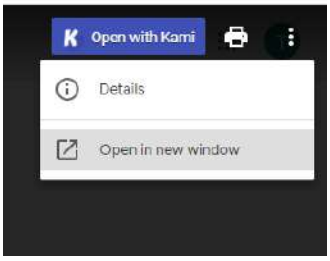
Chapter 11 Practice Directions

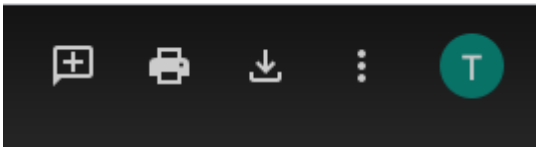
1. Before opening up IDLE, we need to save the game assets that we will be using this chapter to our game directory folder. Assets are images, videos, or other media that are not code but are part of your game. Navigate out to Google Classroom and find the assignment for this Chapter.

2. Select the garlic.png image to view the preview screen for the image.

3. Click the three dots at the top right corner and select the “Open in new window” option from the list.



4. In the new window that opens up, click the Download button in the top right corner to download the image.

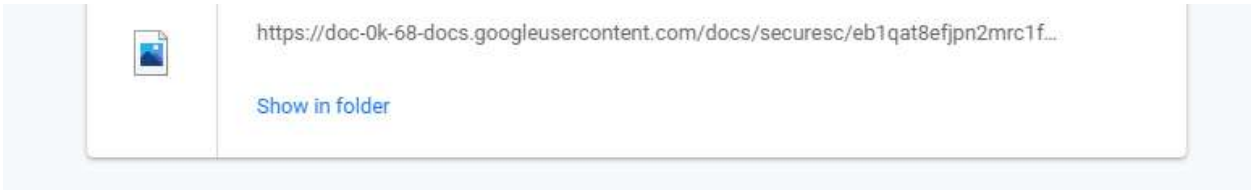


5. You will see your downloaded image appear in a task bar along the bottom of your browser window.

6. Our game assets must be saved in the same folder on the PC that our actual game file is saved in. That is why we created the game directory folder last chapter. Click the “Show All” button in the bottom right corner on the task bar.

7. A list of all of your Downloads will open up.

8. Click “Show in folder”.

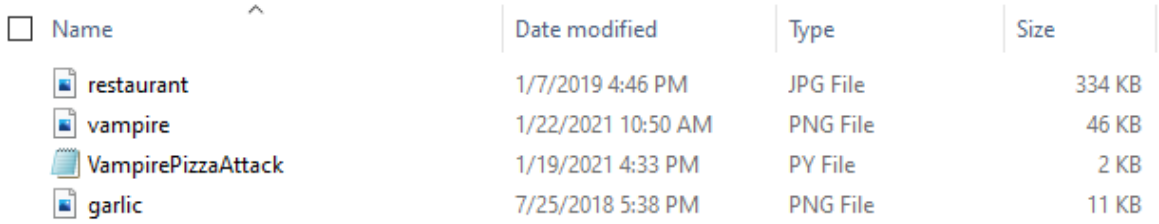


9. This will take you to your Downloads folder on your PC.

10. Copy or cut the garlic.png file.

11. Navigate to your vampire_pizza_directory folder on your H: or V: drive. You may have to click the “This PC” link in the menu at the left and then find your student drive and the appropriate folder.

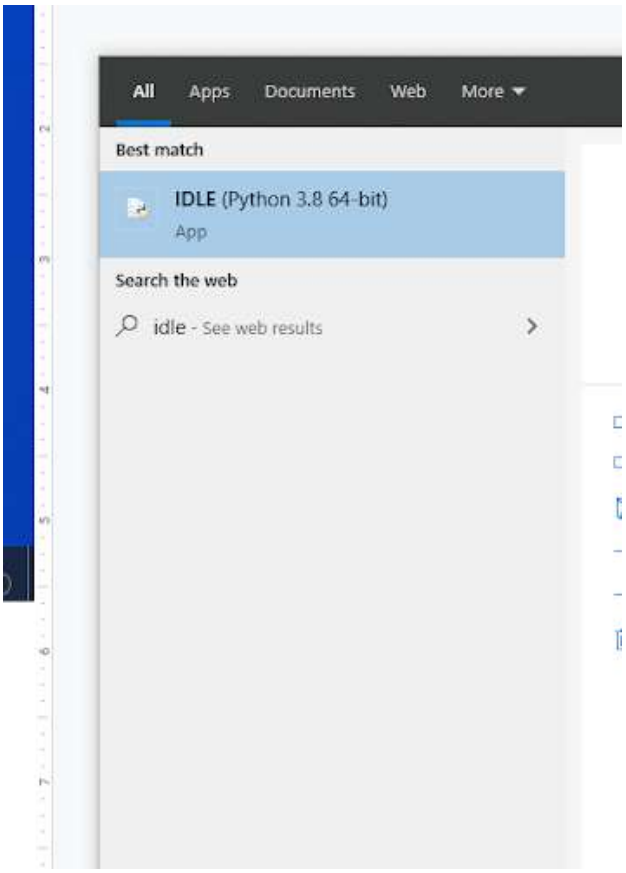
12. Paste the image file into your vampire_pizza_directory folder.



