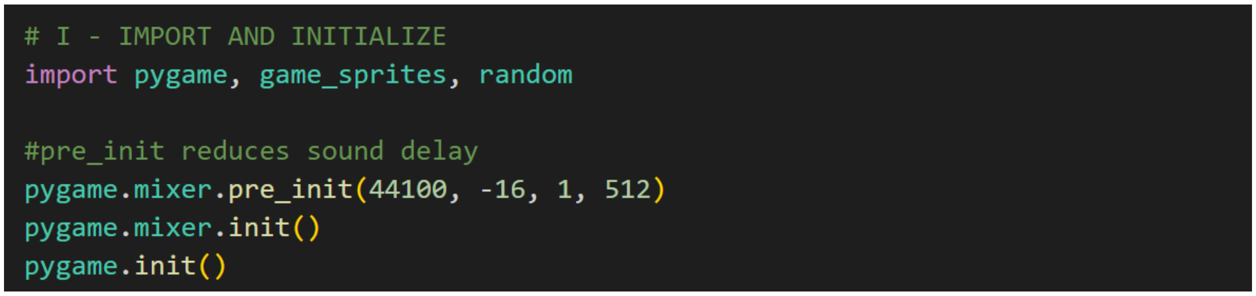
Michael Arianno Chandrarieta - 2802499711

main.py

Imports and initialization

1. Import Statements



```python
# I - IMPORT AND INITIALIZE
import pygame, game_sprites, random

#pre_init reduces sound delay
pygame.mixer.pre_init(44100, -16, 1, 512)
pygame.mixer.init()
pygame.init()
```

Imports the necessary libraries: pygame for game development, game_sprites for custom sprite classes, and random for random number generation.

2. Pygame Mixer Initialization

pygame.mixer.pre_init(44100, -16, 1, 512)

This line initializes the Pygame mixer module with specific parameters before the main initialization. The parameters are as follows:

44100: This is the sample rate in Hertz (Hz)
-16: This specifies the size of each audio sample in bits
1: This indicates the number of channels
512: This is the buffer size

pygame.mixer.init()
This line initializes the Pygame mixer module with the settings specified in the pre_init call. It prepares Pygame to handle sound playback, allowing the game to load and play sound effects and music.

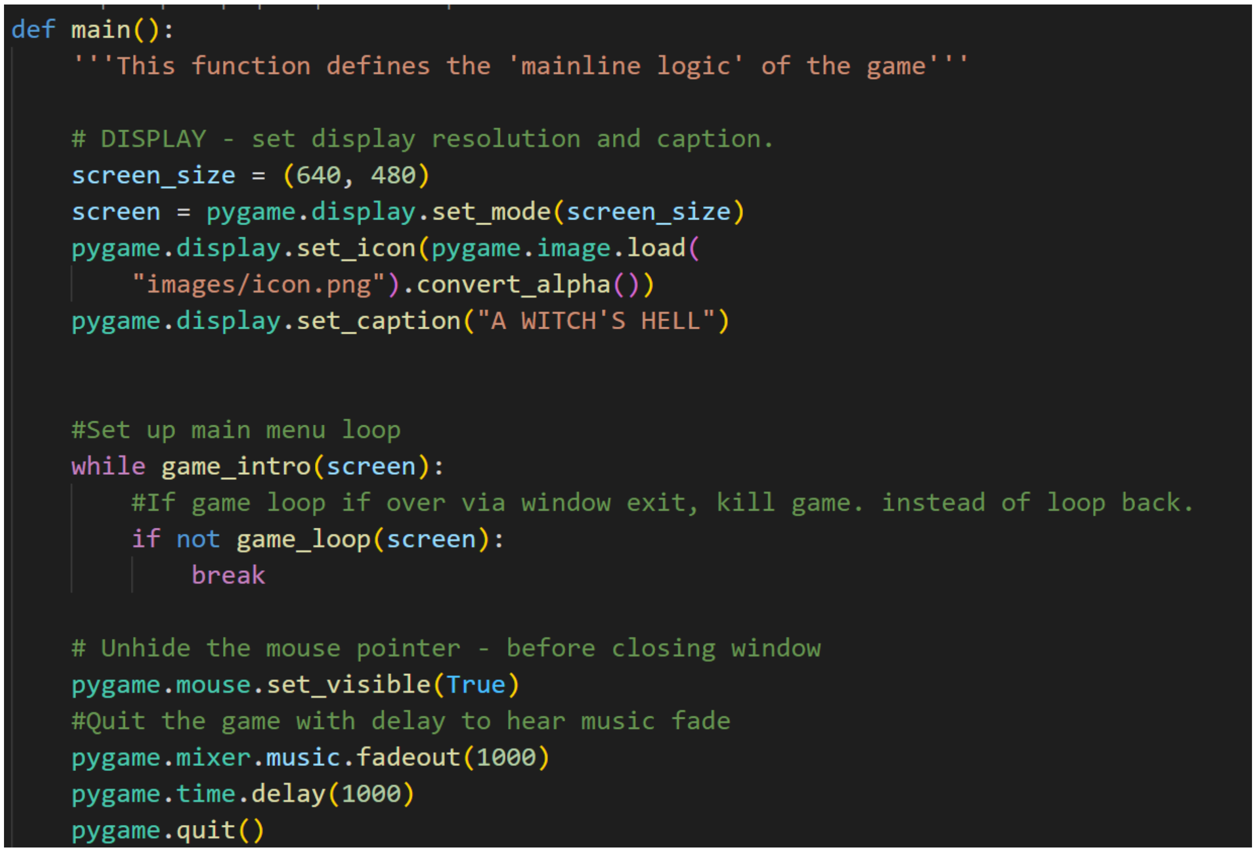
Michael Arianno Chandrarieta - 2802499711

3. Pygame Initialization

pygame.init()
This function initializes all Pygame modules. It sets up the necessary components for Pygame to function, including graphics, sound, and input handling. It is typically called once at the beginning of a Pygame program to prepare the environment for the game to run.

Michael Arianno Chandrarieta - 2802499711

Main function


```python
def main():
    '''This function defines the 'mainline logic' of the game'''

    # DISPLAY - set display resolution and caption.
    screen_size = (640, 480)
    screen = pygame.display.set_mode(screen_size)
    pygame.display.set_icon(pygame.image.load(
        "images/icon.png").convert_alpha())
    pygame.display.set_caption("A WITCH'S HELL")


    #Set up main menu loop
    while game_intro(screen):
        #If game loop if over via window exit, kill game. instead of loop back.
        if not game_loop(screen):
            break

    # Unhide the mouse pointer - before closing window
    pygame.mouse.set_visible(True)
    #Quit the game with delay to hear music fade
    pygame.mixer.music.fadeout(1000)
    pygame.time.delay(1000)
    pygame.quit()
```

1. Function Definition

def main():
This line defines the main function, which serves as the entry point for the game. It contains the core logic that orchestrates the flow of the game from start to finish.

2. Display Setup

screen_size = (640, 480)
This line defines the size of the game window. The screen_size variable is a tuple representing the width (640 pixels) and height (480 pixels) of the window.

screen = pygame.display.set_mode(screen_size)
This line creates the game window using the specified dimensions. The set_mode function initializes a window or screen for display, returning a surface object (in this case, screen) that can be used for drawing.

pygame.display.set_icon(pygame.image.load("images/icon.png").convert_alpha())

This line sets the window's icon. It loads an image file (icon.png) from the images directory and converts it to a format suitable for Pygame. The convert_alpha() method ensures that the image retains its transparency information.

pygame.display.set_caption("A WITCH'S HELL"):
This line sets the title of the game window to "A WITCH'S HELL". This title appears in the title bar of the window.

3. Main Menu Loop

while game_intro(screen):
This line initiates a loop that runs the game_intro function, which displays the main menu of the game. The loop continues until the game_intro function returns a value indicating that the player has selected to start the game or exit.

if not game_loop(screen):
After the main menu, the game_loop function is called to start the actual gameplay. If game_loop returns a value that indicates the game is over (for example, if the player has lost all lives or exited), the loop breaks, and the program proceeds to the cleanup section.

4. Cleanup and Exit

pygame.mouse.set_visible(True)
This line unhides the mouse pointer before the game window is closed. By default, the mouse pointer may be hidden during gameplay to provide a more immersive experience.

pygame.mixer.music.fadeout(1000)
This line gradually fades out any currently playing background music over a period of 1000 milliseconds (1 second). This creates a smoother transition when quitting the game.

pygame.time.delay(1000)
This line introduces a delay of 1000 milliseconds (1 second) after fading out the music. It allows players to hear the music fade before the game window closes.

pygame.quit()
This line cleanly exits Pygame, releasing any resources that were allocated during the game. It should always be called before the program ends to ensure that Pygame shuts down properly.

Michael Arianno Chandrarieta - 2802499711

Pause Function

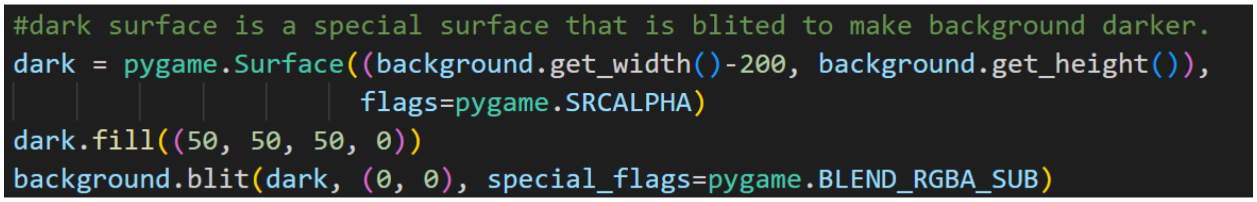
1. Function Definition

def pause(screen):
This line defines the pause function, which is responsible for displaying the pause menu when the game is paused. It accepts a screen parameter, which is the display surface where the pause menu will be drawn.

2. Background Setup

background = screen
This line sets the background variable to the current screen surface, allowing the function to draw over the existing game state.

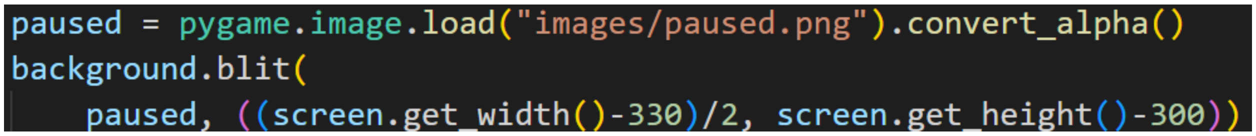
Creating a Dark Overlay:



```
#dark surface is a special surface that is blited to make background darker.
dark = pygame.Surface((background.get_width()-200, background.get_height()),
                      flags=pygame.SRCALPHA)
dark.fill((50, 50, 50, 0))
background.blit(dark, (0, 0), special_flags=pygame.BLEND_RGBA_SUB)
```

- A new surface dark is created that is slightly narrower than the screen width. This surface will be used to create a darker overlay effect.
- The fill method fills this surface with a color (50, 50, 50) which is a dark gray with full transparency (the last value is 0, indicating no opacity).
- The blit method draws this dark surface onto the background, using the BLEND_RGBA_SUB flag to blend the dark overlay with the existing screen, creating a dimmed effect.

Loading and Drawing the "Paused" Image:



```
paused = pygame.image.load("images/paused.png").convert_alpha()
background.blit(
    paused, ((screen.get_width()-330)/2, screen.get_height()-300))
```

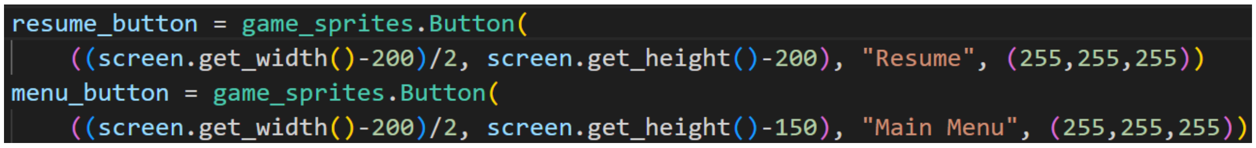
The "paused" image is loaded from the images directory and converted to a format suitable for Pygame.

The blit method draws this image onto the background at a calculated position, centered horizontally and positioned near the bottom of the screen.

Michael Arianno Chandrarieta - 2802499711

3. Button Creation and Sprite Group Setup

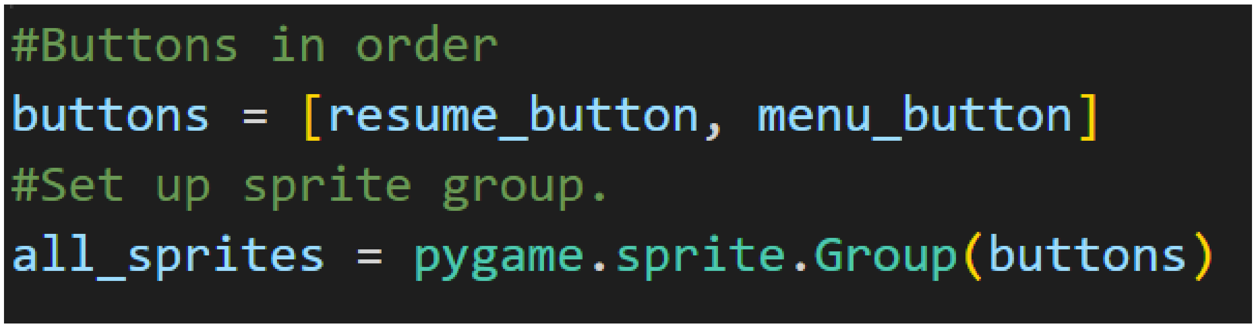
Creating Buttons:



```
resume_button = game_sprites.Button(
    ((screen.get_width()-200)/2, screen.get_height()-200), "Resume", (255,255,255))
menu_button = game_sprites.Button(
    ((screen.get_width()-200)/2, screen.get_height()-150), "Main Menu", (255,255,255))
```

Two buttons are created using the Button class from game_sprites: a "Resume" button and a "Main Menu" button. Each button is positioned at a calculated location, centered horizontally and spaced vertically.

Button List and Sprite Group:

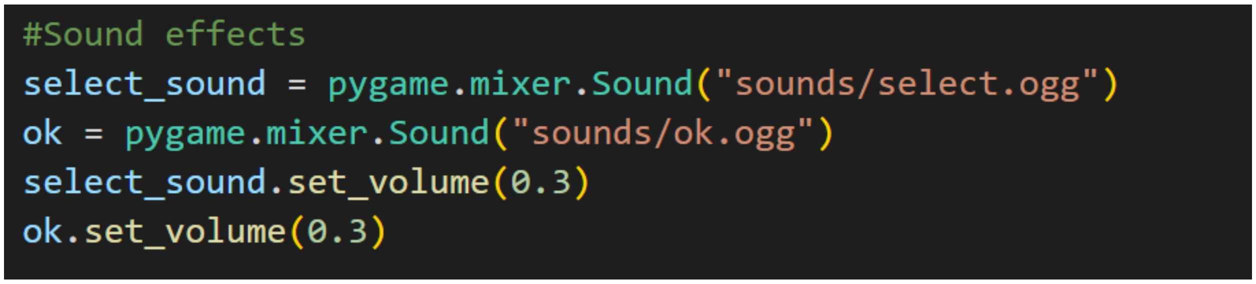


```
#Buttons in order
buttons = [resume_button, menu_button]
#Set up sprite group.
all_sprites = pygame.sprite.Group(buttons)
```

A list buttons is created to hold the two buttons
A sprite group all_sprites is created to manage the buttons as a group.

4. Sound Effects and Initializations

Sound Effects:



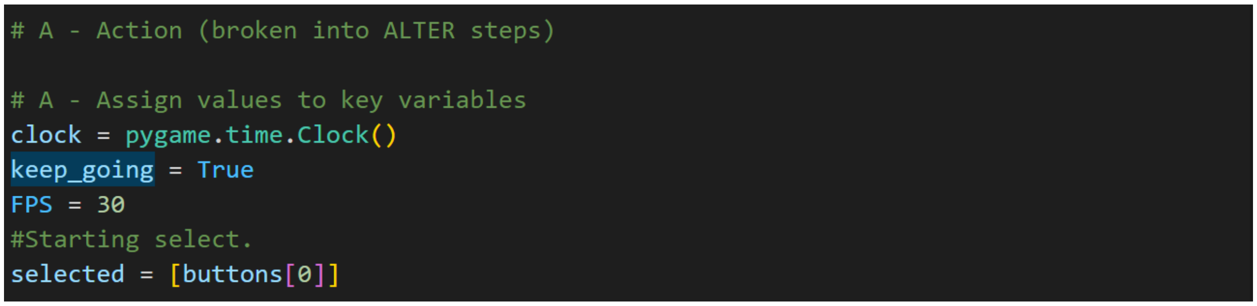
```
#Sound effects
select_sound = pygame.mixer.Sound("sounds/select.ogg")
ok = pygame.mixer.Sound("sounds/ok.ogg")
select_sound.set_volume(0.3)
ok.set_volume(0.3)
```

Two sound effects are loaded: select_sound for navigating through buttons and ok for confirming a button press.

