Michael Arianno Chandrarieta - 2802499711

game_sprites.py

> Button
> Player
> Hitbox
> Bomb
> Enemy
> Bullet
> Explosion
> Spawner
> Pick_up
> Score_tab
> Background
> Cloud

Button

1. Imports statements

```
#Import needed module
import pygame, math, random
```

Imports the necessary libraries: pygame for game development, math for bullet generation, and random for random number generation.

2. Button

Class Definition

```
class Button(pygame.sprite.Sprite):
```

The Button class inherits from pygame.sprite.Sprite, which means it can take advantage of Pygame's sprite functionality, such as grouping and collision detection.

Constructor (__init__ method)

```
def __init__(self, xy_pos, message, colour):
```

Michael Arianno Chandrarieta - 2802499711

The constructor initializes an instance of the Button class with three parameters:

xy_pos: A tuple (x, y) representing the button's position on the screen.
message: A string that represents the text label of the button.
colour: The initial color of the button text.

Inside the Constructor

```
# Call the parent __init__() method
pygame.sprite.Sprite.__init__(self)
```

This line calls the parent class's constructor to ensure the sprite is properly initialized.

```
self.__message = message
self.__font = pygame.font.Font("fonts/Pixeltype.ttf", 40)
self.__select = 0
self.__colours = [colour, (255,99,71)]
```

self.__message: Stores the button's label message.
self.__font: Loads a font from the specified file and sets the font size to 40.
self.__select: Initializes a selection state variable (0 means not selected).
self.__colours: A list containing two colors: the initial color and a different color (in this case, a shade of red).

```
self.image = self.__font.render(
    message, 1, (self.__colours[self.__select]))
```

This line renders the button's text as an image using the specified font and color based on the __select state.

```
self.rect = self.image.get_rect()
self.rect.center = xy_pos
```

self.rect: Gets the rectangular area of the button image, which is used for positioning and collision detection.
self.rect.center: Sets the center of the rectangle to the provided xy_pos, effectively positioning the button on the screen.

Method: set_select

```
def set_select(self):
    '''This method sets the select instance to 1 as it is selected in main.
    '''

    #Set instance to 1.
    self.__select = 1
```

This method is used to change the selection state of the button.

```
#Set instance to 1.
self.__select = 1
```

It sets the __select variable to 1, indicating that the button is now selected. This could be used to change the button's appearance (e.g., change color) when the user hovers over or selects it.

```
def update(self):
    '''This method run automatically called every frame to determine if the
    button should change colour depending on if it is selected.'''

    #Reset colour of label depending on if selected.
    self.image = self.__font.render(
        self.__message, 1, (self.__colours[self.__select]))

    #Reset, it will be 1 if it is still selected next frame.
    self.__select = 0
```

Automatically called every frame to update the button's appearance based on its selection state.

Uses self.__font.render() to create a new image for the button's label.
The color of the label is determined by the __select variable:

- If __select is 0, the button renders in its original color.
- If __select is 1, the button renders in a different color (indicating it is selected).
-

Resetting Selection State:

After updating the appearance, the __select variable is reset to 0.
This reset ensures that the button does not remain in the selected state in the next frame unless it is actively selected again.

Michael Arianno Chandrarieta - 2802499711

Player

1. Initialization (__init__ method)

```
class Player(pygame.sprite.Sprite):
    '''The player class sprite. This class sprite is the player model being
    controlled by the user.'''

    Tabnine | Edit | Test | Explain | Document | Ask
    def __init__(self, screen):
        '''This method initializes the sprite using the screen parameter to
        set boundaries in update.'''

        # Call the parent __init__() method
        pygame.sprite.Sprite.__init__(self)

        #Set up and load images for animation frames while idle
        self.__station_frames = []
        for index in range(5):
            self.__station_frames.append(pygame.image.load("images/player"
                +str(index)+".png").convert_alpha())

        #Load images for animation while turning.
        self.__turn_frames_left = []
        for index in range(5):
            self.__turn_frames_left.append(pygame.image.load(
                "images/player_turn"+str(index)+".png").convert_alpha())

        #Fliped all left turn frames for right turning animation frames.
        self.__turn_frames_right = []
        for frame in self.__turn_frames_left:
            self.__turn_frames_right.append(pygame.transform.flip(frame, 1,0))
```

Image Loading: The class loads different images for the player's animations:

Stationary Frames: Loaded from files named player0.png to player4.png.
Turning Frames: Loaded for turning left and then flipped for turning right.

```
#Invisible image, used when invincible
self.__temp = pygame.image.load("images/temp.png").convert_alpha()

#Set other instances needed for this class.
self.__frames = self.__station_frames
self.image = self.__frames[0]
self.rect = self.image.get_rect()
self.rect.center = ((screen.get_width()-200)/2,
                    screen.get_height()- 2*self.rect.height)
self.__index = 0 #For animation image
self.__frame_refresh = 2 #For animation refresh
self.__temp_frame_refresh = self.__frame_refresh
self.__screen = screen
self.__dx = 0
self.__dy = 0
self.__diagonal = 1
self.__focus = 1
self.__invincible_frames = 110
self.__lock = 0
self.__shoot = 0
self.__cool_rate = 3
self.__focus_cool_rate = 9
self.__temp_cool_rate = self.__cool_rate
self.__temp_invincible = 0

#Set up spawn
self.reset()
```

Invisible Image: Used when the player is invincible.
Positioning: The player is initially positioned at the center-bottom of the screen, allowing for movement within the defined boundaries.
Movement Variables: Initializes various attributes related to movement (__dx, __dy), focus and shooting modes, cooldown rates, and invincibility frames.

2. Movement Control

Michael Arianno Chandrarieta - 2802499711

```
def change_direction(self, xy_change):
    '''This method takes a xy tuple parameter to change the direction of
    the player.'''

    #Multiply appropriate vectors by a magnitude of 8
    if xy_change[0] != 0:
        self.__dx = xy_change[0]*8
    elif xy_change[1] != 0:
        self.__dy = xy_change[1]*8
    #No vectors if idle
    else:
        self.__dx = 0
        self.__dy = 0

    #Toggle/untoggle diagonal mode when depending on if both value = 1.
    if self.__dx != 0 and self.__dy != 0:
        self.diagonal_mode(1)
    else:
        self.diagonal_mode(0)
```

Direction Change: The change_direction method updates the player's movement based on input (an x and y tuple). It calculates the movement vectors and adjusts for diagonal movement if both x and y are non-zero.

```
def diagonal_mode(self, mode):
    '''This method toggle/untoggles the diagonal mode depending on the
    mode parameter (boolean). This is used to ensure consistency
    amongst movement.'''

    #change factor used in update to approximately 0.7071.
    if mode:
        self.__diagonal = ((2.0**0.5)/2.0)
    else:
        self.__diagonal = 1
```

Diagonal Mode: The diagonal_mode method adjusts a factor to ensure consistent movement speed when moving diagonally.

```
def focus_mode(self, mode):
    '''This method toggles/untoggles the focused mode depending on the mode
    parameter (boolean). This is used to toggle between different speeds and
    fire types.'''

    #Change focus factor to the correct value depending on mode.
    if mode:
        self.__focus = 1.75
    else:
        self.__focus = 1
```

Focus Mode: The focus_mode method toggles between normal and focused movement speeds.

```
def shoot_mode(self, mode):
    '''This method toggles/untoggles the shoot mode depending on the mode
    parameter (boolean). This is used to toggle constant firing and non-
    firing.'''

    #Change boolean value
    self.__shoot = mode
```

Shooting Mode: The shoot_mode method toggles the shooting capability of the player.