13. Repeat this process to download and save the pizzacutter.png and pepperoni.png images to your vampire_pizza_directory folder.

14. You can close out of your Windows Explorer window after you have saved all three images into it.

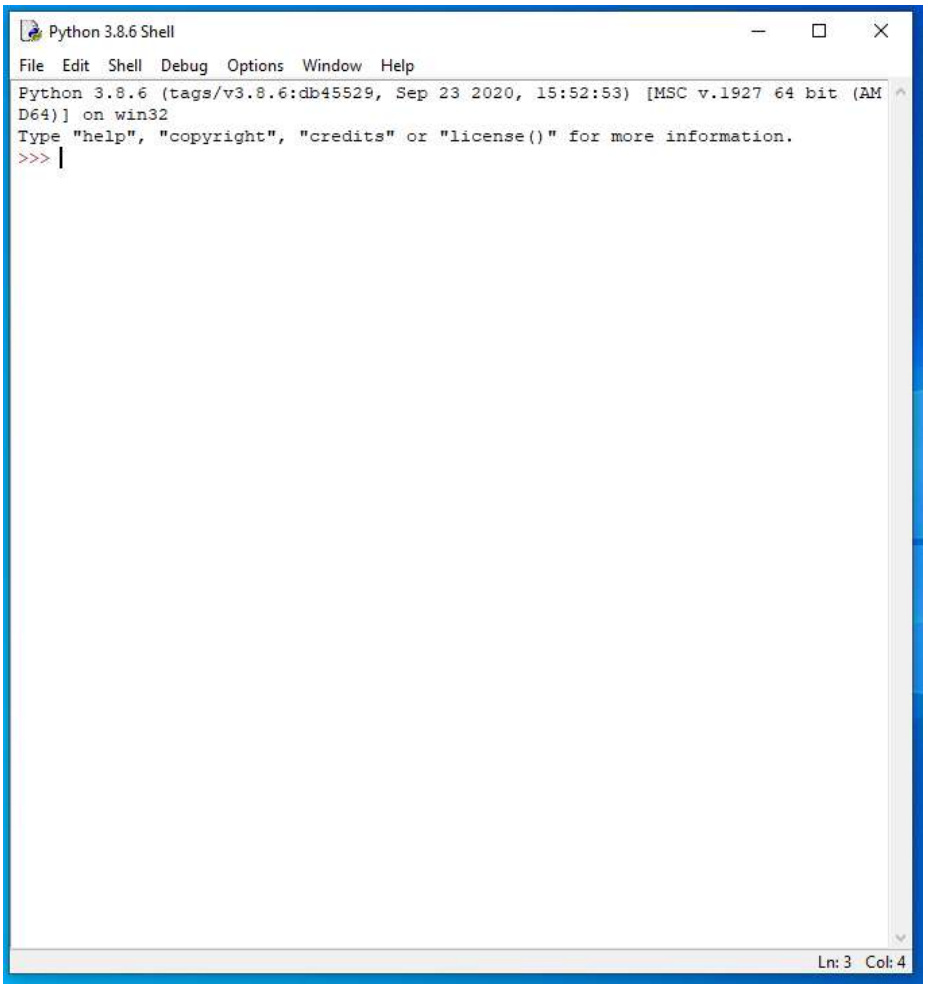
15. Using your Windows button menu, find and launch your IDLE program.



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.