The volume of both sounds is set to 0.3.

Michael Arianno Chandrarieta - 2802499711

Initializations:

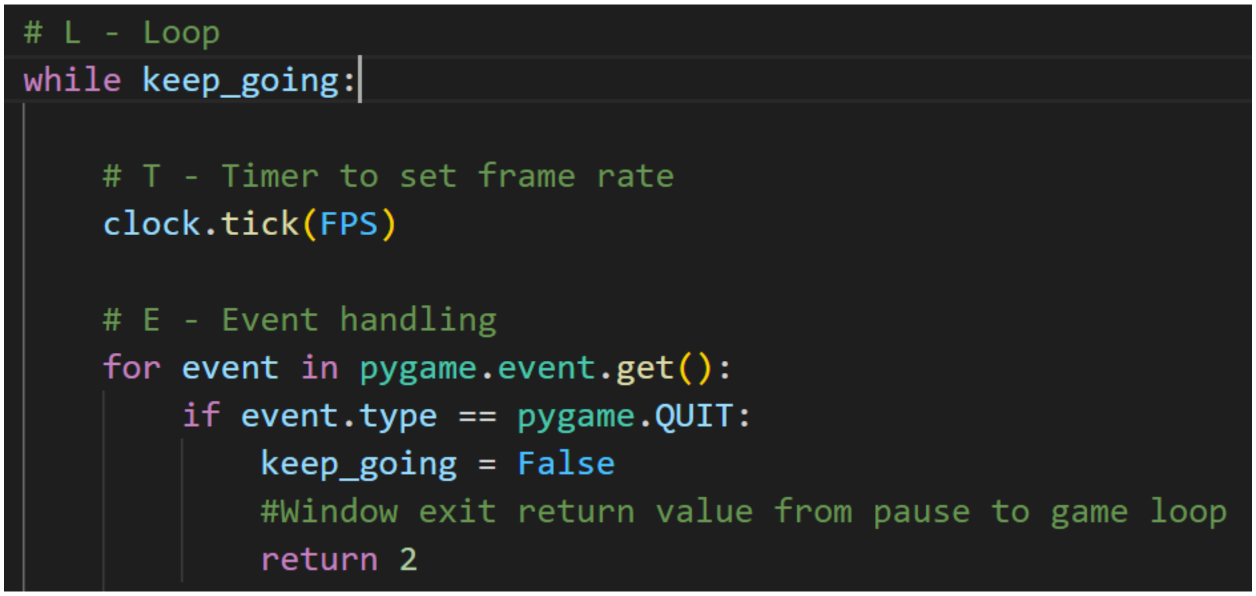


```
# A - Action (broken into ALTER steps)

# A - Assign values to key variables
clock = pygame.time.Clock()
keep_going = True
FPS = 30
#Starting select.
selected = [buttons[0]]
```

A clock object clock is created to control the frame rate.
The keep_going variable is set to True to indicate that the pause loop should continue.
The frame rate FPS is set to 30.
The selected variable is initialized to the first button (resume_button) to highlight it initially.

5. Pause Loop

Timer and Event Handling:

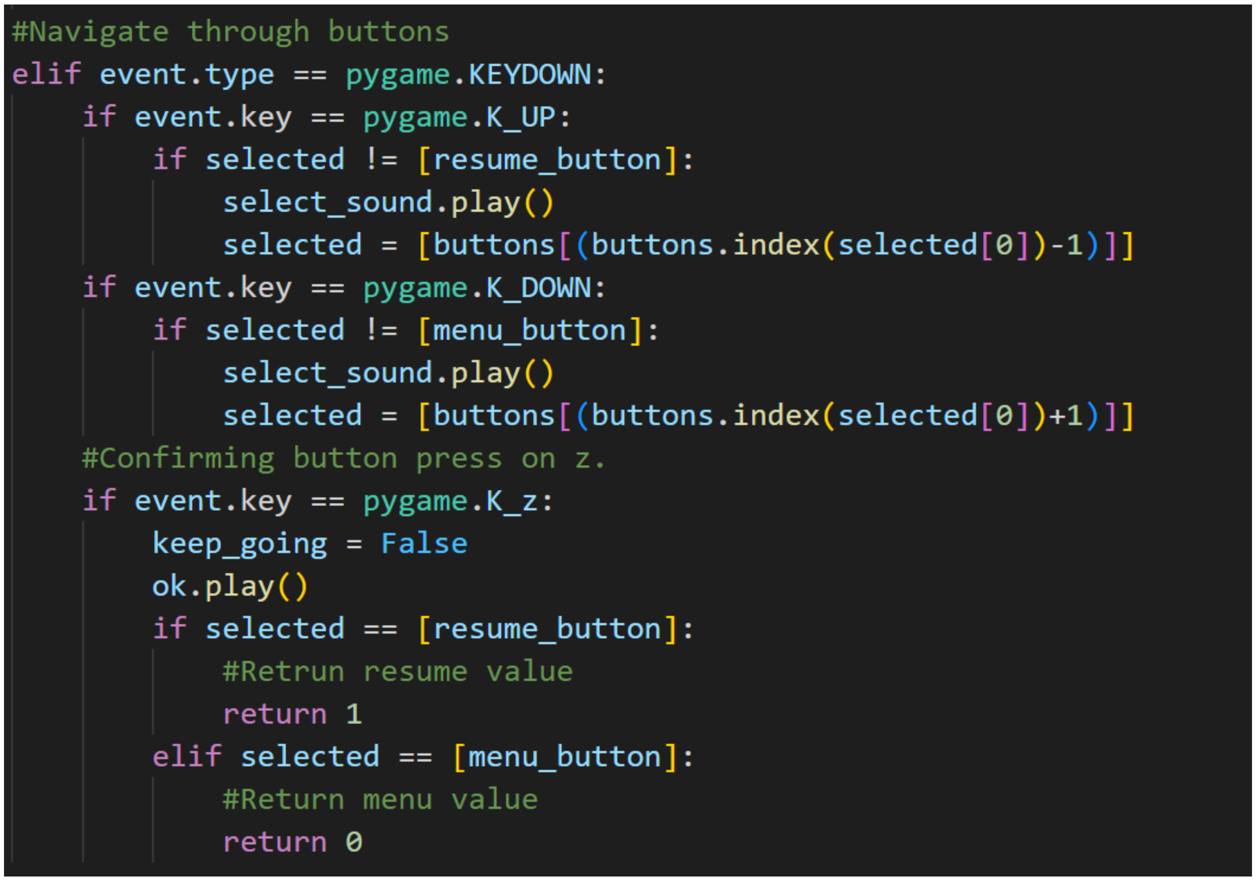


```
# L - Loop
while keep_going:

    # T - Timer to set frame rate
    clock.tick(FPS)

    # E - Event handling
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            keep_going = False
            #Window exit return value from pause to game loop
            return 2
```

The pause loop runs until keep_going is set to False.
The clock.tick (FPS) line limits the frame rate to FPS (30 in this case).
The event handling loop processes events such as key presses, mouse movements, and window close events.

Button Navigation and Confirmation:

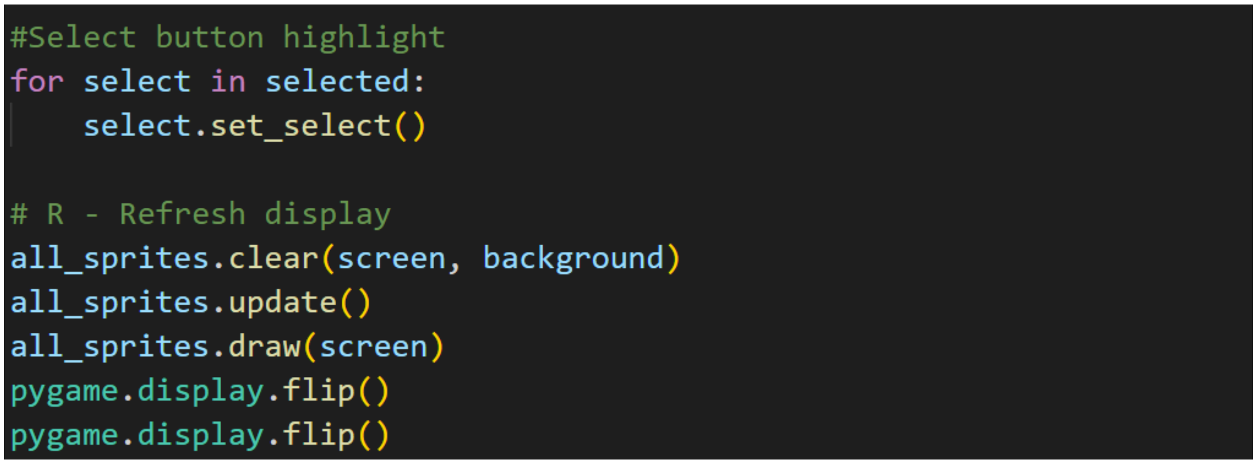
Michael Arianno Chandrarieta - 2802499711



```python
#Navigate through buttons
elif event.type == pygame.KEYDOWN:
    if event.key == pygame.K_UP:
        if selected != [resume_button]:
            select_sound.play()
            selected = [buttons[(buttons.index(selected[0])-1)]]
    if event.key == pygame.K_DOWN:
        if selected != [menu_button]:
            select_sound.play()
            selected = [buttons[(buttons.index(selected[0])+1)]]
    #Confirming button press on z.
    if event.key == pygame.K_z:
        keep_going = False
        ok.play()
        if selected == [resume_button]:
            #Retrun resume value
            return 1
        elif selected == [menu_button]:
            #Return menu value
            return 0
```

The KEYDOWN event is handled to navigate through buttons using the up and down arrow keys. When the Enter key is pressed, the selected button is confirmed, and the pause loop exits.

Button Highlighting and Refresh:



```python
#Select button highlight
for select in selected:
    select.set_select()

# R - Refresh display
all_sprites.clear(screen, background)
all_sprites.update()
all_sprites.draw(screen)
pygame.display.flip()
pygame.display.flip()
```

The selected button is highlighted by calling its set_select method.

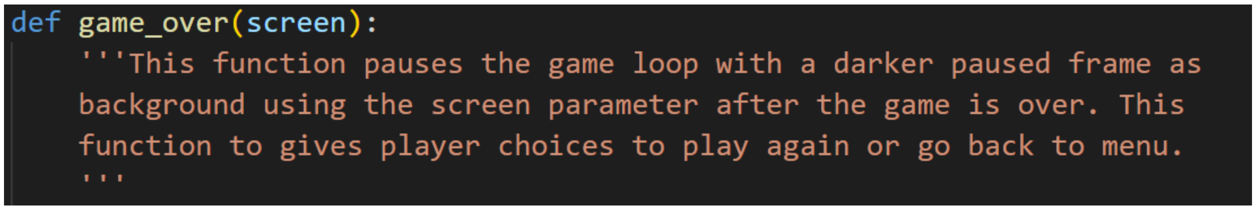
Michael Arianno Chandrarieta - 2802499711

The sprite group all_sprites is cleared, updated, and drawn onto the screen.
The pygame.display.flip method updates the entire display window to show the updated pause menu.

The Pause Function is responsible for displaying the pause menu, handling button navigation and confirmation, and returning control to the game loop when the player resumes or exits to the main menu.

Michael Arianno Chandrarieta - 2802499711

Game Over Function



```python
def game_over(screen):
    '''This function pauses the game loop with a darker paused frame as
    background using the screen parameter after the game is over. This
    function to gives player choices to play again or go back to menu.
    '''
```

1. Function Definition

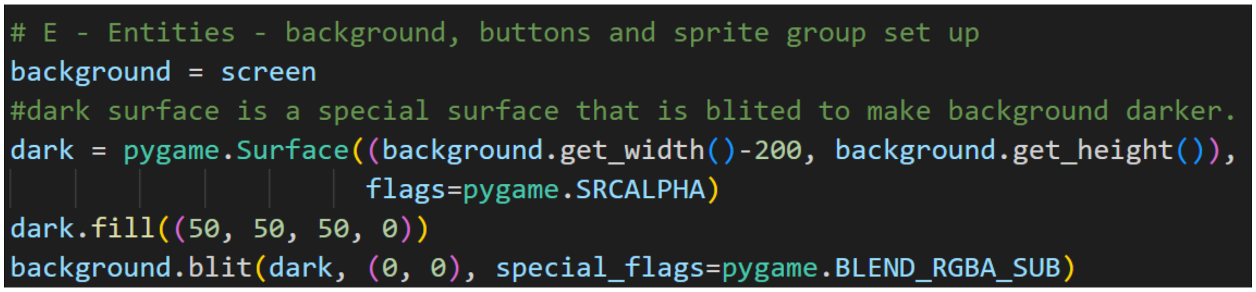
def game_over(screen)
This line defines the game_over function, which is called when the player loses all their lives or the game ends. It presents the player with options to either restart the game or return to the main menu. The function takes a screen parameter, which is the display surface where the game over screen will be rendered.

2. Background Setup

background = screen

This line assigns the current screen to the background variable, allowing the function to draw over the existing game state.

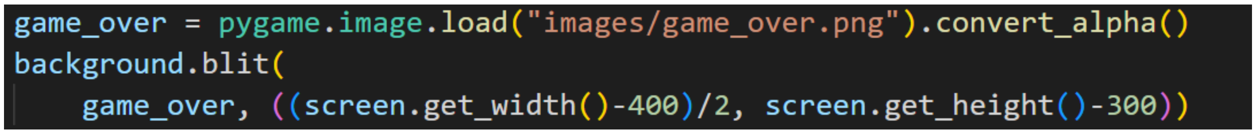
Creating a Dark Overlay:



```python
# E - Entities - background, buttons and sprite group set up
background = screen
#dark surface is a special surface that is blited to make background darker.
dark = pygame.Surface((background.get_width()-200, background.get_height()),
                      flags=pygame.SRCALPHA)
dark.fill((50, 50, 50, 0))
background.blit(dark, (0, 0), special_flags=pygame.BLEND_RGBA_SUB)
```

A new surface dark is created to cover the screen, slightly narrower than the screen width. The fill method fills this surface with a dark gray color, and it is drawn onto the background to create a dimmed effect

Loading and Drawing the "Game Over" Image:



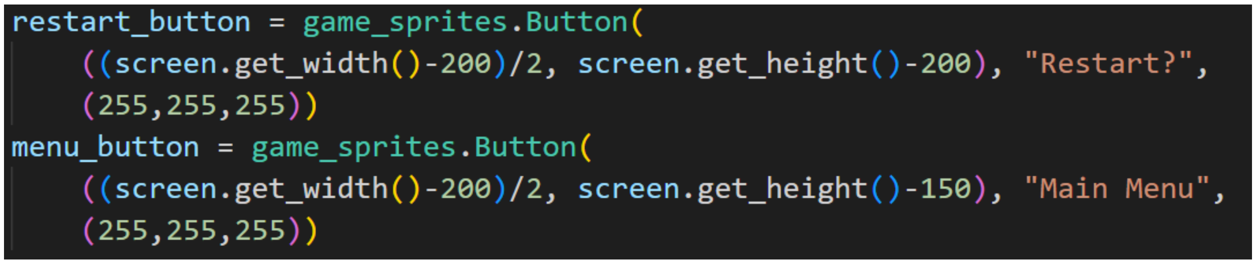
```python
game_over = pygame.image.load("images/game_over.png").convert_alpha()
background.blit(
    game_over, ((screen.get_width()-400)/2, screen.get_height()-300))
```

The "game over" image is loaded from the images directory and converted to a format suitable for Pygame.

The blit method draws this image onto the background at a calculated position, centered horizontally and positioned near the bottom of the screen.

3. Button Creation and Sprite Group Setup

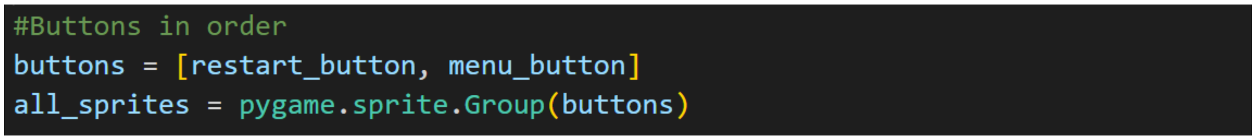
Creating Buttons:



```
restart_button = game_sprites.Button(
    ((screen.get_width()-200)/2, screen.get_height()-200), "Restart?",
    (255,255,255))
menu_button = game_sprites.Button(
    ((screen.get_width()-200)/2, screen.get_height()-150), "Main Menu",
    (255,255,255))
```

Two buttons are created using the Button class from game_sprites: a "Restart?" button and a "Main Menu" button. Each button is positioned at a calculated location, centered horizontally and spaced vertically.