3. Bullet Management

```
def spawn_bullet(self):
    '''This method spawns bullets, used in main. This spawns different
    bullets with different pattern depending on focus type.'''

    #If focused, shoot stream of bullet, and reset the cool down with
    #appropriate cool rate.
    if not self.__focus == 1:
        self.__temp_cool_rate = self.__cool_rate
        return Bullet(self.__screen, self, 0)
    #Unfocused shoots muti-bullets. Resetting with appropriate cool rate.
    else:
        self.__temp_cool_rate = self.__focus_cool_rate
        return [Bullet(self.__screen, self, 1, 60),
                Bullet(self.__screen, self, 1, 75),
                Bullet(self.__screen, self, 1, 90),
                Bullet(self.__screen, self, 1, 105),
                Bullet(self.__screen, self, 1, 120)]
```

Bullet Spawning: The spawn_bullet method creates bullets based on the player's focus state. If in focus mode, it shoots a single bullet; otherwise, it shoots multiple bullets in a spread pattern.

4. Player State Management

```
def reset(self):
    '''This method resets the player to original spawn point. Set up for
    new spawn.'''

    #Reposition player outside of the bottom screen.
    self.rect.center = ((self.__screen.get_width()-200)/2,
                        self.__screen.get_height() + 4*self.rect.height)

    #Set invincible mode for certain frames.
    self.__temp_invincible = self.__invincible_frames

    #Disable player shoot mode and focus mode.
    self.__shoot = 0
    self.__focus = 1
```

Resetting the Player: The reset method repositions the player and sets initial states, such as making the player temporarily invincible after respawning.

```
Tabnine | Edit | Test | Explain | Document | Ask
def set_invincible(self, frames):
    '''This method sets the player invincible frames to the amount
    indicated by the frames.'''

    #Set invincible frames
    self.__temp_invincible = frames
```

Invincibility Management: The set_invincible method allows the player to become invincible for a specified number of frames.

5. Accessor Methods

```
def get_cool_rate(self):
    '''This method returns the self instance, cool rate. Used in main to
    see if player can fire.'''

    #Return instance value.
    return self.__temp_cool_rate
```

get_cool_rate: Returns the current cooldown rate for firing bullets.

```
def get_invincible(self):
    '''This method returns the self instance, invincible. Used to detect
    if bullet can harm player in main.'''

    #Return instance value.
    return self.__temp_invincible
```

get_invincible: Indicates if the player is currently invincible.

```
def get_lock(self):
    '''This method returns the self instance, lock. Used to determine if
    user can control player sprite.'''

    #Return instance.
    return self.__lock
```

get_lock: Checks if the player is currently locked and cannot move.

```
def get_shoot(self):
    '''This method returns the self instance, shoot mode. Used in main to
    see if player can fire.'''

    #Return instance.
    return self.__shoot
```

get_shoot: Returns the shooting state (enabled or disabled).

```
def get_center(self):
    '''This method returns the self instance, center. Used in respawn and
    positioning.'''

    #Return instance.
    return self.rect.center
```

get_center: Provides the current center position of the player sprite.

6. Update Method

The update method is called every frame to:

```
def update(self):
    '''Update method used to update animation frames, cooldowns, and
    movement.'''

    #Invincible frames tick and image mannipulation to invisible.
    if self.__temp_invincible > 0:
        #Movement lock while invincible for 30 frames. Lock player.
        if self.__temp_invincible >= self.__invincible_frames-50:
            self.__lock = 1
            self.__dx, self.__dy = 0,0
            #Move up only
            self.__dy = -5
        #Unlock player.
        else:
            self.__lock = 0

        #Stop movement from lock when end position reached.
        if not self.__lock and self.__dy == -5:
            self.__dy = 0
```

Manage Invincibility: Controls movement and visibility when the player is invincible. The player may be locked and moved upwards during this state.

Michael Arianno Chandrarieta - 2802499711

```
#Switch to appropriate animation frames (stationary or turning).
if self.__dx < 0:
    self.__frames = self.__turn_frames_left
elif self.__dx > 0:
    self.__frames = self.__turn_frames_right
else:
    self.__frames = self.__station_frames

#Update sprite animation frames
if self.__temp_frame_refresh > 0:
    self.__temp_frame_refresh -= 1
else:
    self.__temp_frame_refresh = self.__frame_refresh
    self.__index += 1
    self.image = self.__frames[self.__index % len(self.__frames)]

#Invincible frames tick and image mannipulation to invisible.
if self.__temp_invincible > 0:
    self.__temp_invincible -= 1
    #Every 15 frames and every 3 frames close by will be invisible.
    if self.__temp_invincible % 15 <= 6 and \
       self.__temp_invincible % 15 > 2:
        self.image = self.__temp
```

Animation Handling: Switches between animation frames based on the current movement direction and updates the displayed image.

```
#Update sprite position using boundaries.
#Horizontal position.
if ((self.rect.left > 0) and (self.__dx < 0)) or\
   ((self.rect.right < self.__screen.get_width()-200) and\
    (self.__dx > 0)):
    self.rect.centerx += self.__dx/self.__focus*self.__diagonal
#Vertical position
if ((self.rect.top > 0) and (self.__dy < 0)) or\
          ((self.rect.bottom < self.__screen.get_height()) and\
           (self.__dy > 0)):
    self.rect.centery += self.__dy/self.__focus*self.__diagonal

#Cool down control.
if self.__temp_cool_rate > 0 :
    self.__temp_cool_rate -= 1
```

Position Updating: Adjusts the player's position based on movement vectors while ensuring that they stay within screen boundaries.

Cooldown Management: Decreases the cooldown for shooting, allowing the player to fire again after a certain period.

Michael Arianno Chandrarieta - 2802499711

Hitbox

1. Initialization (__init__ method)

```
def __init__(self, screen, player):
    '''This method initializes the sprite using the player sprite.'''

    # Call the parent __init__() method
    pygame.sprite.Sprite.__init__(self)

    #Image loading
    self.__hitbox = pygame.image.load("images/hitbox.png")\
        .convert_alpha()
    self.__temp = pygame.image.load("images/temp.png").convert_alpha()

    #Instance value setting.
    self.image = self.__hitbox
    self.rect = self.image.get_rect()
    self.__player = player
    self.__screen = screen
```

This method initializes the hitbox, loading the necessary images and setting its initial position.

