes.

17. Click at the end of Line 133, as shown in the screenshot below:

```
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)
```

18. Press ENTER twice.

19. Type the code you see on Lines 135 – 142 of the screenshot below. Ensure your indentation matches the indentation shown in the screenshot.

```
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 draw_bad_reviews(self, game_window):
    if bool(self.bad_reviews_rect):
        game_window.blit(BACKGROUND, (self.bad_reviews_rect.x, self.bad_reviews_rect.y), self.bad_reviews_rect)
    bad_reviews_surf = self.display_font.render(str(self.bad_reviews), True, WHITE)
    self.bad_reviews_rect = bad_reviews_surf.get_rect()
    self.bad_reviews_rect.x = WINDOW_WIDTH - 150
    self.bad_reviews_rect.y = WINDOW_HEIGHT - 50
    game_window.blit(bad_reviews_surf, self.bad_reviews_rect)
```

Line 135 creates another class method for the Counters class. This method is called draw_bad_reviews and it takes the game_window argument when it is called.

Line 136 checks to see if the bad_reviews_rect setting on the current class object is set to True, meaning that it has a value. Remember that when you set up the bad_reviews_rect attribute in the __init__ method, it was set to None. Whenever the bad_reviews_rect setting is set to True, it means that there is a value for the bad_reviews counter. If this is true, Line 137 will execute.

Line 137 erases the old value in the bad_reviews_rect by blitting the backtround image onto the game window at the location of the bad_reviews_rect.x and bad_reviews_rect.y coordinates. The final item in the parentheses indicates how large the blitted background square is. In this case, this item will encompass the size specified in the self_bad_reviews.rect variable. The goal is to cover up any number that was already in that position with the background image.

Line 138 creates a bad_reviews_surf variable and sets its value to be equal to the display_font.render method to set the font display settings for the bad reviews text. This method will use the integer inside the bad_reviews setting of the class object and convert it to a string so it can be displayed as text. Anti-aliasing is turned on (or set to

True) so that the letters appear smooth, and the text is set to the equal to the color settings stored in the WHITE variable.

Line 139 creates a variable called self.bad_reviews_rect and sets its value to be the result of the .get_rect() function. This function creates a rectangle the size of the bad_reviews_surf variable.

Lines 140 and 141 calculate the display location of the bad_reviews_rect x and y coordinates. By default, whenever rectangles are created using .get_rect(), their coordinates are automatically set to 0, 0. The calculations you perform in these lines will set the bad reviews rectangle to appear in the second-to-last column and the bottom row of the grid.

Now that you have erased your previous number from the screen and set the display font, color, and location, you have to blit your new number to the screen. Line 142 will blit the value of the bad_reviews_surf (the font) at the location specified (self.bad_reviews_rect).

20. Press ENTER twice.

21. Type the code you see on Lines 144 – 151 of the screenshot below. Ensure your indentation matches what is shown in the screenshot.

```
def draw_bad_reviews(self, game_window):
    if bool(self.bad_reviews_rect):
        game_window.blit(BACKGROUND, (self.bad_reviews_rect.x, self.bad_reviews_rect.y), self.bad_reviews_rect)
    bad_reviews_surf = self.display_font.render(str(self.bad_reviews), True, WHITE)
    self.bad_reviews_rect = bad_reviews_surf.get_rect()
    self.bad_reviews_rect.x = WINDOW_WIDTH - 150
    self.bad_reviews_rect.y = WINDOW_HEIGHT - 50
    game_window.blit(bad_reviews_surf, self.bad_reviews_rect)

def draw_time(self, game_window):
    if bool(self.timer_rect):
        game_window.blit(BACKGROUND, (self.timer_rect.x, self.timer_rect.y), self.timer_rect)
    timer_surf = self.display_font.render(str(int((WIN_TIME - self.loop_count)/ FRAMERATE)), True, WHITE)
    self.timer_rect = timer_surf.get_rect()
    self.timer_rect.x = WINDOW_WIDTH - 250
    self.timer_rect.y = WINDOW_HEIGHT - 50
    game_window.blit(timer_surf, self.timer_rect)
```

Line 144 creates another class method in the Counters class called draw_time. This method will run through similar steps as the previous method in order to update the time on the screen. The time will be displayed in the third-to-last column and the bottom row of the grid.

It is important to note that the math function found on Line 147 (in the timer_surf function) calculates the time left in the game, in seconds. Remember that the loop_count setting of the object counts how many time the game loop has run. By taking the WIN_TIME (number of frames in the game) minus the number of loops the game has made (or how many frames have already elapsed), you can get the number of frames

remaining in the game. From there, dividing those frames by the FRAMERATE of 60 will give you the number of seconds left in the game.