Button List and Sprite Group:

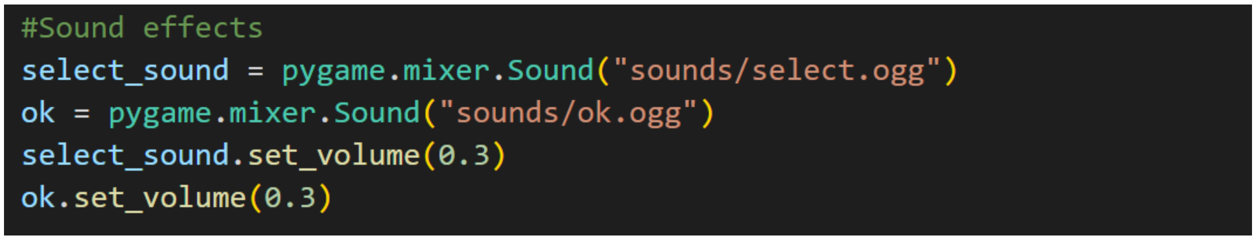


```
#Buttons in order
buttons = [restart_button, menu_button]
all_sprites = pygame.sprite.Group(buttons)
```

A list buttons is created to hold the two buttons.
A sprite group all_sprites is created to manage the buttons as a group.

4. Sound Effects and Initializations

Sound Effects:

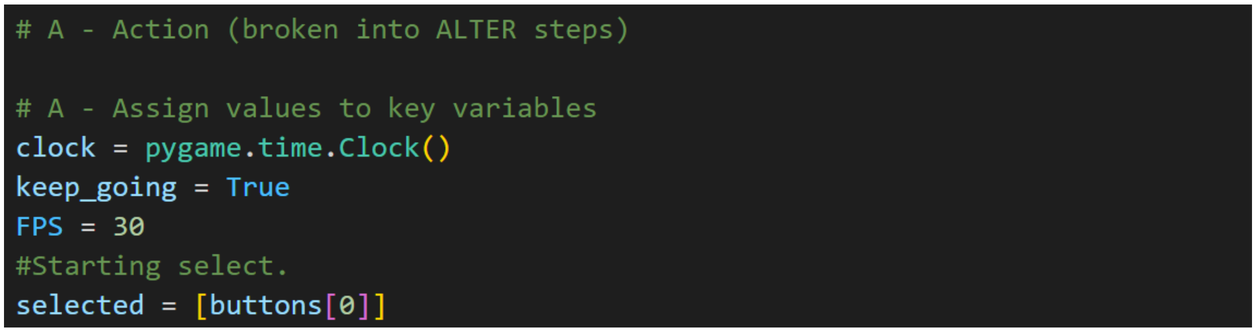


```
#Sound effects
select_sound = pygame.mixer.Sound("sounds/select.ogg")
ok = pygame.mixer.Sound("sounds/ok.ogg")
select_sound.set_volume(0.3)
ok.set_volume(0.3)
```

Michael Arianno Chandrarieta - 2802499711

Two sound effects are loaded: select_sound for navigating through buttons and ok for confirming a button press.
The volume of both sounds is set to 0.3.

Initializations:

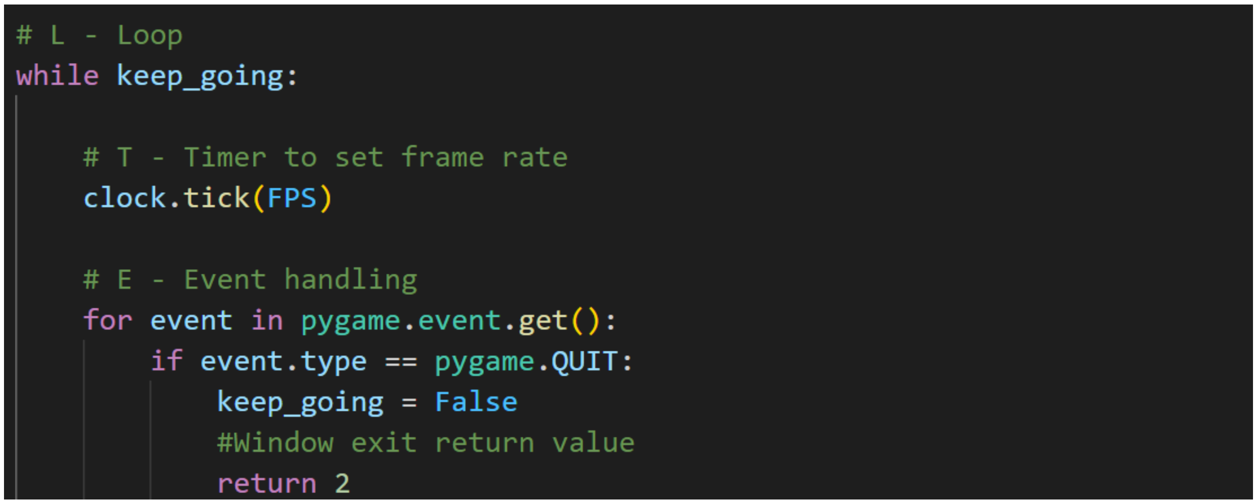


```
# A - Action (broken into ALTER steps)

# A - Assign values to key variables
clock = pygame.time.Clock()
keep_going = True
FPS = 30
#Starting select.
selected = [buttons[0]]
```

A clock object clock is created to control the frame rate.
The keep_going variable is set to True to indicate that the game over loop should continue.
The frame rate FPS is set to 30.
The selected variable is initialized to the first button (restart_button) to highlight it initially.

5. Game Over Loop

Timer and Event Handling:

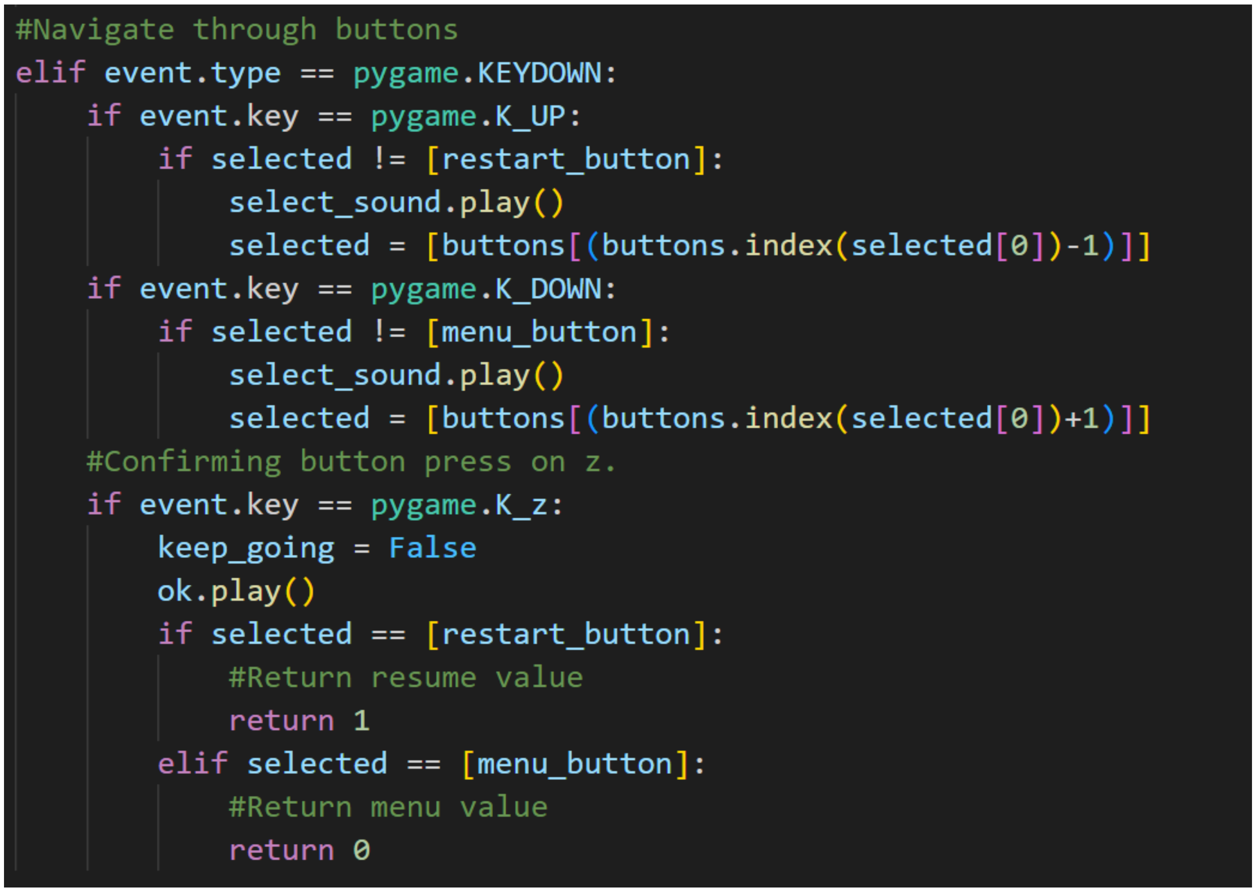


```
# L - Loop
while keep_going:

    # T - Timer to set frame rate
    clock.tick(FPS)

    # E - Event handling
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            keep_going = False
            #Window exit return value
            return 2
```

The game over loop runs until keep_going is set to False.
The clock.tick(FPS) line limits the frame rate to FPS (30 in this case).
The event handling loop processes events such as key presses, mouse movements, and window close events.

Michael Arianno Chandrarieta - 2802499711

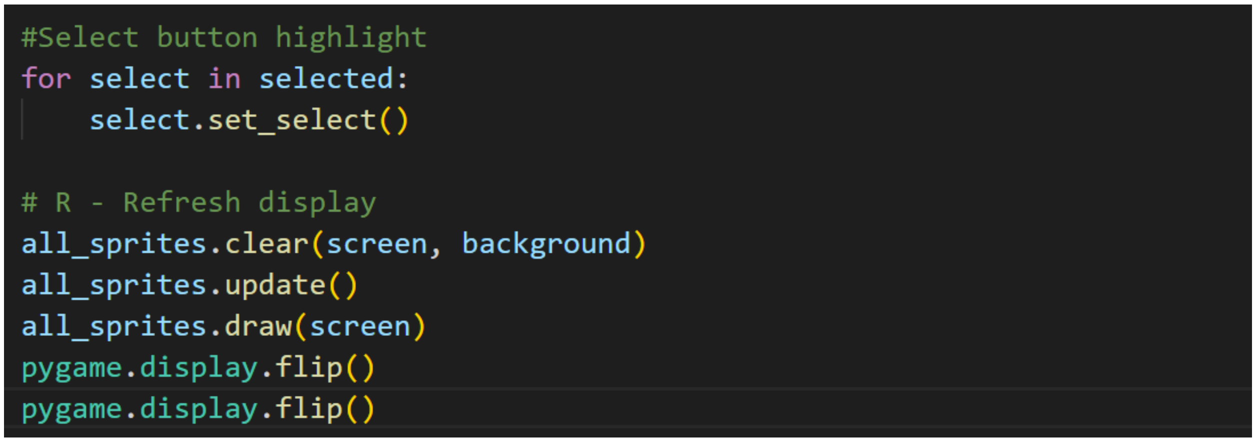
Button Navigation and Confirmation:



```
#Navigate through buttons
elif event.type == pygame.KEYDOWN:
    if event.key == pygame.K_UP:
        if selected != [restart_button]:
            select_sound.play()
            selected = [buttons[(buttons.index(selected[0])-1)]]
    if event.key == pygame.K_DOWN:
        if selected != [menu_button]:
            select_sound.play()
            selected = [buttons[(buttons.index(selected[0])+1)]]
    #Confirming button press on z.
    if event.key == pygame.K_z:
        keep_going = False
        ok.play()
        if selected == [restart_button]:
            #Return resume value
            return 1
        elif selected == [menu_button]:
            #Return menu value
            return 0
```

The KEYDOWN event is handled to navigate through buttons using the up and down arrow keys.
When the Enter key is pressed, the selected button is confirmed, and the game over loop exits.

Button Highlighting and Refresh:



```
#Select button highlight
for select in selected:
    select.set_select()

# R - Refresh display
all_sprites.clear(screen, background)
all_sprites.update()
all_sprites.draw(screen)
pygame.display.flip()
pygame.display.flip()
```

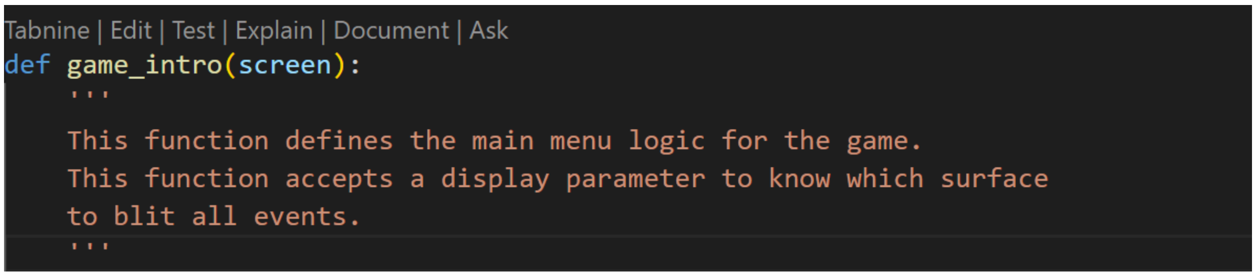
Michael Arianno Chandrarieta - 2802499711

The selected button is highlighted by calling its set_select method.
The sprite group all_sprites is cleared, updated, and drawn onto the screen.
The pygame.display.flip method updates the entire display window to show the updated game over screen.

The Game Over Function is responsible for displaying the game over screen, handling button navigation and confirmation, and returning control to the game loop when the player chooses to restart or exit to the main menu.

Michael Arianno Chandrarieta - 2802499711

Game Intro Function



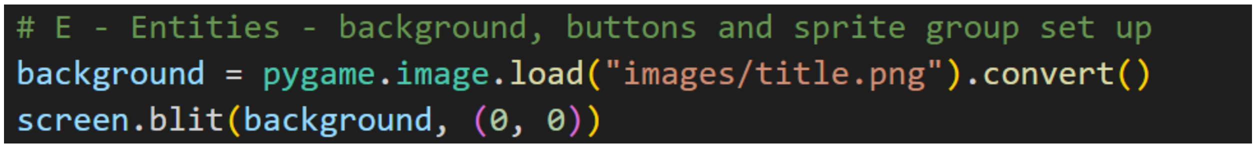
```
Tabnine | Edit | Test | Explain | Document | Ask
def game_intro(screen):
    '''
    This function defines the main menu logic for the game.
    This function accepts a display parameter to know which surface
    to blit all events.
    '''
```

1. Function Definition

def game_intro(screen)
This line defines the game_intro function, which is responsible for displaying the main menu of the game. It accepts a screen parameter, which is the display surface where the menu will be rendered.

2. Background Setup

Loading Background Image:

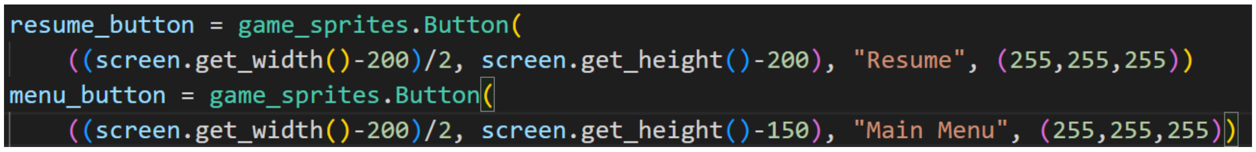


```
# E - Entities - background, buttons and sprite group set up
background = pygame.image.load("images/title.png").convert()
screen.blit(background, (0, 0))
```

The background image for the title screen is loaded from the images directory and converted to a format suitable for Pygame.
The blit method draws this image onto the screen, filling the background.

3. Button Creation and Sprite Group Setup

Creating Buttons:

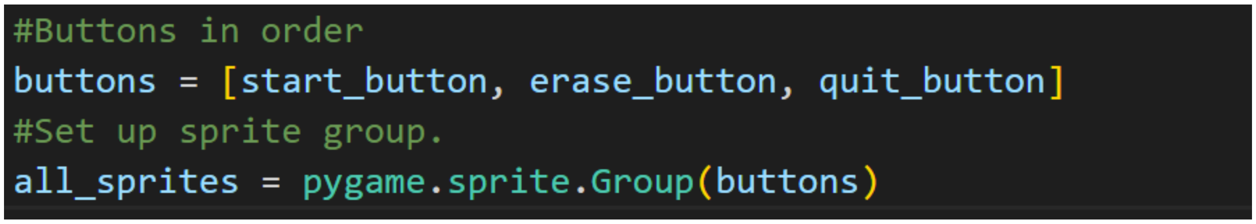


```
resume_button = game_sprites.Button(
    ((screen.get_width()-200)/2, screen.get_height()-200), "Resume", (255,255,255))
menu_button = game_sprites.Button(
    ((screen.get_width()-200)/2, screen.get_height()-150), "Main Menu", (255,255,255))
```

Three buttons are created using the Button class from game_sprites: a "Start" button, an "Erase Data" button, and a "Quit" button. Each button is positioned at a calculated location, aligned to the right edge of the screen.