2. Positioning the Hitbox

```
def position(self, player):
    '''This method uses the player sprite instance to reposition itself.'''

    #Mutate self center.
    self.rect.center = player.rect.center
```

The position method updates the hitbox's position to match the player's position.

3. Visibility Control

```
def set_visible(self, visible):
    '''This method uses the visible parameter (boolean), to set image from
    visible to invisible.'''

    #Change image depending on if visible
    if visible:
        self.image = self.__hitbox
    else:
        self.image = self.__temp
```

The set_visible method allows the hitbox to toggle between visible and invisible states based on a boolean parameter.

4. Updating the Hitbox

```
def update(self):
    '''This sprite updates the position of the hitbox sprite. using a
    method.'''

    #Position hit box in the center of the player sprite.
    self.position(self.__player)

    #Set invisible if outside bottom of screen - player reset property
    if self.rect.top > self.__screen.get_height():
        self.set_visible(0)
```

The update method is called every frame to update the hitbox's position and visibility based on the player's state.

Michael Arianno Chandrarieta - 2802499711

Bomb

1. Initialization (__init__ method)

```
    def __init__(self, xy_position):
        '''This method initializes the class using the xy parameter
        (tuple position) to start bomb at a point.'''

        # Call the parent __init__() method
        pygame.sprite.Sprite.__init__(self)

        #Set instance values.
        self.__side = 20
        self.image = pygame.Surface((self.__side,self.__side))
        self.rect = self.image.get_rect()
        self.__start = xy_position
        self.rect.center = xy_position
        self.__finish_raidus = 700
        self.__expand = 30
        self.__width = 3
```

This method sets up the bomb's initial properties, including its size, position, and state.

2. Getting the Bomb's Size

```
def get_side(self):
    '''This method returns the instance side. Used in accurate rect
    detection in main.'''

    #Return instance.
    return self.__side
```

The get_side method allows other parts of the program to access the current size of the bomb.

3. Updating the Bomb

Michael Arianno Chandrarieta - 2802499711

```
def update(self):
    ''''This method updates the bomb by increasing the size, the width of
    the circle drawn and to seemingly animate the bomb to expand off the
    screen.'''

    #If not finished expanding, continue.
    if self.__side/2 < self.__finish_raidus:
        self.__side += self.__expand
        self.__width += 1

        #Create new surface with the new size
        self.image = pygame.Surface((self.__side,self.__side))

        #Make background invisible
        self.image.set_colorkey((0,0,0))

        #Draw circle in surface.
        pygame.draw.circle(self.image, (255,255,255), (self.__side//2
                                , self.__side//2), self.__side//2, self.__width)

        #Reset rect for proper collision.
        self.rect = self.image.get_rect()
        self.rect.center = self.__start

    #If done, kill bomb.
    else:
        self.kill()
```

The update method is called every frame to handle the bomb's expansion and to redraw it based on its current size.

Michael Arianno Chandrarieta - 2802499711

Enemy

1. Initialization (__init__ method)

```
def __init__(self, screen, enemy_type = 1):
    '''This method initializes the enemy sprite with correct properties
    according to enemy type parameter. The screen parameter is used in for
    boundaries in update.'''

    # Call the parent __init__() method
    pygame.sprite.Sprite.__init__(self)

    #Set up and load animation frames.
    self.__unlock_frames = []
    for index in range(4):
        self.__unlock_frames.append(pygame.image.load("images/fairy"
            +str(enemy_type)+"_"+str(index)+".png").convert_alpha())

    #Load frames of turning animation for boss type enemies.
    self.__lock_frames_right = []
    self.__lock_frames_left = []
    if enemy_type < 4:
        #Create right turning frames from image files.
        for index in range(4):
            self.__lock_frames_right.append(pygame.image.load("images/turn"
                +str(enemy_type)+"_"+str(index)+".png").convert_alpha())

        #Left turning frames created by fliping right turning frames
        for frame in self.__lock_frames_right:
            self.__lock_frames_left.append(pygame.transform.flip(frame,
                                                                 1, 0))
```

This method sets up the enemy's properties based on the enemy type and initializes its animation frames.

```
#Setting default properites
self.__frames = self.__unlock_frames
self.image = self.__frames[0]
self.__down_frames = 0
self.__active_frames = 0
self.__cool_rate = 0
self.__dx = 0
self.__dy = 0
self.__hp = 0
self.__degs_change = 0
self.rect = self.image.get_rect()
self.rect.center = (400-50*enemy_type, 340-50*enemy_type)
self.__screen = screen
self.__target_degs = None
self.__target_y = screen.get_height()+self.rect.height
self.__enemy_type = enemy_type
self.__index = 0
self.__frame_refresh = 2
self.__images = 4
self.__lock = 0
self.__killed  = 0
```

Default properties

```
#Setting special enemy instance values depending on enemy type.
if enemy_type == 1:
    self.__down_frames = 60
    self.__active_frames = 60
    self.__cool_rate = 5
    self.__hp = 35
elif enemy_type == 2:
    self.__down_frames = 30
    self.__active_frames = 40
    self.__cool_rate = 10
    self.__hp = 40
elif enemy_type == 3:
    self.__down_frames = 0
    self.__active_frames = 12
    self.__cool_rate = 4
    self.__hp = 45
    self.__degs_change = 6
elif enemy_type == 4:
    self.__down_frames = 60
    self.__active_frames = 15
    self.__cool_rate = 3
    self.__hp = 10
elif enemy_type == 5:
    self.__down_frames = 30
    self.__active_frames = 30
    self.__cool_rate = 15
    self.__hp = 15
```

Enemy Default properties

```
#Set up initial spawn position
self.setup()

#Set temps used in countdowns.
self.__temp_down_frames = self.__down_frames
self.__temp_frame_refresh = self.__frame_refresh
self.__temp_active_frames = self.__active_frames
self.__temp_cool_rate = self.__cool_rate
```

Setup initial spawn and temporaries for countdowns

Michael Arianno Chandrarieta - 2802499711

2. Setting Up the Enemy

```
def setup(self):
    '''This method sets up the sprite when it spawns. This method decides
    where the sprite spawns, and where it goes.'''

    #Spawn location setting.
    x_pos = random.randrange(self.rect.width,
        self.__screen.get_width()-201-self.rect.width)
    y_pos = 0-self.rect.height
    self.rect.center = (x_pos, y_pos)

    #Static target position for boss type sprites.
    if self.__enemy_type < 4:
        target_x = random.randrange(100,
            self.__screen.get_width()-201-self.rect.width, 100)
        target_y = random.randrange(100,
            self.__screen.get_height()-229, 50)

        #Get the degrees between the target position and starting position.
        degs = self.calc_degs(target_x,target_y)
        #Get the appropriate vector movement according to degrees.
        self.__dx = math.cos(math.radians(degs)) * 5
        self.__dy = -(math.sin(math.radians(degs)) * 5)

        #Save the target y position.
        self.__target_y = target_y

        #Disable shooting at start when moving to target position.
        self.__lock = 1

    #Common type sprites randomly goes down screen.
    elif self.__enemy_type >= 4:
        self.__dx = 1
        self.__dy = 2
        if x_pos > (self.__screen.get_width()-200)/2:
            self.__dx = -self.__dx
```

The setup method initializes the enemy's spawn position and movement behavior.