22. Click at the end of Line 156, as shown below.

```
            self.timer_rect.y = WINDOW_HEIGHT - 50
            game_window.blit(timer_surf, self.timer_rect)

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

23. Press ENTER.

24. Type the code you see on Lines 157 – 158 of the screenshot below. Ensure your indentation matches what is shown in the screenshot.

```
        self.timer_rect.y = WINDOW_HEIGHT - 50
        game_window.blit(timer_surf, self.timer_rect)

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

The code on Lines 157 and 158 call the draw_bad_reviews and the draw_time functions inside the update method. These functions both require the programmer to specify what game_window they want the functions to run in, and so we tell the program that we want our functions to run in the game window called game_window. If our game had more than one window going, it would be more important to label our game_windows with better names.

25. Click at the end of Line 226, as shown in the screenshot below.

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

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

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

26. Modify the code in the parentheses to match the screenshot below.

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

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

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

This addition allows the WIN_TIME variable to be used whenever an object instance of the Counters class is created. Adding the WIN_TIME variable here is important because the draw_time method using the WIN_TIME variable in a calculation that it performs. If you do not tell import the WIN_TIME variable into the object instance, than the draw_time method would not be able to perform a calculation and you would receive an error.

Class methods cannot reference variable information located outside of the class. Since the WIN_TIME variable was defined at the beginning of the code and not within the Counters class, it needs to be imported into the counter class objects to be able to be used.

27. Click at the end of Line 274, as shown in the screenshot below.

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

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

#Game loop
game_running = True
```

28. Press ENTER.

29. Type the code you see on Line 275 of the screenshot below.

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

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

#Game loop
game_running = True
exited = True
while game_running:
```

This code creates another variable called exited and sets its value to True. This variable represents True if either the win or lose conditions have been met. If the player opts to quit the game, we will change this variable to False since neither the win or lose conditions have been met at that time. This will make more sense as we get into the end-of-game loop that we will create.

30. Click at the end of Line 285, as shown in the screenshot below.

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

31. Press ENTER.

32. Type the code you see on Line 286 of the screenshot below. Ensure your indentation matches what is shown in the screenshot.

```
#------------------------------------------------------------
#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
            exited = False
```

The code on Line 286 will change the value of the exited variable to False, meaning that neither the win or lose conditions have been met and the player has quit the game.

33. Click at the end of Line 322, as shown in the screenshot below.

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

34. Press ENTER twice.

35. Type the code you see on Lines 324 – 329 of the screenshot below. Ensure your indentation matches what is shown in the screenshot.

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

#-------------------------------------------------------------
#Set win/lose conditions
    if counters.bad_reviews >= MAX_BAD_REVIEWS:
        game_running = False
    if counters.loop_count > WIN_TIME:
        game_running = False
```

Line 324 contains a comment separator to separate blocks of code.

Line 325 contains a comment.

Line 326 tests to see if the bad_reviews counter value is greater than or equal to the value of the MAX_BAD_REVIEWS variable (in this case, 3).

If Line 326 is found to be true, Line 327 runs to set the game_running variable to False, meaning the game is over. Notice that the exited variable is NOT changed, meaning that it is still set to True. This is because a win or a loss condition has been met – the player has lost the game by exceeding the amount of bad reviews they are allowed to have.

Line 327 tests to see if the loop_count value is greater than the WIN_TIME variable. If this is the case, then the player has won the game because the player has made it all the way through the play time without receiving 3 or more bad reviews. Since the game is over, the game_running variable will again be set to False. But, again, the exited variable is NOT changed because a win or a loss condition has been met – the player has won the game by making it through the play time without 3 bad reviews.

36. Click at the end of Line 348, as shown in the screenshot below.

```
        #Update counters
        counters.update(GAME_WINDOW)

        display.update()

        #set the framerate
        clock.tick(FRAMERATE)

#Close main game loop
```

37. Press ENTER.

38. Type the code you see on Lines 349 – 361 of the screenshot below. Ensure your indentation matches what is shown in the screenshot.

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

#Close main game loop
#-------------------------------------------------------------
#End of game loop

#Set end game message
end_font = pygame.font.SysFont("Arial", 25)

if exited:
    if counters.bad_reviews >= MAX_BAD_REVIEWS:
        end_surf = end_font.render('Game Over', True, WHITE)
    else:
        end_surf = end_font.render('You Win!!', True, WHITE)
    GAME_WINDOW.blit(end_surf, (350, 200))
    display.update()
```

Line 349 contains a comment separator.

Lines 350 and 352 contain comments.

Line 353 creates a variable called end_font. The end_font variable is set to use the Arial font from our PC in size 25pt.

Line 355 creates an “if” function that checks to see if the exited variable is equal to True. Remember that this loop will only run when the game_running variable has been set to False.

If the “exited” variable is still set to True, meaning that either a win or a lose condition has been met, the code will continue down to Line 356.