Michael Arianno Chandrarieta - 2802499711

Button List and Sprite Group:

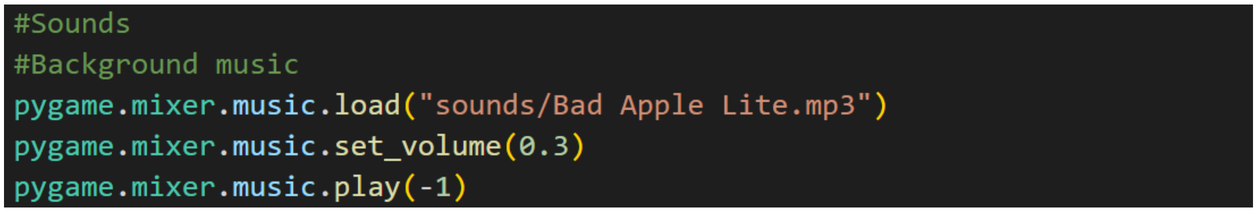


```
#Buttons in order
buttons = [start_button, erase_button, quit_button]
#Set up sprite group.
all_sprites = pygame.sprite.Group(buttons)
```

A list buttons is created to hold the three buttons.
A sprite group all_sprites is created to manage the buttons as a group.

4. Sound Effects and Initializations

Background Music:

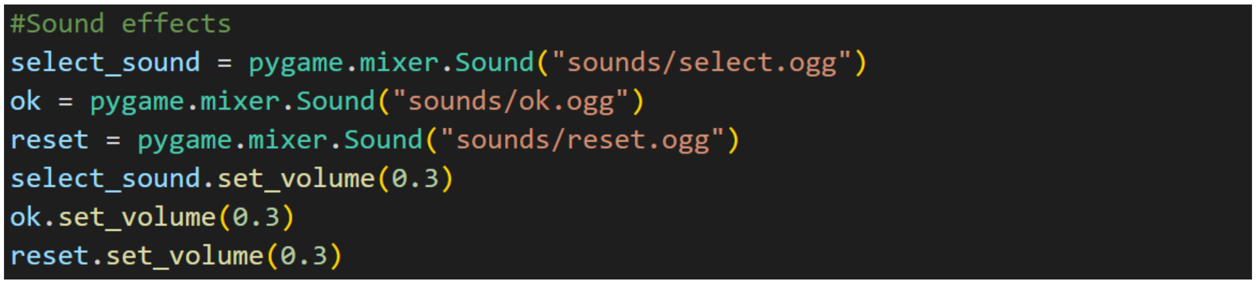


```
#Sounds
#Background music
pygame.mixer.music.load("sounds/Bad Apple Lite.mp3")
pygame.mixer.music.set_volume(0.3)
pygame.mixer.music.play(-1)
```

(BAD APPLE MENTIONED!!!)

The background music for the title screen is loaded from the sounds directory and set to play indefinitely with a volume of 0.3.

Sound Effects:

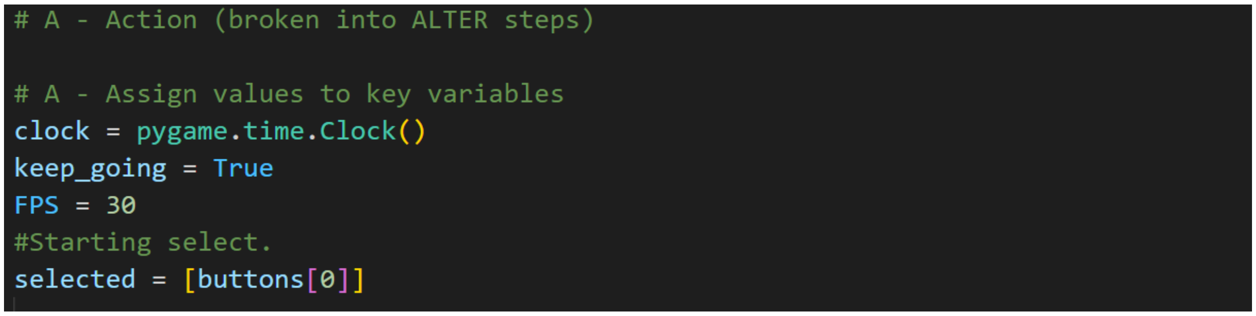


```
#Sound effects
select_sound = pygame.mixer.Sound("sounds/select.ogg")
ok = pygame.mixer.Sound("sounds/ok.ogg")
reset = pygame.mixer.Sound("sounds/reset.ogg")
select_sound.set_volume(0.3)
ok.set_volume(0.3)
reset.set_volume(0.3)
```

Three sound effects are loaded: select_sound for navigating through buttons, ok for confirming a button press, and reset for resetting the high score.
The volume of each sound effect is set to 0.3.

Initializations:

Michael Arianno Chandrarieta - 2802499711

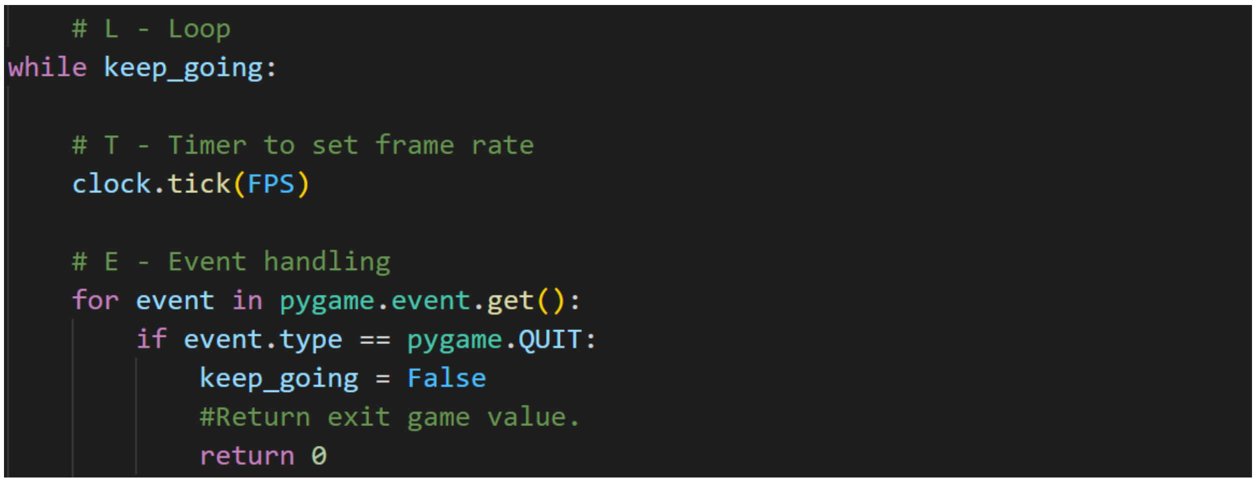


```
# A - Action (broken into ALTER steps)

# A - Assign values to key variables
clock = pygame.time.Clock()
keep_going = True
FPS = 30
#Starting select.
selected = [buttons[0]]
```

A clock object clock is created to control the frame rate.
The keep_going variable is set to True to indicate that the title screen loop should continue.
The frame rate FPS is set to 30.
The selected variable is initialized to the first button (start_button) to highlight it initially.

5. Title Screen Loop

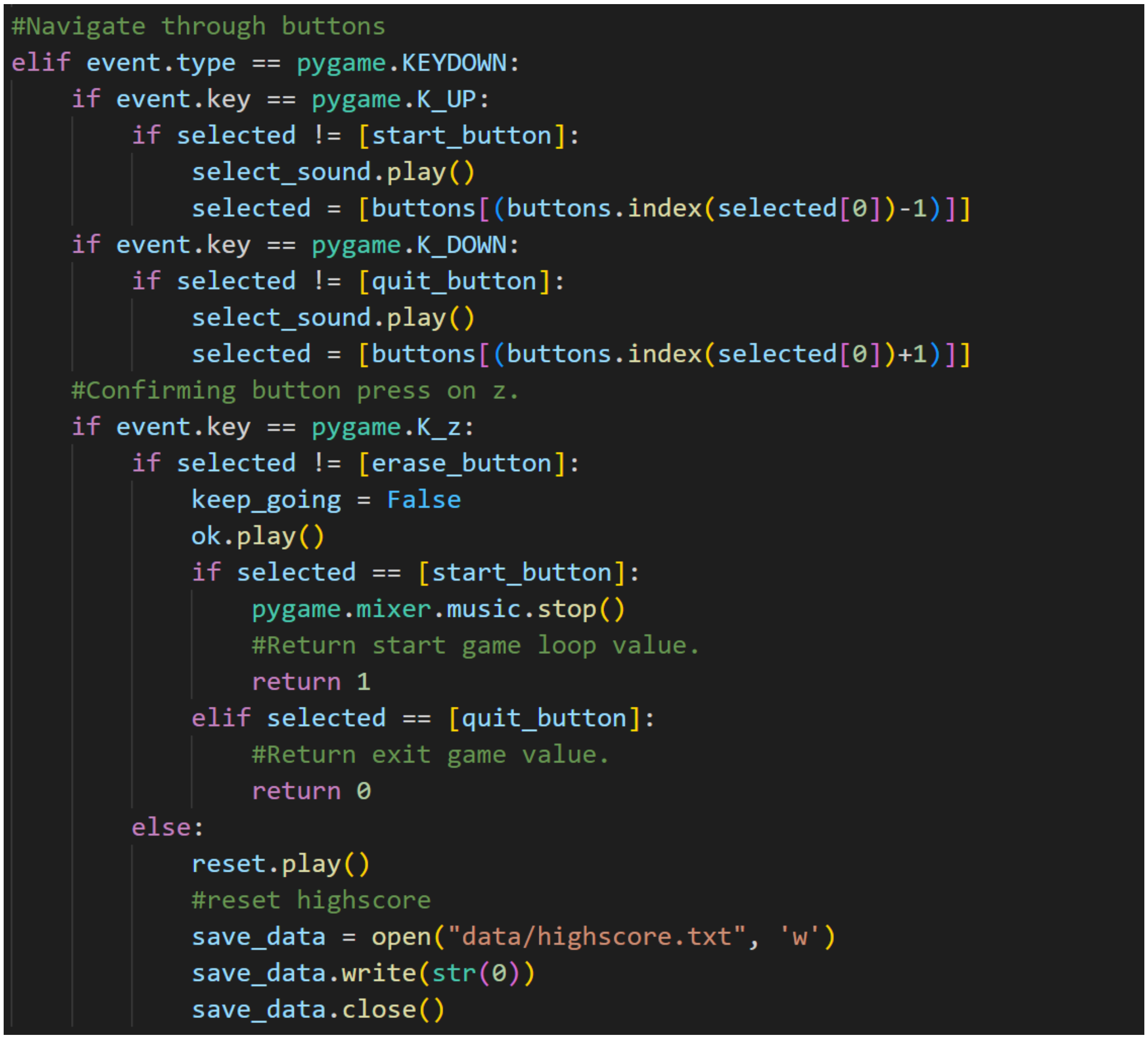
Timer and Event Handling:



```
    # L - Loop
while keep_going:

    # T - Timer to set frame rate
    clock.tick(FPS)

    # E - Event handling
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            keep_going = False
            #Return exit game value.
            return 0
```

The title screen loop runs until keep_going is set to False.
The clock.tick(FPS) line limits the frame rate to FPS (30 in this case).
The event handling loop processes events such as key presses, mouse movements, and window close events.

Button Navigation and Confirmation:

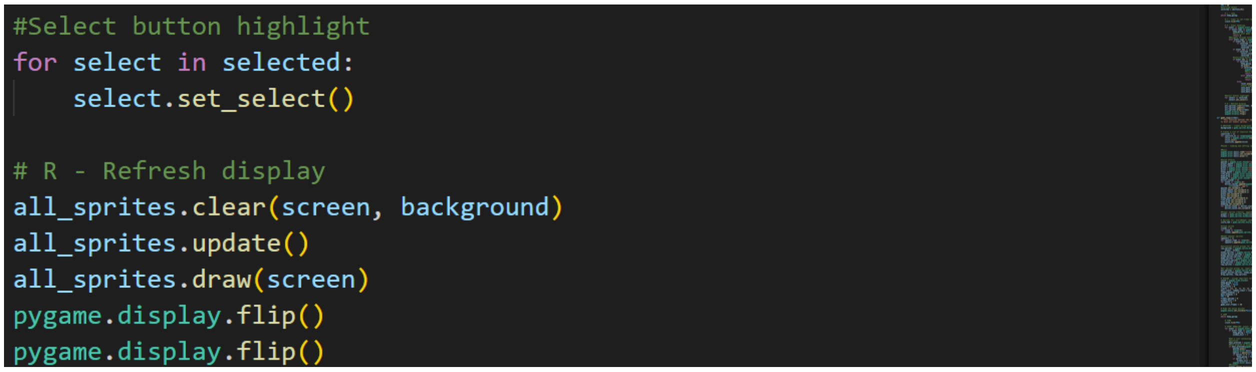


```python
#Navigate through buttons
elif event.type == pygame.KEYDOWN:
    if event.key == pygame.K_UP:
        if selected != [start_button]:
            select_sound.play()
            selected = [buttons[(buttons.index(selected[0])-1)]]
    if event.key == pygame.K_DOWN:
        if selected != [quit_button]:
            select_sound.play()
            selected = [buttons[(buttons.index(selected[0])+1)]]
    #Confirming button press on z.
    if event.key == pygame.K_z:
        if selected != [erase_button]:
            keep_going = False
            ok.play()
            if selected == [start_button]:
                pygame.mixer.music.stop()
                #Return start game loop value.
                return 1
            elif selected == [quit_button]:
                #Return exit game value.
                return 0
        else:
            reset.play()
            #reset highscore
            save_data = open("data/highscore.txt", 'w')
            save_data.write(str(0))
            save_data.close()
```

The KEYDOWN event is handled to navigate through buttons using the up and down arrow keys.
When the Enter key is pressed, the selected button is confirmed, and the title screen loop exits.

Button Highlighting and Refresh:

Michael Arianno Chandrarieta - 2802499711



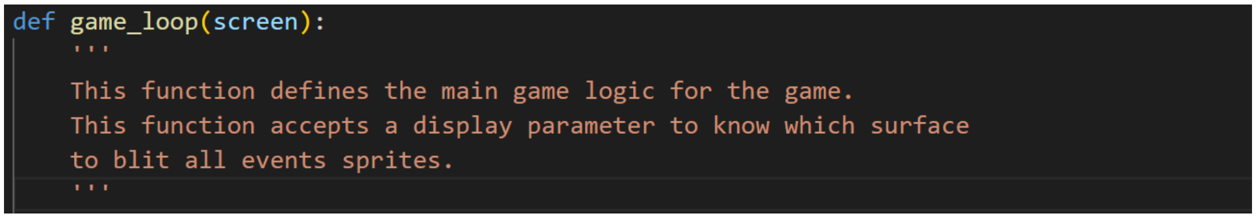
```
#Select button highlight
for select in selected:
    select.set_select()

# R - Refresh display
all_sprites.clear(screen, background)
all_sprites.update()
all_sprites.draw(screen)
pygame.display.flip()
pygame.display.flip()
```

The selected button is highlighted by calling its set_select method.
The sprite group all_sprites is cleared, updated, and drawn onto the screen.
The pygame.display.flip method updates the entire display window to show the updated title screen.

The Game Intro Function is responsible for displaying the title screen, handling button navigation and confirmation, and returning control to the game loop when the player chooses to start the game, erase data, or quit.

Michael Arianno Chandrarieta - 2802499711

Game Loop Function



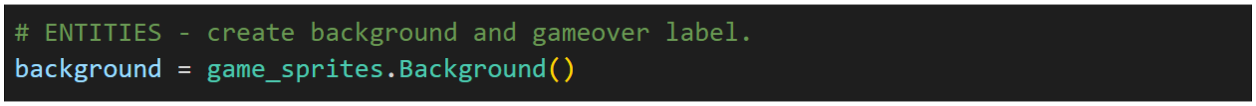
```
def game_loop(screen):
    '''
    This function defines the main game logic for the game.
    This function accepts a display parameter to know which surface
    to blit all events sprites.
    '''
```

1. Function Definition

def game_loop(screen):
This line defines the game_loop function, which is the core of the game. It manages the main game mechanics and renders the game state on the provided screen surface.

2. Background and Joystick Setup

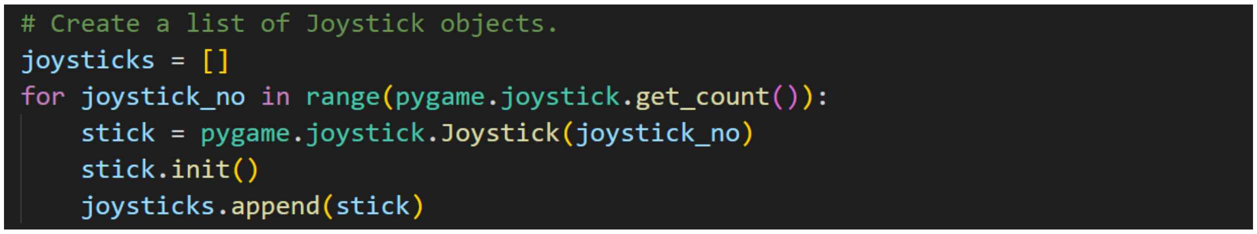
Creating the Background:



```
# ENTITIES - create background and gameover label.
background = game_sprites.Background()
```

An instance of the Background class is created, which likely handles the game's background graphics.