3. Shooting Bullets

Michael Arianno Chandrarieta - 2802499711

```
def spawn_bullet(self, target):
    '''This method accepts a target parameter (a sprite) and use it to
    spawn/return bullets. The pattern will vary depending on the enemy
    types.'''

    #Get degrees using the target and the shooter.
    degs = self.calc_degs(target.rect.centerx, target.rect.centery)
```

The spawn_bullet method generates bullets based on the enemy type and target position.

Type 1 Enemy:

```
#Type 1, fire 1 bullet that tracks in the direction of the target
#with some variation.
if self.__enemy_type == 1:
    self.__target_degs = degs
    vary = random.randrange(-10, 26, 2)
    self.__target_degs += vary
    self.__temp_cool_rate = self.__cool_rate
    return Bullet(self.__screen, self, self.__enemy_type+1,
                  self.__target_degs)
```

Type 2 Enemy:

```
#Type 2, fire three bullets in a triple spread pattern towards target
#with little variation.
elif self.__enemy_type == 2:
    if self.__target_degs == None:
        self.__target_degs = degs
        vary = random.randrange(-2, 12, 2)
        self.__target_degs += vary
    self.__temp_cool_rate = self.__cool_rate
    return [Bullet(self.__screen, self, self.__enemy_type+1,
                   self.__target_degs-50),
            Bullet(self.__screen, self,
                   self.__enemy_type+1, self.__target_degs-25),
            Bullet(self.__screen, self, self.__enemy_type+1,
                   self.__target_degs),
            Bullet(self.__screen, self, self.__enemy_type+1,
                                        self.__target_degs+25),
            Bullet(self.__screen, self, self.__enemy_type+1,
                                        self.__target_degs+55)]
```

Type 3 Enemy:

```
#Type 3, Fire four bullets in a 90 degree gap each. The degree of
#direction will change and rotate as frames pass by.
elif self.__enemy_type == 3:
    if self.__target_degs == None:
        self.__target_degs = 0
    factor = random.randrange(1,4,2)
    if self.__target_degs < 0 and self.__degs_change < 0:
        self.__degs_change = 9 * factor
    elif self.__target_degs > 180 and self.__degs_change > 0:
        self.__degs_change = -9 * factor
    self.__target_degs += self.__degs_change
    self.__temp_cool_rate = self.__cool_rate
    return [Bullet(self.__screen, self, self.__enemy_type+1,
                   self.__target_degs),
            Bullet(self.__screen, self, self.__enemy_type+1,
                   self.__target_degs+90),
            Bullet(self.__screen, self, self.__enemy_type+1,
                   self.__target_degs+180),
            Bullet(self.__screen, self, self.__enemy_type+1,
                   self.__target_degs+270)]
```

Type 4 Enemy:

```
#Type 4, fire at target will a lot of variation in direction.
elif self.__enemy_type == 4:
    self.__target_degs = degs
    vary = random.randrange(-16, 30, 1)
    self.__target_degs += vary
    self.__temp_cool_rate = self.__cool_rate
    return Bullet(self.__screen, self, self.__enemy_type+1,
                  self.__target_degs)
```

Michael Arianno Chandrarieta - 2802499711

Type 5 Enemy:

```
#Type 5, spread bullets in all directions, 60 degrees gap.
elif self.__enemy_type == 5:
    self.__target_degs = random.randrange(0, 360, 15)
    bullets = []
    for extra_degs in range(0, 360, 60):
        bullets.append(Bullet(self.__screen, self, self.__enemy_type+1,
            self.__target_degs+extra_degs))
    self.__temp_cool_rate = self.__cool_rate
    return bullets
```

4. Accessor Methods

1. damaged(self, damage)

This method is called to apply damage to the enemy, reducing its health points (HP).

```
def damaged(self, damage):
    '''This method takes a damage parameter (integer). The damage parameter
    is used to decrease enemy/class health.'''

    #Mutate hp instance.
    self.__hp -= damage
```

Parameters: damage (integer) - The amount of damage to apply.
Functionality: Decreases the enemy's health by the specified damage amount.

2. set_killed(self)

This method marks the enemy as killed, preventing it from being processed multiple times for destruction.

```
def set_killed(self):
    '''This method mutates the killed instance to 1 to inform that the
    sprite is already killed. This makes it so that the sprite doesn't
    die mutiple times.'''

    #Set instance.
    self.__killed = 1
```

Functionality: Sets the internal __killed flag to 1, indicating that the enemy has been killed.

3. get_killed(self)

This method retrieves the status of whether the enemy has been killed.

```
def get_killed(self):
    '''This method returns the killed instance to inform main if sprite
    is already killed.'''

    #Return instance.
    return self.__killed
```

Returns: The value of __killed, which indicates if the enemy is dead.

4. get_cool_rate(self)

This method returns the cooldown rate of the enemy, which determines how often it can fire bullets.

```
def get_cool_rate(self):
    '''This method returns the cool rate of the enemy. Used in main to see
    when enemy can fire.'''

    #Return instance.
    return self.__temp_cool_rate
```

Returns: The current cooldown rate, which is used to manage firing intervals.

5. get_down_frames(self)

This method returns the number of frames the enemy must wait before it can fire again.

```
def get_down_frames(self):
    '''This method returns the down frames of the enemy. Used to see when
    the enemy can fire in main.'''

    #Return instance
    return self.__temp_down_frames
```

Returns: The remaining frames the enemy must wait before it can shoot again.

6. get_lock(self)

This method retrieves the lock status of the enemy, indicating whether it can shoot.

```
def get_lock(self):
    '''This method returns the lock mode value. Used to see if enemy can
    shoot.'''

    #Return instance
    return self.__lock
```

Returns: The lock status, which indicates if the enemy is currently unable to shoot.

7. get_hp(self)

This method returns the current health points of the enemy.

```
def get_hp(self):
    '''This method returns the hp of the enemy. Used to see when
    the enemy can be deleted in main.'''

    #Return instance.
    return self.__hp
```

Returns: The current HP of the enemy, which is used to determine if the enemy should be removed from the game.

8. get_center(self)

This method returns the center position of the enemy sprite, useful for positioning other elements relative to the enemy.

```
def get_center(self):
    '''This method returns the down frames of the enemy. Used to position
    sprites.'''

    #Return instance.
    return self.rect.center
```

Returns: The center coordinates (x, y) of the enemy's rectangle, which can be used for positioning other sprites or effects.

9. get_type(self)

This method returns the type of the enemy, which can be used to differentiate behavior or appearance.

```
def get_type(self):
    '''This method returns the enemy type to determine numbers of types on
    screen in main.'''

    #Return instance.
    return self.__enemy_type
```

Returns: The enemy type identifier, which can be used for logic related to different enemy behaviors or properties.