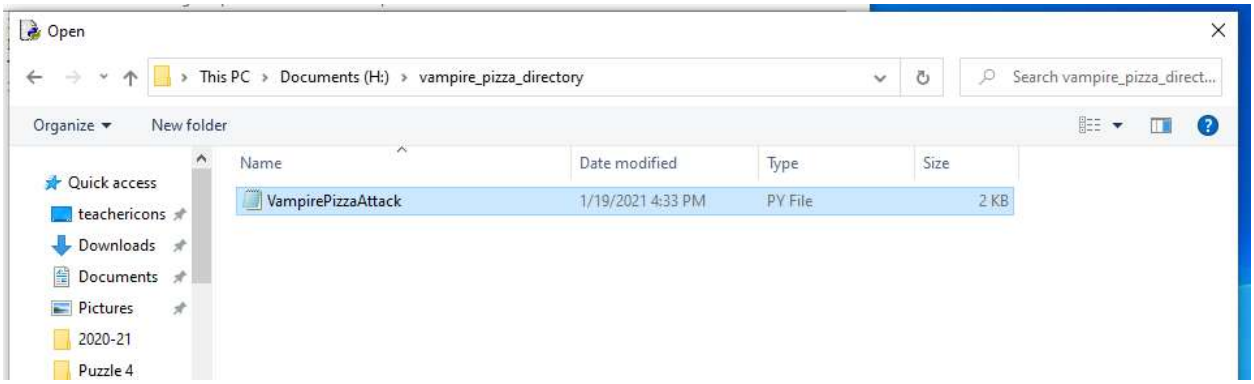
16. Your IDLE window should look something like this once it has launched.:



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.

17. Go to File > Open and then browse to find your VampirePizzaAttack file that we created and open it.



18. Your Python file and code from last chapter will open up.

19. 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.

    In this chapter, we will actually create the three types of traps that the player can use to attempt to stop or slow the vampire sprite's progress across the grid. Our three traps will be:

    a. Garlic: Used to slow down the vampire pizzas in their tracks. The garlic will cut the speed of the vampire pizza sprites in half.
    b. Wooden pizza cutters: Similar to wooden stakes but are specifically designed for pizza-shaped vampires. They do damage that will destroy the pizzas over time.
    c. Pepperoni: Allows you to earn pizza bucks faster so you can buy more traps.

20. Click at the end of Line 57, as shown in the screenshot below.

```
#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):
```

21. Press ENTER twice.

22. Type the code that you see on Lines 59 – 71 of the screenshot below.

```
#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
```

The code on Line 59 creates a variable called garlic_img. The value of the garlic_img variable is set to the garlic.png image.

Line 60 converts the image in the garlic_img variable (your garlic.png image) to a surface and assigns the newly created surface to a new variable called garlic_surf.

The code on Line 61 creates a new constant variable called GARLIC. The value of this variable will be set to a scaled image of the garlic_surf surface that you just created. The surface's size will be equal to the value of the WIDTH and HEIGHT variables.

The code on Line 62 sets the surface's alpha value. Alpha values range from 0 to 255 and represent the opacity or transparency of a surface. A surface with an alpha value of 255 is completely opaque, while a surface with an alpha value of 0 is completely transparent. If alpha values are not specified, than the alpha setting for a surface is disabled. Giving our surface an alpha value of 127 will make It slightly transparent.

The code on Lines 63 – 71 repeat the same procedure (loading an image, converting it to a surface, scaling it, and then giving it an alpha value) for the cutter.png and the pepperoni.png images.

23. Click at the end of Line 120, as shown in the screenshot below.

```
        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):
```

24. Press ENTER twice.

25. Type the code that you see on Lines 122 – 128 of the screenshot below. Ensure that your indentation matches what is shown in the screenshot.

```
    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_imag = trap_img

 #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):
```

    The code on Line 122 contains a comment.

    Line 123 establishes another class of objects. This class is called Trap.

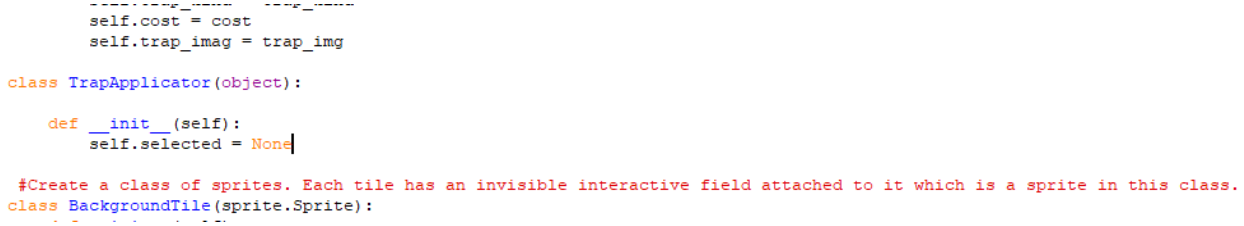
    Line 124 is a blank line.