Joystick Initialization:



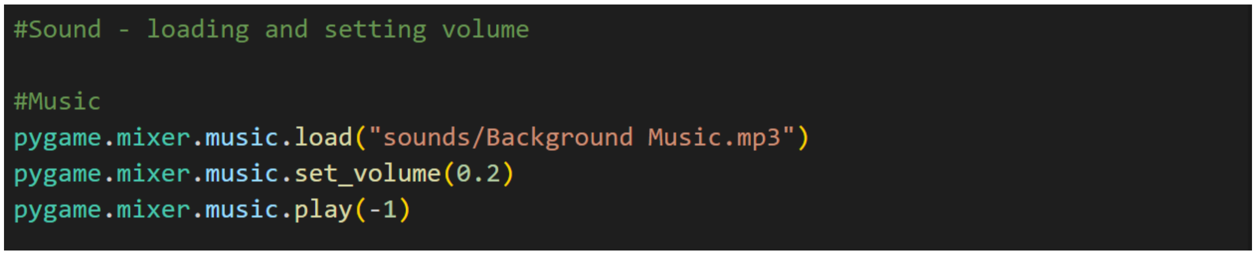
```
# Create a list of Joystick objects.
joysticks = []
for joystick_no in range(pygame.joystick.get_count()):
    stick = pygame.joystick.Joystick(joystick_no)
    stick.init()
    joysticks.append(stick)
```

This section initializes any connected joystick devices. It creates a list of joystick objects and initializes each one.

3. Sound Setup

Loading Music and Sound Effects:

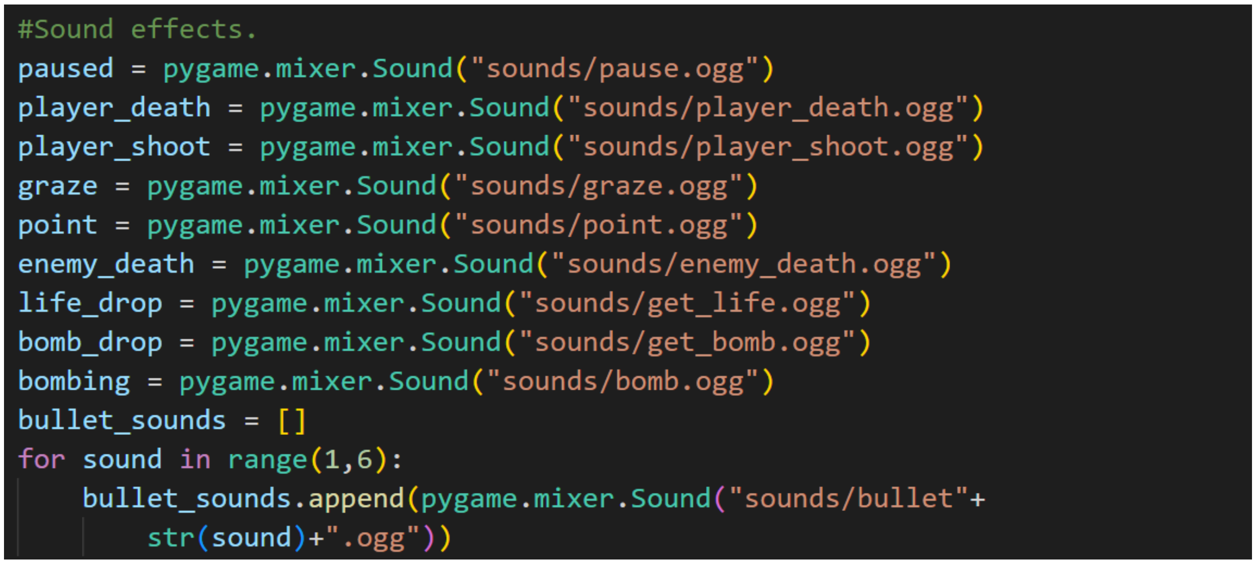
Michael Arianno Chandrarieta - 2802499711



```
#Sound - loading and setting volume

#Music
pygame.mixer.music.load("sounds/Background Music.mp3")
pygame.mixer.music.set_volume(0.2)
pygame.mixer.music.play(-1)
```

Background music is loaded and set to play indefinitely with a volume of 0.2.

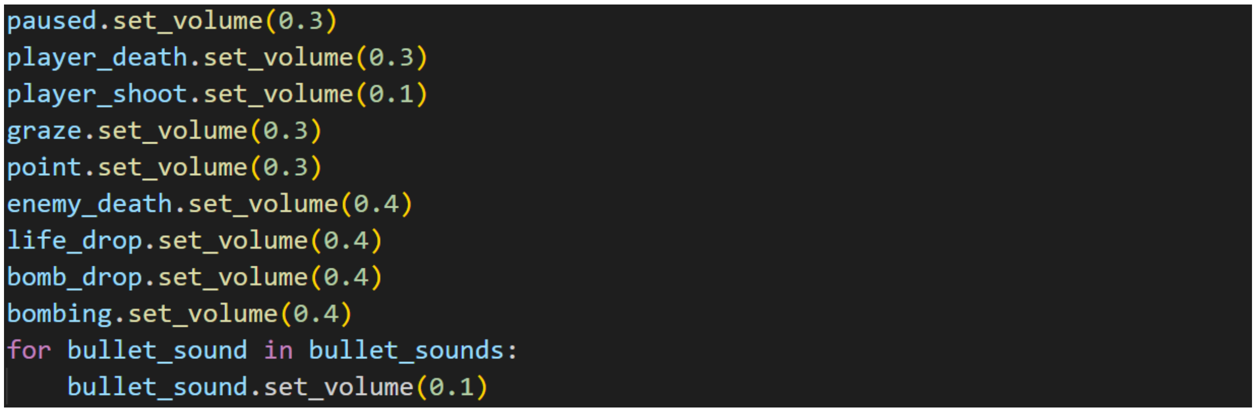
Loading Various Sound Effects:



```
#Sound effects.
paused = pygame.mixer.Sound("sounds/pause.ogg")
player_death = pygame.mixer.Sound("sounds/player_death.ogg")
player_shoot = pygame.mixer.Sound("sounds/player_shoot.ogg")
graze = pygame.mixer.Sound("sounds/graze.ogg")
point = pygame.mixer.Sound("sounds/point.ogg")
enemy_death = pygame.mixer.Sound("sounds/enemy_death.ogg")
life_drop = pygame.mixer.Sound("sounds/get_life.ogg")
bomb_drop = pygame.mixer.Sound("sounds/get_bomb.ogg")
bombing = pygame.mixer.Sound("sounds/bomb.ogg")
bullet_sounds = []
for sound in range(1,6):
    bullet_sounds.append(pygame.mixer.Sound("sounds/bullet"+
        str(sound)+".ogg"))
```

Different sound effects are loaded for various game events such as pausing, player death, shooting, etc.
A loop loads multiple bullet sound effects into a list.

Setting Volumes for Sound Effects:

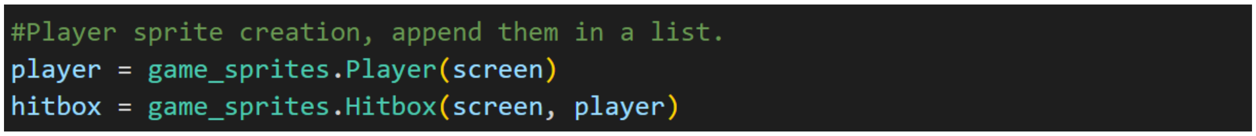


```
paused.set_volume(0.3)
player_death.set_volume(0.3)
player_shoot.set_volume(0.1)
graze.set_volume(0.3)
point.set_volume(0.3)
enemy_death.set_volume(0.4)
life_drop.set_volume(0.4)
bomb_drop.set_volume(0.4)
bombing.set_volume(0.4)
for bullet_sound in bullet_sounds:
    bullet_sound.set_volume(0.1)
```

The volume of each sound effect is set individually for better audio control during gameplay.

4. Sprite Creation and Initialization

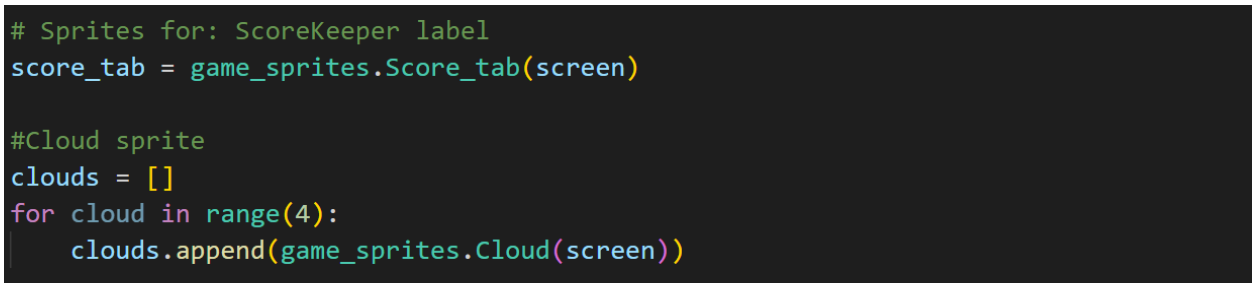
Creating Player and Hitbox:



```
#Player sprite creation, append them in a list.
player = game_sprites.Player(screen)
hitbox = game_sprites.Hitbox(screen, player)
```

An instance of the Player class is created, along with a Hitbox that tracks the player's collision area.

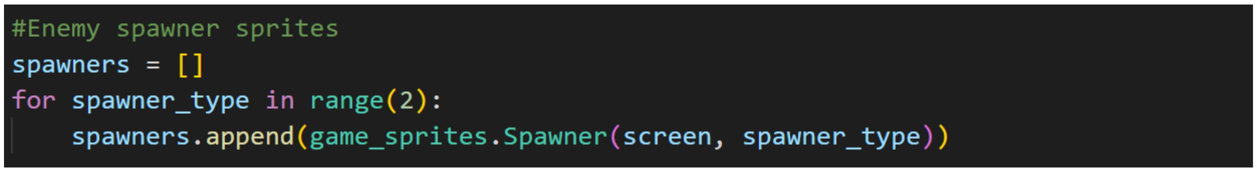
Creating Score Tab and Cloud Sprites:



```
# Sprites for: ScoreKeeper label
score_tab = game_sprites.Score_tab(screen)

#Cloud sprite
clouds = []
for cloud in range(4):
    clouds.append(game_sprites.Cloud(screen))
```

A score tracking object is created, and multiple cloud sprites are instantiated for background visuals.

Creating Enemy Spawner Sprites:

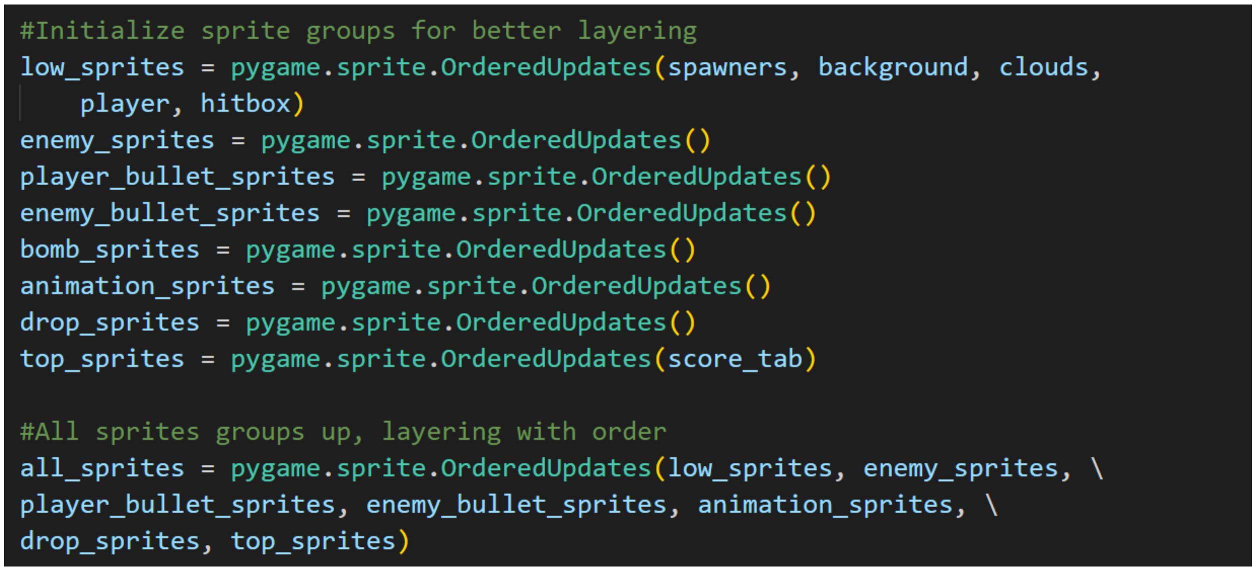


```
#Enemy spawner sprites
spawners = []
for spawner_type in range(2):
    spawners.append(game_sprites.Spawner(screen, spawner_type))
```

Two enemy spawner objects are created, which will handle the spawning of enemies during the game.

5. Sprite Group Initialization

Organizing Sprites into Groups:

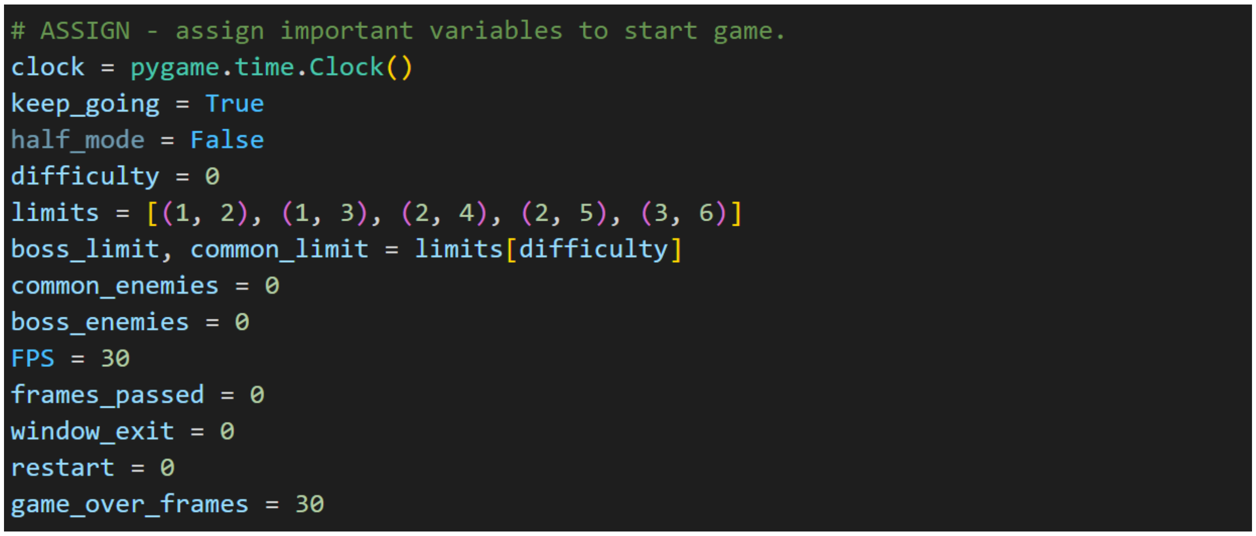


```
#Initialize sprite groups for better layering
low_sprites = pygame.sprite.OrderedUpdates(spawners, background, clouds,
    player, hitbox)
enemy_sprites = pygame.sprite.OrderedUpdates()
player_bullet_sprites = pygame.sprite.OrderedUpdates()
enemy_bullet_sprites = pygame.sprite.OrderedUpdates()
bomb_sprites = pygame.sprite.OrderedUpdates()
animation_sprites = pygame.sprite.OrderedUpdates()
drop_sprites = pygame.sprite.OrderedUpdates()
top_sprites = pygame.sprite.OrderedUpdates(score_tab)

#All sprites groups up, layering with order
all_sprites = pygame.sprite.OrderedUpdates(low_sprites, enemy_sprites, \
player_bullet_sprites, enemy_bullet_sprites, animation_sprites, \
drop_sprites, top_sprites)
```

Different groups of sprites are created for better management and rendering order.
all_sprites combines all sprite groups for overall rendering.