5. Updating the Enemy

```
def update(self):
    '''This method updates what happens to the enemy class as frames passes
    by. This method kills, moves, animates, and controls cool down of
    enemy class.'''

    #Decide to kill if enemy dies.
    if self.__killed or self.rect.top > self.__screen.get_height():
        self.kill()

    #While turning load appropriate turning frames
    if self.__lock and int(self.__dx) > 0:
        self.__frames = self.__lock_frames_right
    elif self.__lock and int(self.__dx) < 0:
        self.__frames = self.__lock_frames_left
    else:
        self.__frames = self.__unlock_frames

    #Animate frames, refresh counter
    if self.__temp_frame_refresh > 0:
        self.__temp_frame_refresh -= 1
    else:
        self.__temp_frame_refresh = self.__frame_refresh
        self.__index += 1
    self.image = self.__frames[self.__index % self.__images]
```

```
#Tick cool rate if appropriate.
if self.__temp_cool_rate > 0 and self.__lock != 1:
    self.__temp_cool_rate -= 1

#Reset down frames and active frames to original.
if self.__temp_active_frames == 0 == self.__temp_down_frames:
    self.__temp_down_frames = self.__down_frames
    self.__temp_active_frames = self.__active_frames

    #Set target degrees to none as appropriate to type 2 pattern
    if self.__enemy_type == 2:
        self.__target_degs = None

#Tick down frames and active frames if appropriate.
if self.__temp_down_frames > 0 and self.__lock != 1:
    self.__temp_down_frames -= 1
if self.__temp_active_frames > 0 and self.__temp_down_frames == 0:
    self.__temp_active_frames -= 1

#Change x vector when colliding with boundary for opposite direction.
if self.rect.left <= 0 and self.__dx < 0:
    self.__dx = abs(self.__dx)
elif self.rect.right >= self.__screen.get_width()-200 and self.__dx > 0:
    self.__dx = -self.__dx

#move class/sprite according to vectors.
self.rect.centerx += self.__dx

if not self.rect.centery >= self.__target_y:
    self.rect.centery += self.__dy
else:
    self.__dx = 0
    self.__dy = 0
    self.__lock = 0
```

The update method handles the enemy's movement, animation, and cooldown management.

Michael Arianno Chandrarieta - 2802499711

Explosion

1. Initialization (__init__ method)

This method sets up the explosion's initial properties, including loading the appropriate frames based on the explosion type.

```
def __init__(self, xy_position, explosion_type):
    '''This method initializes the class by using paramters starting
    position and type. The starting position is used as spawn point and
    explosion type is used to load appropriate image.
    '''
    # Call the parent __init__() method
    pygame.sprite.Sprite.__init__(self)

    #Set up and load frames depending on type
    self.__frames = []
    if explosion_type == 0:
        for num in range(4):
            self.__frames.append(pygame.image.load("images/death"
                                                   +str(num)+".png").convert_alpha())
    elif explosion_type == 1:
        for num in range(3):
            self.__frames.append(pygame.image.load("images/burst"
                                                   +str(num)+".png").convert_alpha())

    #Set up instances.
    self.image = self.__frames[0]
    self.rect = self.image.get_rect()
    self.rect.center = xy_position
    self.__frame_refresh = 1
    self.__temp_refresh = self.__frame_refresh
    self.__index = 0
```

Parameters:
xy_position: A tuple representing the x and y coordinates where the explosion will be displayed.
explosion_type: An integer that determines which set of explosion frames to load.

2. Updating the Explosion (update method)

The update method is called each frame to handle the animation of the explosion.

Michael Arianno Chandrarieta - 2802499711

```
def update(self):
    '''Update class as frames passes by. Control frame refresh, new frame
    and kill.'''

    #Frame tick
    if self.__temp_refresh > 0:
        self.__temp_refresh -= 1

    #Kill after animation
    else:
        if self.__index >= len(self.__frames)-1:
            self.kill()
        #Next frame if appropriate
        else:
            self.__index += 1
            self.image = self.__frames[self.__index]
        self.__temp_refresh = self.__frame_refresh
```

Functionality:

The method checks if the temporary refresh counter has reached zero. If it has, it updates the current frame of the explosion animation.
If the current frame index exceeds the number of frames, the explosion sprite is killed (removed from the game).
Otherwise, it advances to the next frame and updates the displayed image.

Michael Arianno Chandrarieta - 2802499711

Bullet

1. Initialization (__init__ method)

The constructor initializes the bullet's properties based on the provided parameters.

```
def __init__(self, screen, shooter, shoot_type, degs = None):
    '''This method initializes the bullet sprite using parameters such
    as screen, shooter of bullet, shoot type, and degrees.'''

    # Call the parent __init__() method
    pygame.sprite.Sprite.__init__(self)

    #Load appropriate image for bullet depending on shoot type.
    self.image = pygame.image.load("images/bullet"
                                   +str(shoot_type)+".png").convert_alpha()

    #Set up default values.
    self.rect = self.image.get_rect()
    self.rect.center = shooter.rect.center
    self.__screen = screen
    self.__dx = 0
    self.__dy = 0
    self.__grazed = 0
    self.__graze_frames = 10
    self.__shoot_type = shoot_type
    self.__temp_graze = self.__graze_frames
```

```
#Set unique bullet speed and direction depending on shoot type.
if shoot_type == 0:
    self.__dy = -20
elif shoot_type ==1:
    self.__dx = math.cos(math.radians(degs)) * 20
    self.__dy = -(math.sin(math.radians(degs)) * 20)
elif shoot_type == 2:
    self.__dx = math.cos(math.radians(degs)) * 6
    self.__dy = -(math.sin(math.radians(degs)) * 6)
elif shoot_type ==3:
    self.__dx = math.cos(math.radians(degs)) * 6
    self.__dy = -(math.sin(math.radians(degs)) * 6)
elif shoot_type ==4:
    self.__dx = math.cos(math.radians(degs)) * 6
    self.__dy = -(math.sin(math.radians(degs)) * 6)
elif shoot_type == 5:
    self.__dx = math.cos(math.radians(degs)) * 4
    self.__dy = -(math.sin(math.radians(degs)) * 4)
elif shoot_type == 6:
    self.__dx = math.cos(math.radians(degs)) * 4
    self.__dy = -(math.sin(math.radians(degs)) * 4)
```

Parameters:

screen: The Pygame screen where the bullet will be rendered.
shooter: The entity that fired the bullet, used to position the bullet.
shoot_type: An integer that determines the bullet's speed and direction.
degs: An optional parameter representing the angle in degrees for angled shots.

2. Grazing Mechanism (set_grazed and get_grazed methods)

These methods manage the grazing state of the bullet.

```
def set_grazed(self, mode):
    '''This method sets it so that the bullet is grazed.'''

    #Mutate grazed to true.
    self.__grazed = mode
```

```
def get_grazed(self):
    '''This method returns the grazed instance (boolean). Used in main
    for sprite positioning.'''

    #Return instance.
    return self.__grazed
```

set_grazed: Sets the grazing state of the bullet.
get_grazed: Returns the current grazing state.