Line 356 will check the value of the bad_reviews counter to see if it is greater than or equal to the value of the MAX_BAD_REVIEWS variable (currently set to 3). If this is true, than the end_surf variable is created (Line 357), which will render the text ‘Game Over’ using the Arial font. The text anti-aliasing will be set to True, meaning that the letters will appear smooth, and the color will be set to the color settings specified in the WHITE variable.

Line 358 will run if the bad_reviews counter is NOT greater than or equal to the value of the MAX_BAD_REVIEWS variable, meaning that the player has won the game. If this is the case, Line 359 will run to create the end_surf variable that renders the text ‘You

Win!''. Again, anti-aliasing is turned on and the color is set to be the RGB color values specified in the WHITE variable.

After the appropriate message has been rendered, Line 360 will display the end_surf message at the location of 350, 200.

Line 361 runs the update function to update all displays in the game.

39. Press ENTER.

40. Type the code you see on Lines 362 – 370 of the screenshot below. Ensure your indentation matches what is shown in the screenshot.

```
if exited:
    if counters.bad_reviews >= MAX_BAD_REVIEWS:
        end_surf = end_font.render('Game Over', True, WHITE)
    else:
        end_surf = end_font.render('You Win!!', True, WHITE)
    GAME_WINDOW.blit(end_surf, (350, 200))
    display.update()
#-------------------------------------------------------------
#Enable exit from end game loop
while exited:
    for event in pygame.event.get():
        if event.type == QUIT:
            exited = False
    clock.tick(FRAMERATE)

#Close end game loop
```

Line 362 contains another comment separator.

Line 363 contains a comment.

Line 364 begins a “while” loop that will run while the exited variable is equal to True.

Line 365 begins a “for” loop that will check for any events stored in the program by running the event.get() method.

Line 366 checks to see if the events value is equal to QUIT. If it is, then the exited variable will be set to False and the loop will stop (Line 367).

Until the event is equal to QUIT, the game will continue refreshing itself and updating itself every frame. Because of this, we need to also tell the program what the FRAMERATE is. That is what Line 368 does.

Line 370 contains a comment.

41. That is the end of our game! If you run the module now, you should see a functioning game that lets you set traps. The game should track your pizza bucks, the time you have left, and the number of bad reviews you have received. If you win or lose the game, you should see a message display in your game window.

Final Game 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
MAX_BAD_REVIEWS = 3
WIN_TIME = FRAMERATE * 60 * 3

#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()
            if self.rect.x <= 100:
                counters.bad_reviews += 1
        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, timer):
        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
        self.timer = timer
        self.timer_rect = None
        self.bad_reviews = 0
        self.bad_reviews_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 draw_bad_reviews(self, game_window):
        if bool(self.bad_reviews_rect):
            game_window.blit(BACKGROUND, (self.bad_reviews_rect.x, self.bad_reviews_rect.y), self.bad_reviews_rect)
        bad_reviews_surf = self.display_font.render(str(self.bad_reviews), True, WHITE)
        self.bad_reviews_rect = bad_reviews_surf.get_rect()
        self.bad_reviews_rect.x = WINDOW_WIDTH - 150
        self.bad_reviews_rect.y = WINDOW_HEIGHT - 50
        game_window.blit(bad_reviews_surf, self.bad_reviews_rect)

    def draw_time(self, game_window):
        if bool(self.timer_rect):
            game_window.blit(BACKGROUND, (self.timer_rect.x, self.timer_rect.y), self.timer_rect)
        timer_surf = self.display_font.render(str(int((WIN_TIME - self.loop_count)/ FRAMERATE)), True, WHITE)
        self.timer_rect = timer_surf.get_rect()
        self.timer_rect.x = WINDOW_WIDTH - 250
        self.timer_rect.y = WINDOW_HEIGHT - 50
        game_window.blit(timer_surf, self.timer_rect)

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

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
exited = 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
            exited = 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)

#-------------------------------------------------------------
#Set win/lose conditions
    if counters.bad_reviews >= MAX_BAD_REVIEWS:
        game_running = False
    if counters.loop_count > WIN_TIME:
        game_running = False

#-------------------------------------------------------------
    #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
#-------------------------------------------------------------
#End of game loop

#Set end game message
end_font = pygame.font.SysFont("Arial", 25)

if exited:
    if counters.bad_reviews >= MAX_BAD_REVIEWS:
        end_surf = end_font.render('Game Over', True, WHITE)
    else:
        end_surf = end_font.render('You Win!!', True, WHITE)
    GAME_WINDOW.blit(end_surf, (350, 200))
    display.update()
#-------------------------------------------------------------
#Enable exit from end game loop
while exited:
    for event in pygame.event.get():
        if event.type == QUIT:
            exited = False
    clock.tick(FRAMERATE)

#Close end game loop

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