6. Game Variables Initialization

Setting Up Game Variables:



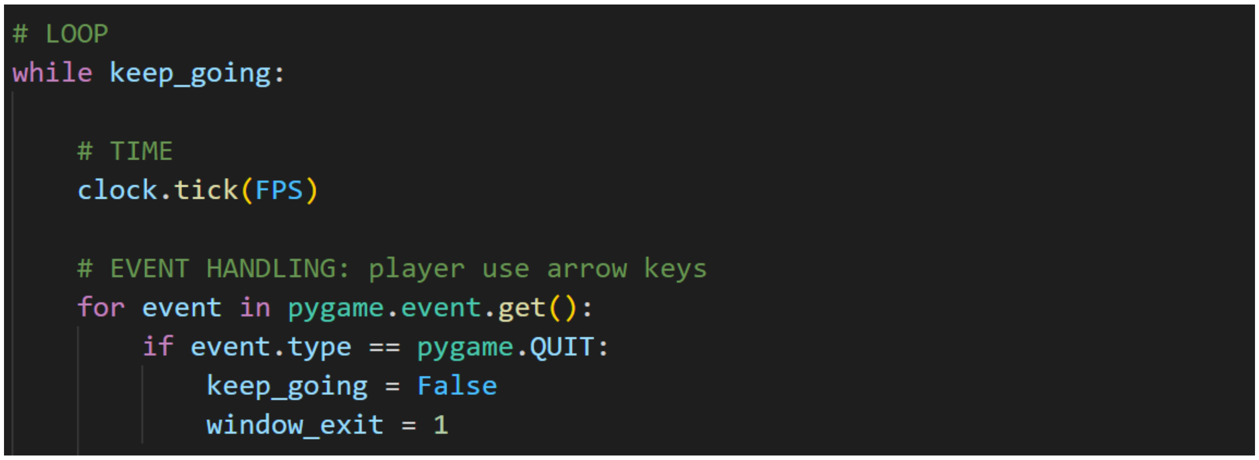
```
# ASSIGN - assign important variables to start game.
clock = pygame.time.Clock()
keep_going = True
half_mode = False
difficulty = 0
limits = [(1, 2), (1, 3), (2, 4), (2, 5), (3, 6)]
boss_limit, common_limit = limits[difficulty]
common_enemies = 0
boss_enemies = 0
FPS = 30
frames_passed = 0
window_exit = 0
restart = 0
game_over_frames = 30
```

A clock is created to control the frame rate.
Various game state variables are initialized, including difficulty levels, enemy limits, and game over conditions.

7. Main Game Loop

Michael Arianno Chandrarieta - 2802499711

Main Game Loop:



```
# LOOP
while keep_going:

    # TIME
    clock.tick(FPS)

    # EVENT HANDLING: player use arrow keys
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            keep_going = False
            window_exit = 1
```

1. Loop Control

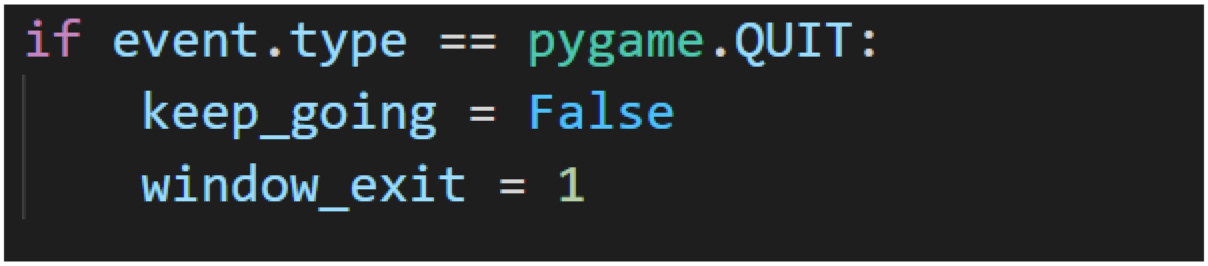
while keep_going:: This line starts the main game loop, which will continue executing as long as the keep_going variable is True. This variable is set to False when the game should exit or pause.

2. Frame Rate Control

clock.tick(FPS): This line controls the frame rate of the game. By calling clock.tick(FPS), the game loop will run at the specified frames per second (FPS), which is set to 30. This ensures that the game runs smoothly and consistently across different hardware.

3. Event Handling

for event in pygame.event.get():
This loop processes all the events that have occurred since the last frame (like key presses, mouse movements, etc.). Each event is handled to determine how the game should respond. Inside this event handling loop, various conditions are checked:

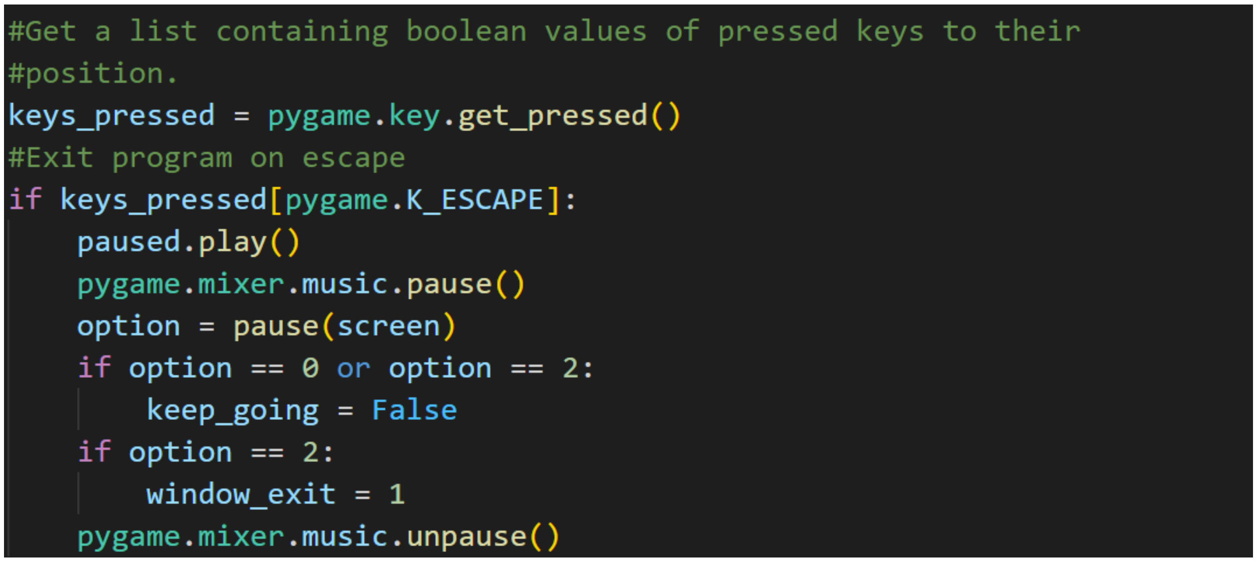


```
if event.type == pygame.QUIT:
    keep_going = False
    window_exit = 1
```

Quit Event: If the player closes the game window (e.g., by clicking the close button), keep_going is set to False, which will exit the loop.

Michael Arianno Chandrarieta - 2802499711

4. Key Press Handling



```
#Get a list containing boolean values of pressed keys to their
#position.
keys_pressed = pygame.key.get_pressed()
#Exit program on escape
if keys_pressed[pygame.K_ESCAPE]:
    paused.play()
    pygame.mixer.music.pause()
    option = pause(screen)
    if option == 0 or option == 2:
        keep_going = False
    if option == 2:
        window_exit = 1
    pygame.mixer.music.unpause()
```

Key Presses: The game checks for specific key presses to control player actions and game states.

Pause Game: If the 'P' key is pressed, the game pauses, and the pause_game function is called.

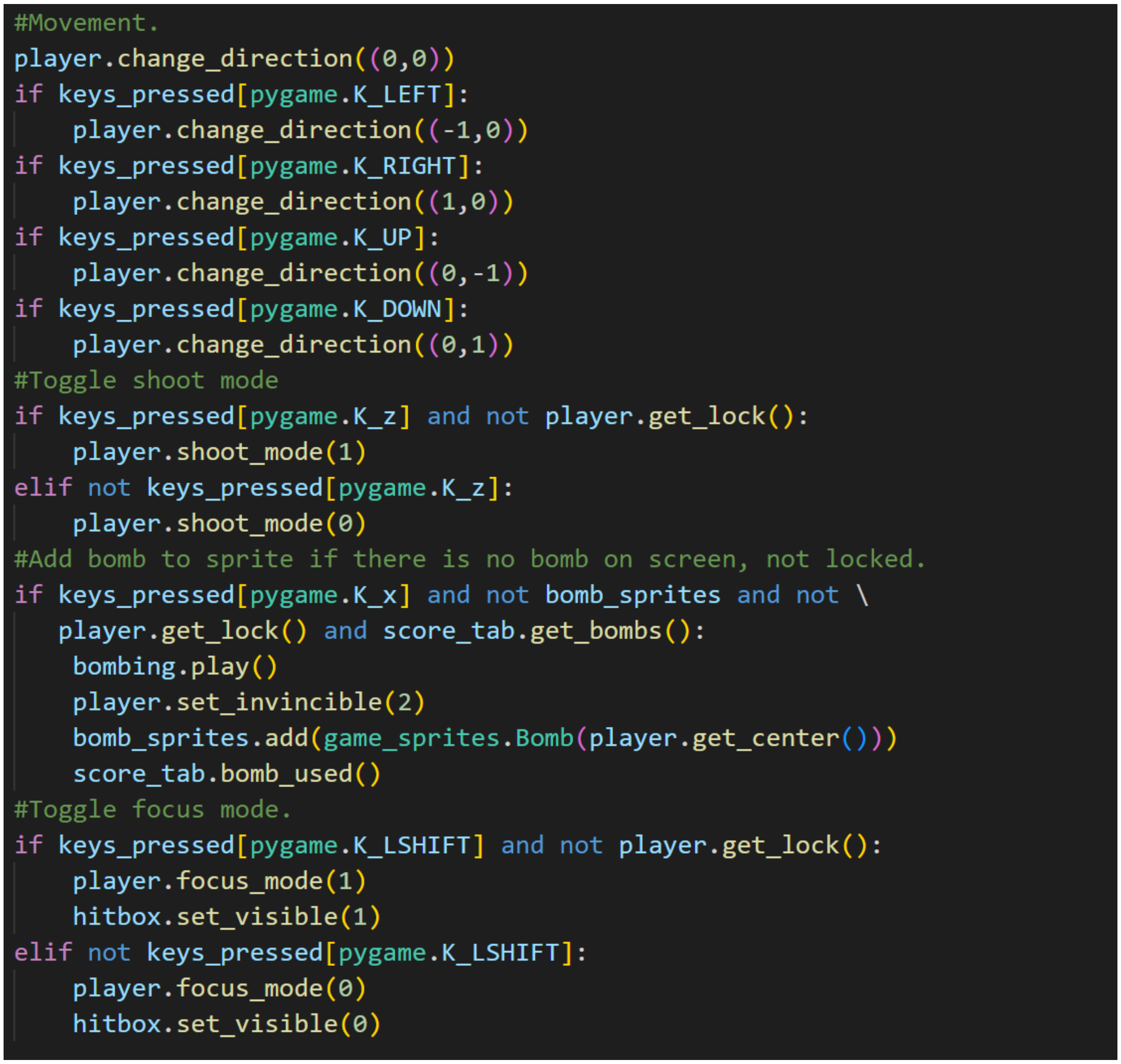
Restart Game: If the 'R' key is pressed, the restart variable is set to 1, and keep_going is set to False to exit the loop.

Exit Game: If the 'ESC' key is pressed, the game will exit by setting keep_going to False.

5. Player Movement and Actions

The game checks for movement key presses (arrow keys) and calls the appropriate player movement methods. For example:

Michael Arianno Chandrarieta - 2802499711



```
#Movement.
player.change_direction((0,0))
if keys_pressed[pygame.K_LEFT]:
    player.change_direction((-1,0))
if keys_pressed[pygame.K_RIGHT]:
    player.change_direction((1,0))
if keys_pressed[pygame.K_UP]:
    player.change_direction((0,-1))
if keys_pressed[pygame.K_DOWN]:
    player.change_direction((0,1))
#Toggle shoot mode
if keys_pressed[pygame.K_z] and not player.get_lock():
    player.shoot_mode(1)
elif not keys_pressed[pygame.K_z]:
    player.shoot_mode(0)
#Add bomb to sprite if there is no bomb on screen, not locked.
if keys_pressed[pygame.K_x] and not bomb_sprites and not \
   player.get_lock() and score_tab.get_bombs():
    bombing.play()
    player.set_invincible(2)
    bomb_sprites.add(game_sprites.Bomb(player.get_center()))
    score_tab.bomb_used()
#Toggle focus mode.
if keys_pressed[pygame.K_LSHIFT] and not player.get_lock():
    player.focus_mode(1)
    hitbox.set_visible(1)
elif not keys_pressed[pygame.K_LSHIFT]:
    player.focus_mode(0)
    hitbox.set_visible(0)
```

Player Movement: If the left arrow key is pressed, the player moves left by calling the move_left method on the player object.

Shooting and Bombing:

If the 'Z' key is pressed, the player shoots bullets, and the shooting sound effect is played.

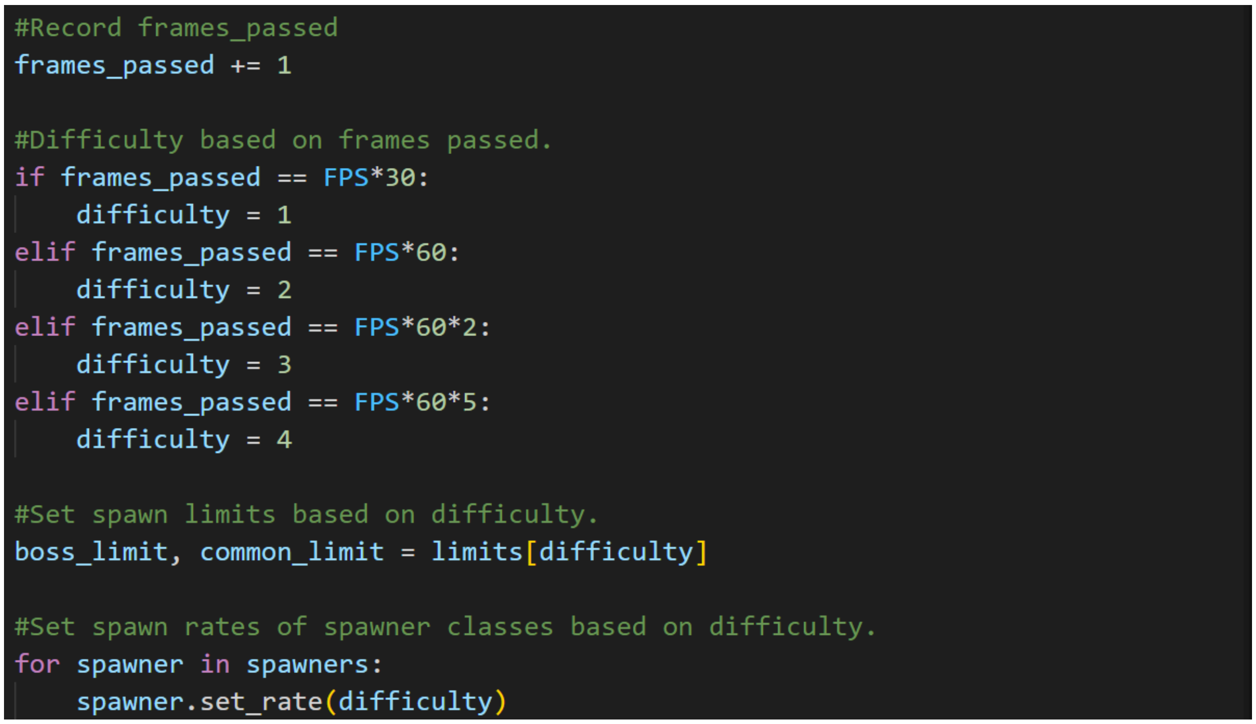
If the 'X' key is pressed, the player uses a bomb, and the bombing sound effect is played.

If the ‘left shift’ key is pressed, the player gets slower in movement

Michael Arianno Chandrarieta - 2802499711

6. Game Logic Updates

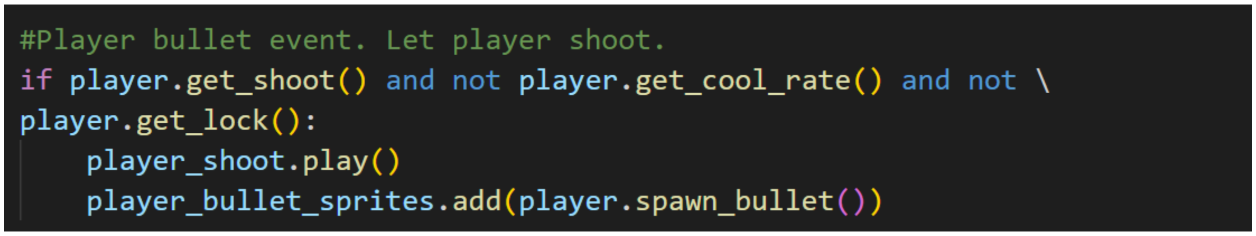
Frame Count and Difficulty Adjustment:



```python
#Record frames_passed
frames_passed += 1

#Difficulty based on frames passed.
if frames_passed == FPS*30:
    difficulty = 1
elif frames_passed == FPS*60:
    difficulty = 2
elif frames_passed == FPS*60*2:
    difficulty = 3
elif frames_passed == FPS*60*5:
    difficulty = 4

#Set spawn limits based on difficulty.
boss_limit, common_limit = limits[difficulty]

#Set spawn rates of spawner classes based on difficulty.
for spawner in spawners:
    spawner.set_rate(difficulty)
```

The frames_passed variable increments with each loop iteration. Every 60 frames (approximately every second), the game increases the difficulty level by updating boss_limit and common_limit based on predefined limits.