3. Positioning (get_center method)

This method retrieves the current center position of the bullet.

```
def get_center(self):
    '''This method returns the center instance. Used in main for sprite
    positioning.'''

    #Return instance.
    return self.rect.center
```

4. Updating the Bullet (update method)

The update method is called every frame to move the bullet and manage its lifecycle.

Michael Arianno Chandrarieta - 2802499711

```
def update(self):
    '''This method will be called automatically to reposition the
    bullet sprite on the screen.'''

    #Reposition
    self.rect.centery += self.__dy
    self.rect.centerx += self.__dx

    #Graze reset
    if self.__grazed == 1:
        self.__temp_graze -= 1
        if self.__temp_graze == 0:
            self.__grazed = 0
            self.__temp_graze = self.__graze_frames

    #Kill bullet if out of screen for effciency.
    if (0 >= self.rect.bottom or self.rect.top >=
        self.__screen.get_height()) or (0 >= self.rect.right or
            self.rect.left >= self.__screen.get_width()-200):

        self.kill()
```

Movement: Updates the bullet's position based on its velocity (__dx and __dy).
Grazing Logic: Manages the grazing state and resets it after a certain number of frames.
Lifecycle Management: Kills the bullet if it moves out of the screen boundaries.

Michael Arianno Chandrarieta - 2802499711

Spawner

1. Initialization (__init__ method)

The constructor initializes the spawner's properties based on the provided parameters.

```
def __init__(self, screen, spawner_type):
        '''This method initializes the class with screen parameter (used to
        spawn enemy) and the type parameter (value). Determines if
        common enemy or boss enemy can be spawned.'''

        # Call the parent __init__() method
        pygame.sprite.Sprite.__init__(self)

        self.image = pygame.Surface((0,0))
        self.rect = self.image.get_rect()
        self.__type = spawner_type
        self.__screen = screen
        self.__lock = 0
        self.__spawn_types = []

        if self.__type == 0:
            self.__spawn_frames = 150
            self.__spawn_range = [4, 6]
        elif self.__type == 1:
            self.__spawn_frames = 300
            self.__spawn_range = [1, 4]

        self.__temp_frames = self.__spawn_frames
```

Parameters:

screen: The Pygame screen where enemies will be rendered.
spawner_type: An integer that determines the type of enemies to spawn (0 for common enemies, 1 for boss enemies).

2. Spawning Enemies (spawn_enemy method)

This method is responsible for creating and returning a new enemy instance.

```
def spawn_enemy(self):
    '''Spawn enemy if appropriate.'''

    #Reset temp.
    self.__temp_frames = self.__spawn_frames

    #Spawn appropriate enemy depending on type
    enemy_type = random.randrange(self.__spawn_range[0],
                                  self.__spawn_range[1])
    return Enemy(self.__screen, enemy_type)
```

Functionality: Resets the temporary frame counter and generates a random enemy type within the defined range, then returns a new Enemy instance.

3. Lock Management (set_lock method)

This method sets the lock state of the spawner, controlling whether it can spawn enemies.

```
def set_lock(self, mode):
    '''This method sets the instance lock between boolean values.
    Determines when to countdown spawn time.'''

    #Set instance to boolean.
    self.__lock = mode
```

4. Adjusting Spawn Rate (set_rate method)

This method modifies the spawn rate and enemy range based on the game's difficulty level.

```
def set_rate(self, difficulty):
    '''This method changes the spawn rate depending on its own spawn type
    and difficulty of the game. This method also changes the range of
    enemies that a spawner can summon depending on difficulty.'''

    #Common spawn type, decrease spawn rate by 15 frames each difficulty.
    if self.__type == 0:
        self.__spawn_frames = 150 - (difficulty*15)
        #Different enemy range in common spawner depending on difficulty.
        if difficulty == 0 or difficulty == 1:
            self.__spawn_range = [4,5]
        elif difficulty == 2:
            self.__spawn_range = [4,6]
    #Boss spawn type, decrease spawn rate by 30 frames each difficulty.
    elif self.__type == 1:
        self.__spawn_frames = 300 - (difficulty*30)
        #Different enemy range in boss spawner depending on difficulty.
        if difficulty == 0:
            self.__spawn_range = [1,2]
        elif difficulty == 1:
            self.__spawn_range = [1,3]
        elif difficulty == 3:
            self.__spawn_range = [1,4]
```

Functionality: Adjusts the spawn timing and the range of enemy types based on the difficulty level.

5. Frame Counting (get_spawn_frames method)

This method returns the current temporary frame counter.

```
def get_spawn_frames(self):
    '''This method returns the instance temp_frames to determine if spawn
    is possible in main.'''

    #Return instance.
    return self.__temp_frames
```

6. Type Identification (get_type method)

This method returns the type of spawner (common or boss).

```
def get_type(self):
    '''This method returns the instance type to determine if which limit
    in main should be looked at.'''

    #Return instance.
    return self.__type
```

7. Update (update method)

This method updates the spawner's state, decrementing the temporary frame counter if the spawner is not locked.

```
def update(self):
    '''Tick down spawn time if the spawner is not locked.'''

    if not self.__lock:
        if self.__temp_frames >= 0:
            self.__temp_frames -= 1
```

Michael Arianno Chandrarieta - 2802499711

Pick_up

1. Initialization (__init__ method)

The constructor initializes the pickup's properties based on the provided parameters.

```
def __init__(self, screen, enemy, drop_type):
        '''This method initializes the pick class using the screen parameter
        as bounaries, sprite to get spawn position and drop_type.'''

        # Call the parent __init__() method
        pygame.sprite.Sprite.__init__(self)

        #Set screen, drop type, and rate of speed update
        self.__screen = screen
        self.__type = drop_type
        self.__speed_frames = 5
        self.__temp_speed = self.__speed_frames

        #Image loading
        self.image = pygame.image.load("images/drop"
            +str(self.__type)+".png").convert_alpha()
        self.rect = self.image.get_rect()

        #Rect setting.
        self.rect = self.image.get_rect()
        self.rect.center = enemy.rect.center

        #Randomized initial speeds of drop
        self.__dx = random.randrange(-3,4)
        self.__dy = random.randrange(-2,3)
```

Parameters:

screen: The Pygame screen where the pickup will be rendered.
enemy: The enemy that drops the pickup, used to position the pickup at the enemy's location.
drop_type: An integer representing the type of pickup (e.g., points, lives, bombs), which determines the image loaded.