    Line 125 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 trap_kind, cost, and trap_img. The values of these attributes for each instance of the Trap class can be different if the programmer chooses. The programmer will enter these values manually when the object is created.

    Lines 126 - 128 contain attribute specifications for each Trap object created. For example, Line 126 assigns the object's trap_kind attribute to match the trap_kind parameter/argument entered by the programmer when the object is created. Lines 127 and 128 do the same thing to assign the object's cost and trap_img attributes to match the parameters/arguments entered by the programmer when the object is created.

26. Press ENTER twice.

27. Type the code that you see on Lines 130 – 133 of the screenshot below. Ensure your indentation matches what is shown in the screenshot.



Line 130 creates another class called TrapApplicator. In the last section of code, we created a trap object. However, we need a separate tool to select the traps and apply them to the tiles. This is what we will call a TrapApplicator.

The TrapApplicator is just an object that keeps track of which trap the player wants to use and the tile on which the player wants to add the trap. Whenever an instance of the TrapApplicator is created, the player will have to specify the trap and the tile that they would like to use. This will make more sense at the end of the next chapter.

Line 131 is blank.

Line 132 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. Under the __init__ method, we specified that the value of the selected attribute for each new class object will be set to None.

28. Press ENTER twice.

29. Type the code you see on Lines 134 – 136 of the screenshot below. Ensure your indentation matches what is shown in the screenshot.

```
class TrapApplicator(object):

    def __init__(self):
        self.selected = None

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

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

Line 134 creates another class method called select_trap. The select_trap method takes the trap argument, so whenever this method is called, the programmer will have to tell the computer what trap has been selected.

Line 135 contains an if statement that checks to see if the value of that particular trap's cost is less than or equal to the value of the pizza_bucks variable that represents how much money the player has.

If the player has enough money to purchase the trap, Line 136 executes and the selected attribute for that class object is changed to "trap".

30. Press ENTER twice.

31. Type the code that you see on Lines 138 – 139 of the screenshot below. Ensure your indentation matches what is shown in the screenshot.

```
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):
```

Line 138 creates another class method called select_tile. This method takes the tile and counters argument.

Line 139 changes the value of the self.selected attribute to be the result of the set_trap method. We haven't yet created the set_trap method so this line won't completely make sense until the next chapter. However, just know that this method will be used to place the selected trap on the selected tile.

32. Click at the end of Line 154, 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)

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

33. Press ENTER twice.

34. Type the code you see on Lines 156 – 160 of the screenshot below.

```
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
```

The code on Line 156 creates the SLOW constant variable. The value of the SLOW variable is set to be a new Trap class object. This class object will be assigned the value of SLOW for the trap_kind object attribute, 5 for the cost attribute, and GARLIC for the trap_img attribute. Essentially, you are setting the trap_kind value for this new object to SLOW, the cost of this new class object to 5, and the image displayed by this new class attribute to be the GARLIC surface/image.

Lines 157 and 158 do the same thing to create new class objects using the CUTTER and PEPPERONI surface images, different costs (3 and 7), and different trap_kind values (DAMAGE and EARN).

Line 159 is blank.

Line 160 creates a variable called trap_applicator and sets its value to be a new class object using the TrapApplicator() class.

What these five lines of code have done is create three Trap class object instances, each using different specifications for traps, and an object instance of the TrapApplicator class.

35. Click at the end of Line 203, 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

        #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
```

36. Revise the code on Line 203 to match what is shown in the screenshot below.

```
        #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)

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

The revision to the code on Line 203 tells the trap_applicator object to run its select_tile method at the location that the use has clicked on the grid. It also uses the counters variable to figure out how much money the player has. This will be relevant in a later chapter.

37. Click at the end of Line 214, 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:
            GAME_WINDOW.blit(BACKGROUND, (tile.rect.x, tile.rect.y), tile.rect)

    for vampire in all_vampires:
```

38. Press ENTER.

39. Modify the code in this block to appear like the code shown in the screenshot below. Ensure your indentation matches what is shown.

```
#------------------------------------------------------------
#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:
```

The modification you have just made will check to see if the value of the trap variable for that tile is set to True or False. If it is True, the background image will be blitted over that particular trap tile's location. Essentially, this line of code will allow the background image to cover up the trap's location. This isn't relevant right now, but we will need this feature in the next chapter when we work on moving traps across the grid from one tile to another.

40. Go to File > Save.

41. You may choose to run your module, but you will probably end up with some errors because we haven't done all the the programming for the various traps we have in our game.

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))

    #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)

#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_imag = 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):
        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)

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):
        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()
            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 = 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()
```