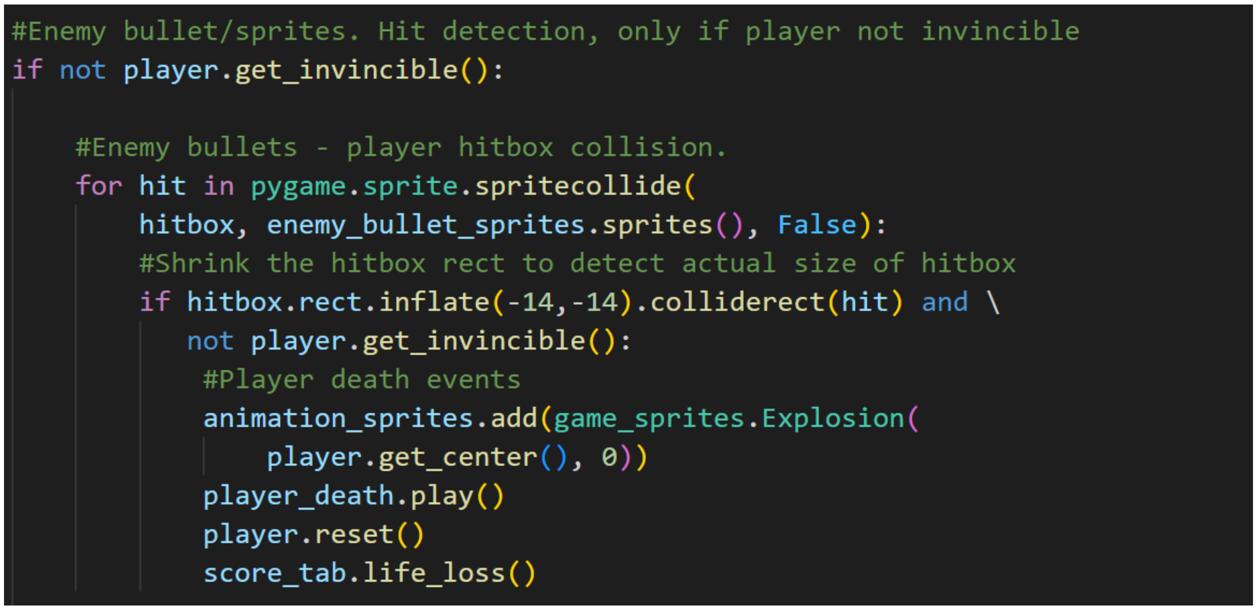
7. Player Shooting



```python
#Player bullet event. Let player shoot.
if player.get_shoot() and not player.get_cool_rate() and not \
player.get_lock():
    player_shoot.play()
    player_bullet_sprites.add(player.spawn_bullet())
```

Player Shooting: This block checks if the player is allowed to shoot (not on cooldown and not locked). If true, it plays the shooting sound and adds a new bullet to the player_bullet_sprites group.

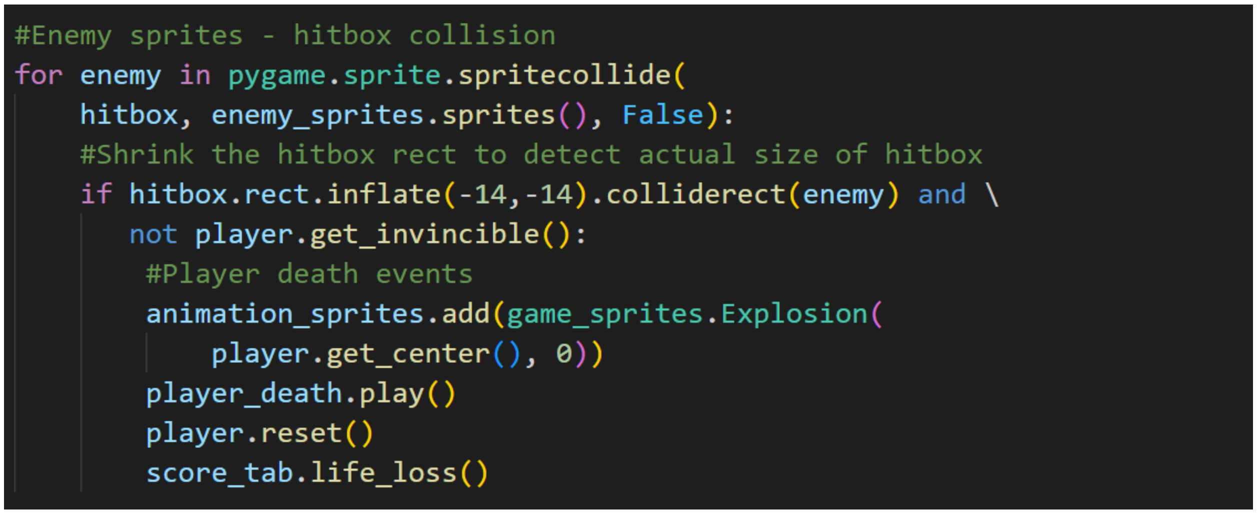
8. Player Hit Detection



```
#Enemy bullet/sprites. Hit detection, only if player not invincible
if not player.get_invincible():

    #Enemy bullets - player hitbox collision.
    for hit in pygame.sprite.spritecollide(
        hitbox, enemy_bullet_sprites.sprites(), False):
        #Shrink the hitbox rect to detect actual size of hitbox
        if hitbox.rect.inflate(-14,-14).colliderect(hit) and \
           not player.get_invincible():
            #Player death events
            animation_sprites.add(game_sprites.Explosion(
                player.get_center(), 0))
            player_death.play()
            player.reset()
            score_tab.life_loss()
```

Hit Detection: This block checks if the player is invincible. If not, it checks for collisions between the player's hitbox and enemy bullets. If a collision occurs, it triggers an explosion animation, plays the death sound, resets the player, and decreases the player's life count.

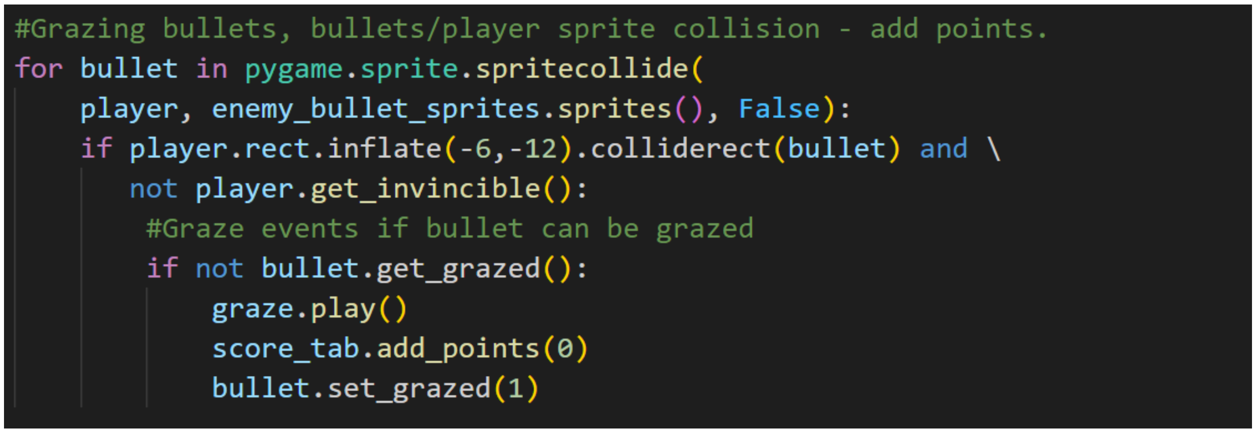
9. Enemy Collision Detection



```
#Enemy sprites - hitbox collision
for enemy in pygame.sprite.spritecollide(
    hitbox, enemy_sprites.sprites(), False):
    #Shrink the hitbox rect to detect actual size of hitbox
    if hitbox.rect.inflate(-14,-14).colliderect(enemy) and \
       not player.get_invincible():
        #Player death events
        animation_sprites.add(game_sprites.Explosion(
            player.get_center(), 0))
        player_death.play()
        player.reset()
        score_tab.life_loss()
```

Enemy Collision: Similar to the previous block, this checks for collisions between the player's hitbox and enemy sprites. If a collision is detected and the player is not invincible, it processes the same death events.

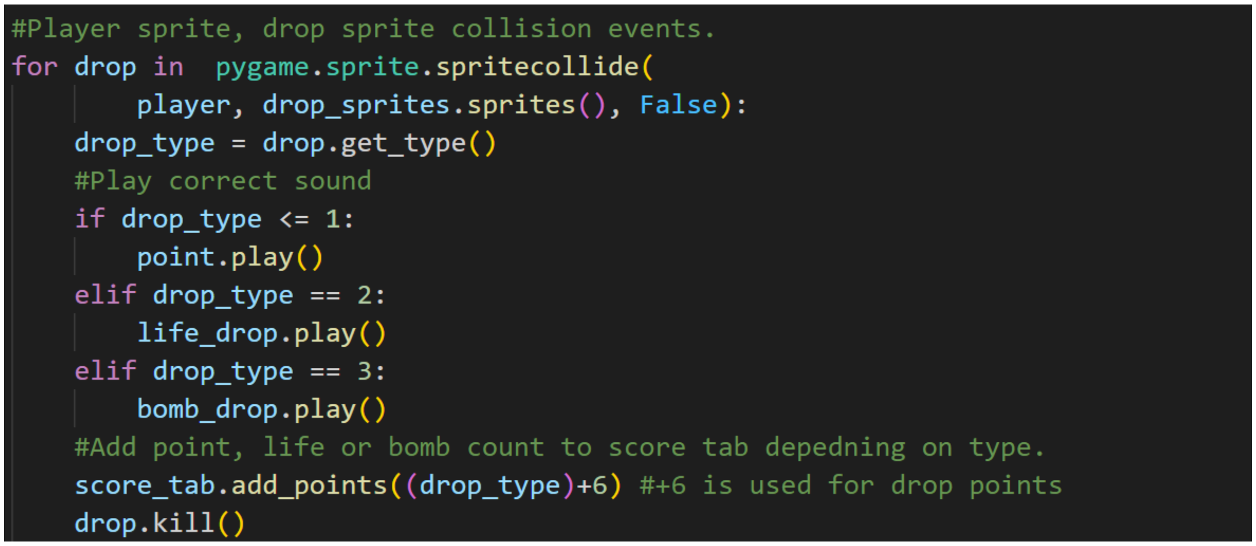
10. Grazing Bullets



```python
#Grazing bullets, bullets/player sprite collision - add points.
for bullet in pygame.sprite.spritecollide(
    player, enemy_bullet_sprites.sprites(), False):
    if player.rect.inflate(-6,-12).colliderect(bullet) and \
       not player.get_invincible():
        #Graze events if bullet can be grazed
        if not bullet.get_grazed():
            graze.play()
            score_tab.add_points(0)
            bullet.set_grazed(1)
```

Grazing: This block detects if the player grazes enemy bullets (i.e., comes very close without getting hit). If the player is not invincible and the bullet has not been grazed before, it plays a grazing sound and adds points.

11. Collecting Drops

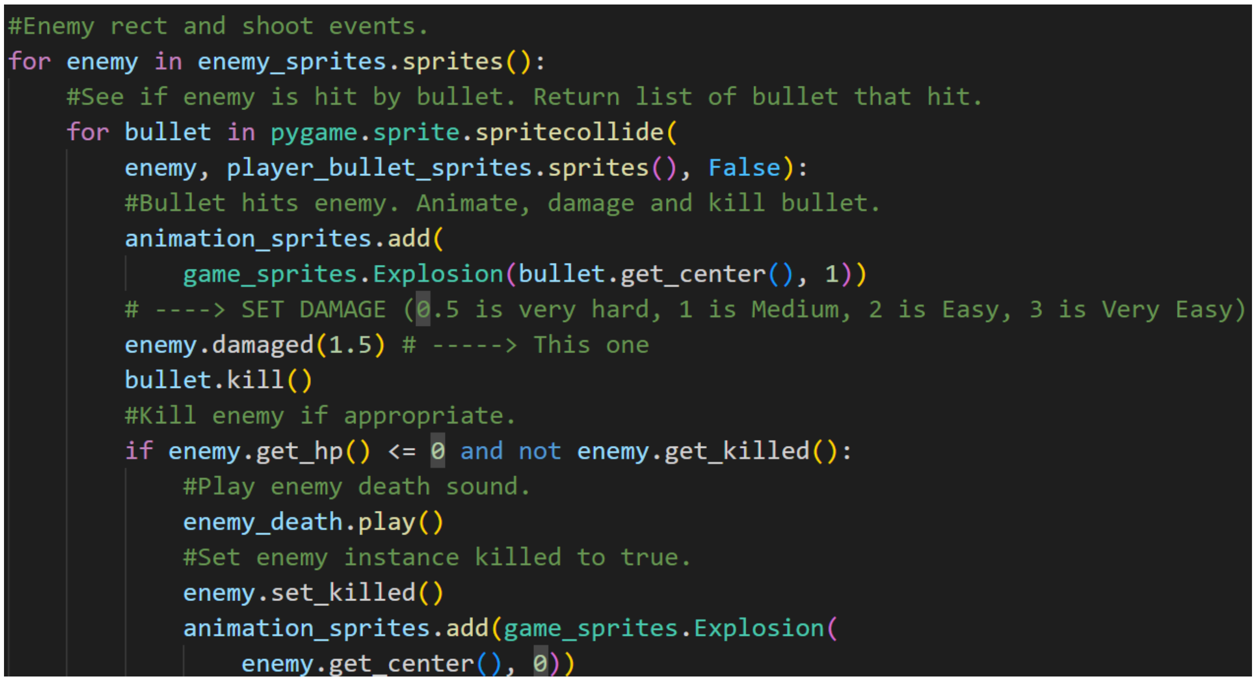


```python
#Player sprite, drop sprite collision events.
for drop in  pygame.sprite.spritecollide(
        player, drop_sprites.sprites(), False):
    drop_type = drop.get_type()
    #Play correct sound
    if drop_type <= 1:
        point.play()
    elif drop_type == 2:
        life_drop.play()
    elif drop_type == 3:
        bomb_drop.play()
    #Add point, life or bomb count to score tab depedning on type.
    score_tab.add_points((drop_type)+6) #+6 is used for drop points
    drop.kill()
```

Collecting Drops: This block checks for collisions between the player and collectible drops (like points, lives, or bombs). Depending on the drop type, it plays the corresponding sound, adds points to the score, and removes the drop from the game.

12. Enemy Hit Detection and Damage

Michael Arianno Chandrarieta - 2802499711

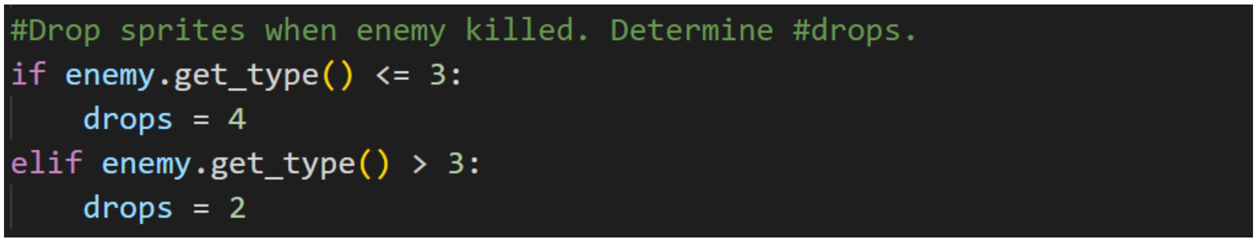


```
#Enemy rect and shoot events.
for enemy in enemy_sprites.sprites():
    #See if enemy is hit by bullet. Return list of bullet that hit.
    for bullet in pygame.sprite.spritecollide(
        enemy, player_bullet_sprites.sprites(), False):
        #Bullet hits enemy. Animate, damage and kill bullet.
        animation_sprites.add(
            game_sprites.Explosion(bullet.get_center(), 1))
        # ----> SET DAMAGE (0.5 is very hard, 1 is Medium, 2 is Easy, 3 is Very Easy)
        enemy.damaged(1.5) # -----> This one
        bullet.kill()
        #Kill enemy if appropriate.
        if enemy.get_hp() <= 0 and not enemy.get_killed():
            #Play enemy death sound.
            enemy_death.play()
            #Set enemy instance killed to true.
            enemy.set_killed()
            animation_sprites.add(game_sprites.Explosion(
                enemy.get_center(), 0))
```

Enemy Damage: This block checks if any player bullets collide with enemies. If a bullet hits, it plays an explosion animation, applies damage to the enemy, and removes the bullet. If the enemy's health drops to zero, it triggers death events, including playing a death sound, setting the enemy as killed, and creating an explosion animation.