2. Getting the Pickup Type (get_type method)

This method returns the type of the pickup, which can be useful for scoring or gameplay logic.

```
def get_type(self):
    '''This method returns the self instance type to be passed in to score
    tab when colliding with player in main.'''

    #Return instance.
    return self.__type
```

3. Updating the Pickup (update method)

The update method is called automatically at the end of each frame to update the state of the pickup.

```
def update(self):
    '''This method is called automatically at the end of every frame to
    update the sprite.'''

    #Kill bullet if out of bottom for effciency.
    if (self.rect.top >= self.__screen.get_height()):
        self.kill()

    #Update speed if appropriate
    if self.__temp_speed == 0:
        #If x velocity is not 0 yet, get closer to it by 1.
        if self.__dx != 0:
            self.__dx -= (self.__dx / abs(self.__dx))

        #If y velocity is not 3 yet, get closer to it by 1.
        if self.__dy < 3:
            self.__dy += 1
        elif self.__dy > 3:
            self.__dy -= 1

        self.__temp_speed = self.__speed_frames

    else:
        self.__temp_speed -= 1

    #Update position using vectors.
    self.rect.centerx += self.__dx
    self.rect.centery += self.__dy
```

Functionality:

Michael Arianno Chandrarieta - 2802499711

Boundary Check: The pickup is removed from the game if it moves off the bottom of the screen.
Speed Adjustment: The method gradually adjusts the horizontal and vertical speeds of the pickup based on the specified logic, creating a smoother movement pattern.
Position Update: The pickup's position is updated based on its current speed.

Michael Arianno Chandrarieta - 2802499711

Score_tab

1. Initialization (__init__ method)

The constructor initializes the score tab with necessary parameters and loads relevant resources.

```
def __init__(self, screen):
    '''This method initializes the class with appropriate parameters.'''

    # Call the parent __init__() method
    pygame.sprite.Sprite.__init__(self)

    #Open data file for highscore.
    try:
        #Open, read highscore and save.
        load_data = open("data/highscore.txt", 'r')
        self.__highscore = int(load_data.read())
        #Close data file.
        load_data.close()

    #If file error, just set highscore to 0.
    except IOError and ValueError:
        #Brand new highscore.
        self.__highscore = 0

    #Load image as main surface.
    self.image = pygame.image.load("images/score_tab.png").convert()

    #Life frames image loading
    self.__life_frames = []
    for frame in range(2):
        self.__life_frames.append(pygame.image.load(
            "images/life"+str(frame)+".png").convert_alpha())

    #Bomb frames image loading
    self.__bomb_frames = []
    for frame in range(2):
        self.__bomb_frames.append(pygame.image.load(
            "images/bomb"+str(frame)+".png").convert_alpha())
```

```
#Set default instances.
self.rect = self.image.get_rect()
self.rect.center = (screen.get_width()-self.rect.width/2,\
                    screen.get_height()/2)
self.__screen = screen
self.__font = pygame.font.Font("fonts/Pixeltype.ttf", 40)
self.__score = 0
self.__scores = ["HIGHSCORE", ("%10s" %(str(self.__highscore))).replace(
    " ", "0"),
                 "SCORE",("%10s" %(str(self.__score))).replace(" ", "0")]
self.__score_labels = []
self.__lives = 2
self.__bombs = 1
self.__stats = ["LIVES", "BOMBS"]
self.__stat_labels = []
self.__score_colour = (255,255,255)
self.__stat_colour = (255,255,255)
self.__colour_frames = 15
self.__temp_colour_frames = self.__colour_frames
```

Parameters:

screen: The Pygame screen where the score tab will be displayed.

Key Actions:

Loads the high score from a file, defaulting to 0 if there's an error.
Loads images for the score tab, lives, and bombs.
Initializes instance variables for score, lives, bombs, and their respective labels.

2. Adding Points (add_points method)

This method updates the score based on the type of points added.

```
def add_points(self, point_type):
    '''This method add to the points value depending on the type of added
    points. The type determines how much points are added'''

    #Grazing points
    if point_type == 0:
        self.__score += 1
    #Enemy type based points 1-5
    elif point_type == 1:
        self.__score += 30
    elif point_type == 2:
        self.__score += 35
    elif point_type == 3:
        self.__score += 50
    elif point_type == 4:
        self.__score += 10
    elif point_type == 5:
        self.__score += 15
    #Drop type based points
    elif point_type == 6:
        self.__score += 5
    elif point_type == 7:
        self.__score += 10
    #Life and bomb drops. If full capacity points are given instead.
    elif point_type == 8:
        if self.__lives < 3:
            self.__lives += 1
        else:
            self.__score += 100
    elif point_type == 9:
        if self.__bombs < 3:
            self.__bombs += 1
        else:
            self.__score += 50
```

Functionality: The method checks the point_type and updates the score accordingly. It also handles life and bomb drops, incrementing them if the player has less than the maximum capacity.

3. Life Loss (life_loss method)

This method decrements the player's lives and resets the bombs.

```
def life_loss(self):
    '''This method mutates the lives instance by decreasing it by 1.'''

    #Decrease lives by 1 after death.
    self.__lives -= 1
    self.reset()
```

Functionality: The method decrements the __lives instance variable and calls the reset method to reset the bombs.

4. Reset (reset method)

This method resets the bomb count after a successful respawn.

```
def reset(self):
    '''This method resets the bomb count after successful respawn.'''

    #Reset bombs to 1 if game not over yet.
    if self.__lives != 0:
        self.__bombs = 1
```

Functionality: The method resets the __bombs instance variable to 1 if the player still has lives remaining.

5. Bomb Used (bomb_used method)

This method decrements the player's bombs.

```
def bomb_used(self):
    '''This method mutates the bombs instance by decreasing it by 1.'''

    #Decrease lives by 1 after death.
    self.__bombs -= 1
```

Functionality: The method decrements the __bombs instance variable.

6. Get High Score (get_highscore method)

This method returns the current high score.

```
def get_highscore(self):
    '''This method returns the instance highscore to be saved in main.'''

    #Return instance.
    return self.__highscore
```

7. Get Lives (get_lives method)

This method returns the current number of lives.

```
def get_lives(self):
    '''This method returns the instance lives to be read in main.'''

    #Return instance.
    return self.__lives
```

Functionality: The method returns the __lives instance variable.

8. Get Bombs (get_bombs method)

This method returns the current number of bombs.

```
def get_bombs(self):
    '''This method returns the instance lives to be read in main.'''

    #Return instance.
    return self.__bombs
```

Functionality: The method returns the __bombs instance variable.

9. Update (update method)

This method updates the score tab and its components.

```
def update(self):
    '''This method is called once per frame to update the score tab.'''

    #Compare scores, flash colours at constant rate if score is highscore.
    if self.__score >= self.__highscore:
        self.__highscore = self.__score
        if self.__temp_colour_frames > 0:
            self.__temp_colour_frames -=1
        if self.__temp_colour_frames == 0:
            if self.__score_colour == (255,255,255):
                self.__score_colour = (255,99,71)
            else:
                self.__score_colour = (255,255,255)
            self.__temp_colour_frames = self.__colour_frames

    #Refresh scores for refreshed labels
    self.__scores = ["HIGHSCORE", ("%10s" %(str(self.__highscore))).replace(
        " ", "0"),
                     "SCORE",("%10s" %(str(self.__score))).replace(" ", "0")]

    #Reset score labels
    self.__score_labels = []
    self.__stat_labels = []

    #Render all score labels
    for score in self.__scores:
        label = self.__font.render(score, 1, (self.__score_colour))
        self.__score_labels.append(label)

    #Render all stat labels
    for stat in self.__stats:
        label = self.__font.render(stat, 1, (self.__stat_colour))
        self.__stat_labels.append(label)
```

Michael Arianno Chandrarieta - 2802499711

```
#Blit image on to surface.
self.image = pygame.image.load("images/score_tab.png").convert()

#Blit labels accordinging to their y position.
y_pos = -20
for label_num in range(len(self.__score_labels)):
    if label_num %2 == 0:
        y_pos += 50
    else:
        y_pos += 25
    self.image.blit(self.__score_labels[label_num], (10,y_pos))

y_pos = 235
for label_num in range(len(self.__stat_labels)):
    y_pos += 75
    self.image.blit(self.__stat_labels[label_num], (10,y_pos))

#Blit 3 life and bomb images with appropriate shade depending on amount.
x_pos = 0
for num in range(1, 4):
    if self.__lives >= num:
        self.image.blit(self.__life_frames[1], (10+x_pos, 345))
    else:
        self.image.blit(self.__life_frames[0], (10+x_pos, 345))
    x_pos += 40
x_pos = 0
for num in range(1, 4):
    if self.__bombs >= num:
        self.image.blit(self.__bomb_frames[1], (10+x_pos, 420))
    else:
        self.image.blit(self.__bomb_frames[0], (10+x_pos, 420))
    x_pos += 40
```

Functionality: The method updates the score tab by:
Comparing the current score with the high score and flashing the score color if they match.
Refreshing the score labels and stat labels.
Rendering the labels with the updated scores and stats.
Blitting the labels onto the score tab surface.
Blitting life and bomb images with appropriate shades depending on the amount.

Michael Arianno Chandrarieta - 2802499711

Cloud

1. Initialization (__init__ method)

The constructor initializes the cloud's properties and sets its initial position.

```
def __init__(self, screen):
    '''This method initializes the cloud class using the screen parameter
    as bounaries.'''

    # Call the parent __init__() method
    pygame.sprite.Sprite.__init__(self)

    #Set instances, randomize properties upon creation.
    self.__screen = screen
    self.__type = 0
    self.__dx = 0
    self.__dy = 0
    self.random_type()
    self.random_speed()

    #Image loading
    self.image = pygame.image.load("images/cloud"
        +str(self.__type)+".png").convert_alpha()
    self.rect = self.image.get_rect()

    #Set position.
    self.random_reset()
```

Parameters:

screen: The Pygame screen where the cloud will be rendered.

Key Actions:

Initializes the cloud's properties and randomizes its type, speed, and position.
Loads the appropriate cloud image based on the randomized type.

2. Random Type (random_type method)

This method assigns a random type to the cloud, allowing for different cloud images.

```
def random_type(self):
    '''This method randomizes the type of the cloud for different images.'''

    #Randomize type value between 0-2
    self.__type = random.randrange(3)
```

Functionality: The method sets the __type instance variable to a random integer between 0 and 2, which corresponds to different cloud images.

3. Random Speed (random_speed method)

This method randomizes the movement speed of the cloud.

```
def random_speed(self):
    '''This method randomizes the speed of the cloud.'''

    #Randomize direction of x vector between -1 or 1.
    self.__dx = random.randrange(-1, 2, 2)

    #Randomize values of y vector between 2-4
    self.__dy = random.randrange(2, 5)
```

Functionality: The method sets the horizontal speed (__dx) to either -1 or 1 (indicating left or right movement) and the vertical speed (__dy) to a random value between 2 and 4, making the cloud descend at a varied rate.

4. Random Reset (random_reset method)

This method sets the cloud's initial position at the top of the screen.

```
def random_reset(self):
    '''This method randomizes the position of the cloud near the top of the
    screen.'''

    #x pos is static
    x_pos = (self.__screen.get_width()-200)/2
    #y pos is randomized for varied cloud appearance
    y_pos = random.randrange(-200, -39, 2)

    #Position cloud.
    self.rect.center = (x_pos, y_pos)
```

Functionality: The method sets the x-coordinate of the cloud to a static value (centered horizontally) and randomizes the y-coordinate to a value above the screen, ensuring that clouds appear from above.

5. Update (update method)

The update method is called automatically once per frame to update the cloud's position and reset it if necessary.

```
def update(self):
    '''This method automatically runs once per frame to update the cloud.
    Reset cloud if appropriate.'''

    #Update position using vectors.
    self.rect.centerx += self.__dx
    self.rect.centery += self.__dy

    #Reset cloud if out of bottom.
    if self.rect.top > self.__screen.get_height():
        self.random_type()
        self.random_speed()
        self.random_reset()
```

Functionality:

Updates the cloud's position based on its speed vectors.
Checks if the cloud has exited the screen (i.e., its top edge is below the screen's height).
If the cloud has exited, resets its type, speed, and position using the random_type, random_speed, and random_reset methods, respectively.

Michael Arianno Chandrarieta - 2802499711

Background

1. Initialization (__init__ method)

The constructor initializes the background surface and sets up the necessary properties for scrolling.

```
def __init__(self):
    '''This initializer creates surface, loads background and initializes
    other instances needed for scrolling in update'''

    # Call the parent __init__() method
    pygame.sprite.Sprite.__init__(self)

    # Create surface initialize rect, position.
    self.image = pygame.Surface((440, 480))
    self.rect = self.image.get_rect()
    self.rect.left, self.rect.top = (0,0)

    #Load image and initialize other instances.
    self.__background = pygame.image.load("images/background.png").convert()
    self.__background_y = -440
    self.__dy = 1
```

Key Actions:

Initializes the background surface and its rectangle for positioning.
Loads the background image and sets its initial vertical position off-screen.

2. Getting the Surface (get_surface method)

This method returns the surface of the background.

```
def get_surface(self):
    '''This method returns the surface of the background which is needed for
    the clear method in main.'''

    #Return instance.
    return self.image
```

Functionality: This method provides access to the background surface, which can be used for clearing or rendering purposes in the main game loop.

Michael Arianno Chandrarieta - 2802499711

3. Update (update method)

The update method is called automatically to reposition the background image, creating the scrolling effect.

```
def update(self):
    '''This method will be called automatically to reposition the
    background image on the surface, seemingly scrolling.'''

    #Update image.
    self.__background_y += self.__dy

    #Blit image on to surface.
    self.image.blit(self.__background, (0, self.__background_y))

    #Reset image to origin when appropriate
    if self.__background_y >= 0:
        self.__background_y = -440
```

Functionality:

Updates the vertical position of the background image based on the scrolling speed (__dy).
Blits (draws) the background image onto the surface at its current position.
Resets the position of the background image to create a seamless scrolling effect when it has moved past the top of the screen.