Enemy Drops

Determine Number of Drops

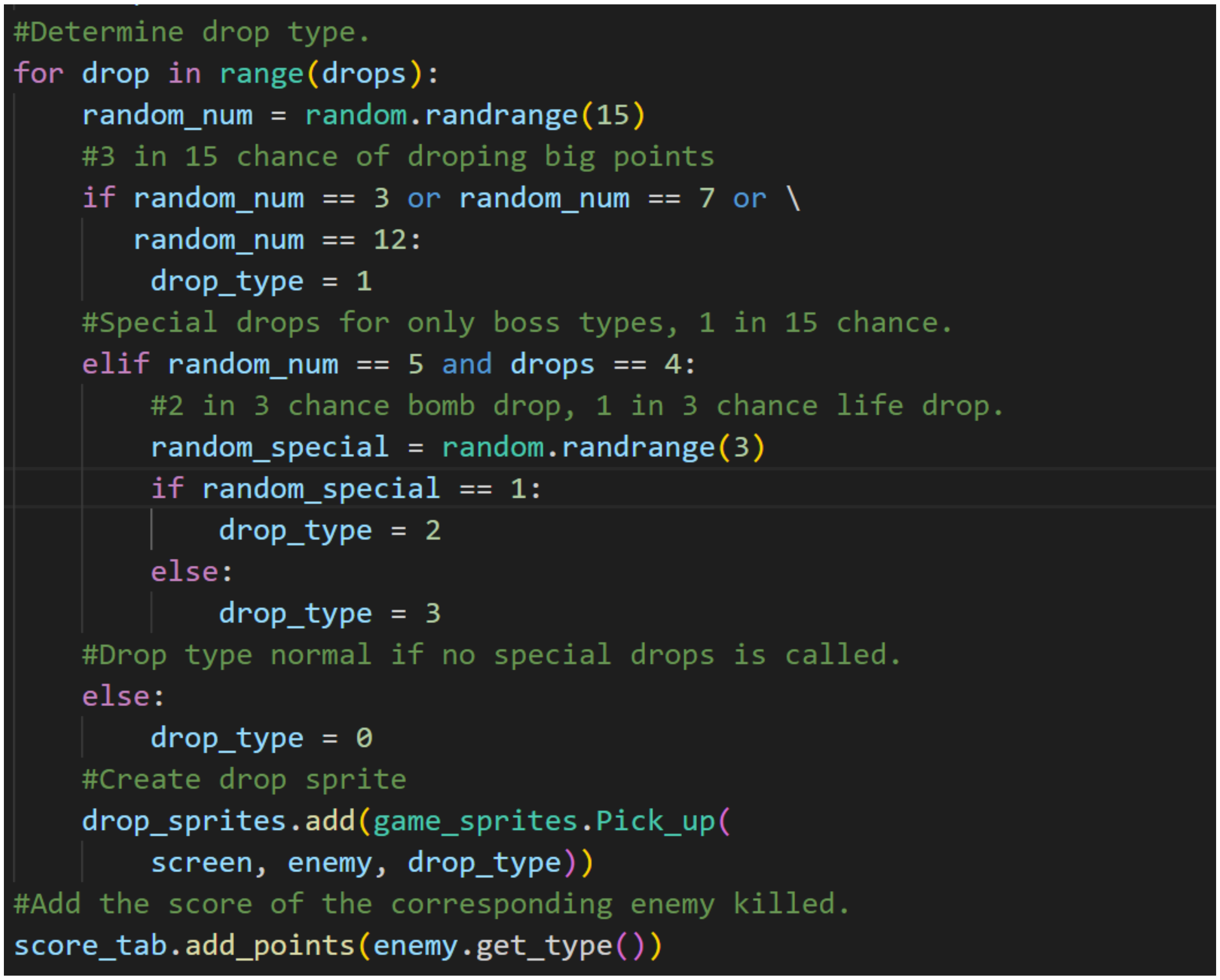


```
#Drop sprites when enemy killed. Determine #drops.
if enemy.get_type() <= 3:
    drops = 4
elif enemy.get_type() > 3:
    drops = 2
```

Drop Count Based on Enemy Type: This block determines how many collectible drops to generate based on the type of the enemy that was killed.
If the enemy type is less than or equal to 3 (which typically represents a boss or stronger enemy), it will drop 4 items.
If the enemy type is greater than 3 (representing weaker common enemies), it will drop 2 items.

Determine Drop Type

Michael Arianno Chandrarieta - 2802499711



```
#Determine drop type.
for drop in range(drops):
    random_num = random.randrange(15)
    #3 in 15 chance of droping big points
    if random_num == 3 or random_num == 7 or \
       random_num == 12:
        drop_type = 1
    #Special drops for only boss types, 1 in 15 chance.
    elif random_num == 5 and drops == 4:
        #2 in 3 chance bomb drop, 1 in 3 chance life drop.
        random_special = random.randrange(3)
        if random_special == 1:
            drop_type = 2
        else:
            drop_type = 3
    #Drop type normal if no special drops is called.
    else:
        drop_type = 0
    #Create drop sprite
    drop_sprites.add(game_sprites.Pick_up(
        screen, enemy, drop_type))
#Add the score of the corresponding enemy killed.
score_tab.add_points(enemy.get_type())
```

Randomized Drop Logic: This loop iterates for the number of drops determined earlier. For each drop, it generates a random number from 0 to 14 (using random.randrange(15)).

Big Points Drop: If the random number is 3, 7, or 12, it sets the drop_type to 1, which represents a drop of big points.

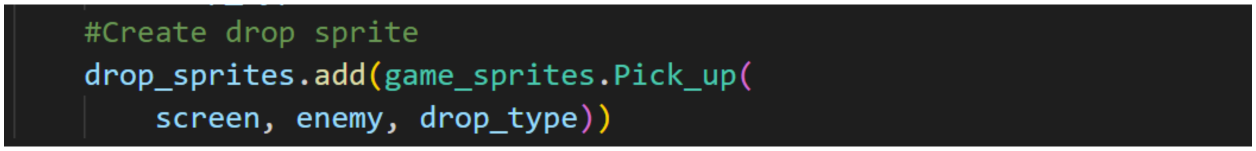
Special Drops for Bosses: If the enemy is a boss (indicated by drops == 4), there's a chance (1 in 15) that it will drop either a bomb or a life. This is determined by another random number:
If random_special equals 1, it sets the drop_type to 2 (bomb drop).

Otherwise, it sets the drop_type to 3 (life drop).

Normal Drop: If none of the conditions are met, the drop_type is set to 0, indicating a standard points drop.

Create Drop Sprite

Michael Arianno Chandrarieta - 2802499711

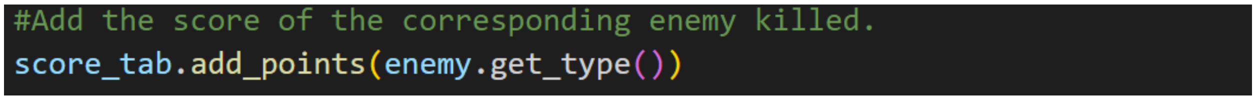


```
    #Create drop sprite
    drop_sprites.add(game_sprites.Pick_up(
        screen, enemy, drop_type))
```

Creating the Drop: This line creates a new drop sprite using the Pick_up class from the game_sprites module. It passes the screen, the enemy that was killed, and the determined drop_type to create the appropriate collectible item.

The created drop sprite is then added to the drop_sprites group, which manages all collectible items on the screen.

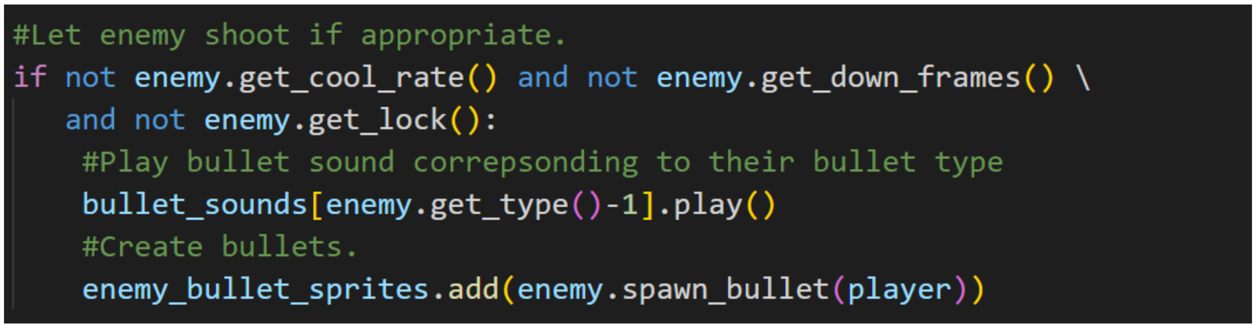
Add Points for Killed Enemy



```
#Add the score of the corresponding enemy killed.
score_tab.add_points(enemy.get_type())
```

Scoring: This line updates the player's score by adding points corresponding to the type of the enemy that was killed. The get_type() method likely returns a value that determines how many points the enemy is worth.

13. Enemy Shooting

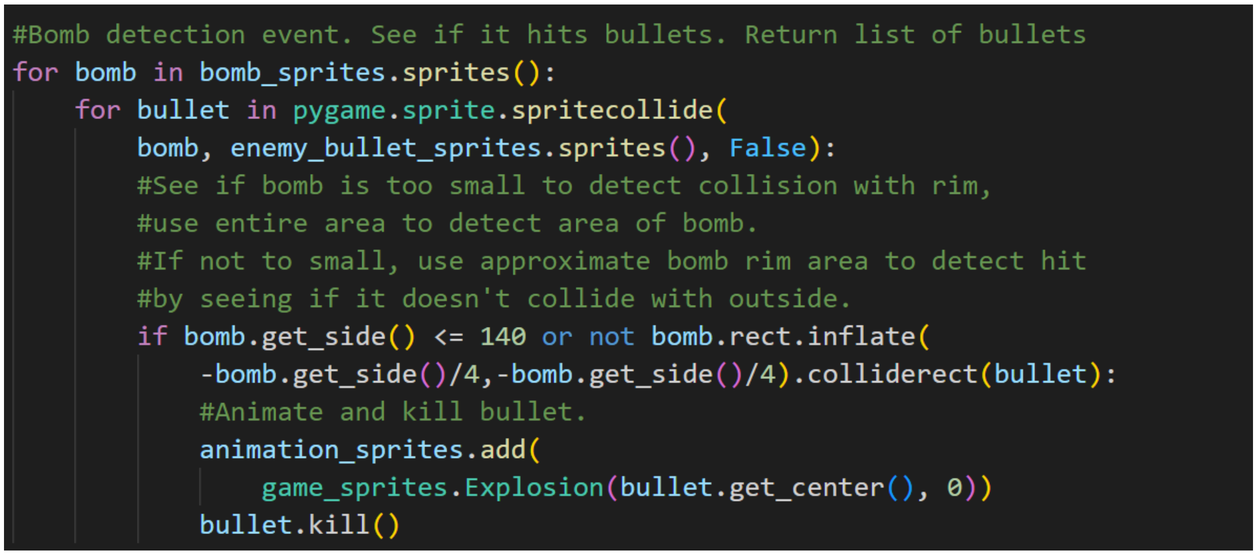


```
#Let enemy shoot if appropriate.
if not enemy.get_cool_rate() and not enemy.get_down_frames() \
   and not enemy.get_lock():
    #Play bullet sound correpsonding to their bullet type
    bullet_sounds[enemy.get_type()-1].play()
    #Create bullets.
    enemy_bullet_sprites.add(enemy.spawn_bullet(player))
```

Enemy Shooting Condition: This block checks if the enemy can shoot by verifying that it is not on cooldown (get_cool_rate()), not in a down state (get_down_frames()), and not locked (get_lock()).

Sound and Bullet Creation: If the conditions are met, it plays the appropriate bullet sound for that enemy type and adds a new bullet to the enemy_bullet_sprites group by calling the spawn_bullet method.

14. Bomb Detection

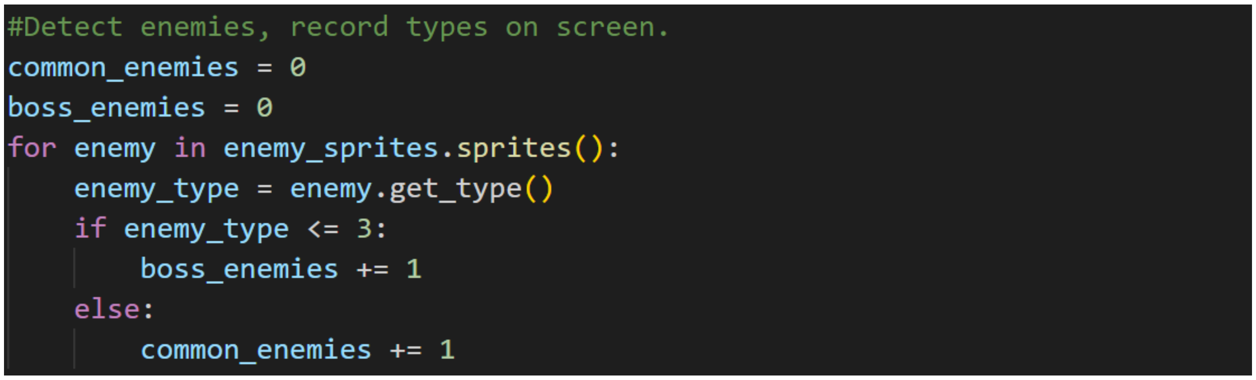


```python
#Bomb detection event. See if it hits bullets. Return list of bullets
for bomb in bomb_sprites.sprites():
    for bullet in pygame.sprite.spritecollide(
        bomb, enemy_bullet_sprites.sprites(), False):
        #See if bomb is too small to detect collision with rim,
        #use entire area to detect area of bomb.
        #If not to small, use approximate bomb rim area to detect hit
        #by seeing if it doesn't collide with outside.
        if bomb.get_side() <= 140 or not bomb.rect.inflate(
            -bomb.get_side()/4,-bomb.get_side()/4).colliderect(bullet):
            #Animate and kill bullet.
            animation_sprites.add(
                game_sprites.Explosion(bullet.get_center(), 0))
            bullet.kill()
```

Collision Detection: This nested loop checks for collisions between bombs and enemy bullets.

Collision Handling: If a bomb is either small enough (get_side() <= 140) or its inflated rectangle does not collide with the bullet, it triggers an explosion animation and removes the bullet.

15. Enemy Type Detection

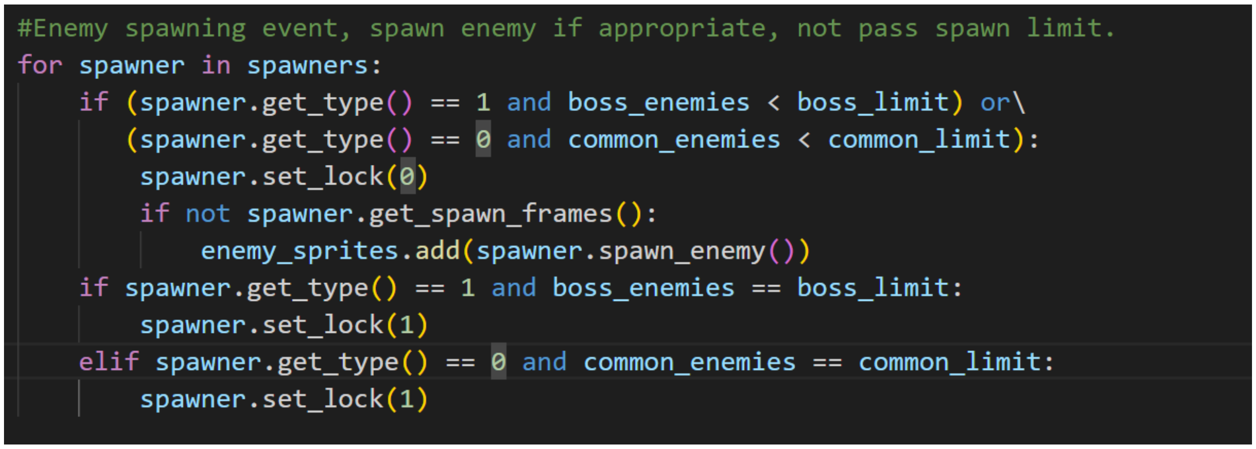


```python
#Detect enemies, record types on screen.
common_enemies = 0
boss_enemies = 0
for enemy in enemy_sprites.sprites():
    enemy_type = enemy.get_type()
    if enemy_type <= 3:
        boss_enemies += 1
    else:
        common_enemies += 1
```

Counting Enemies: This block counts the number of common and boss enemies currently on the screen by checking each enemy's type. Boss enemies are those with types less than or equal to 3.

16. Enemy Spawning

Michael Arianno Chandrarieta - 2802499711

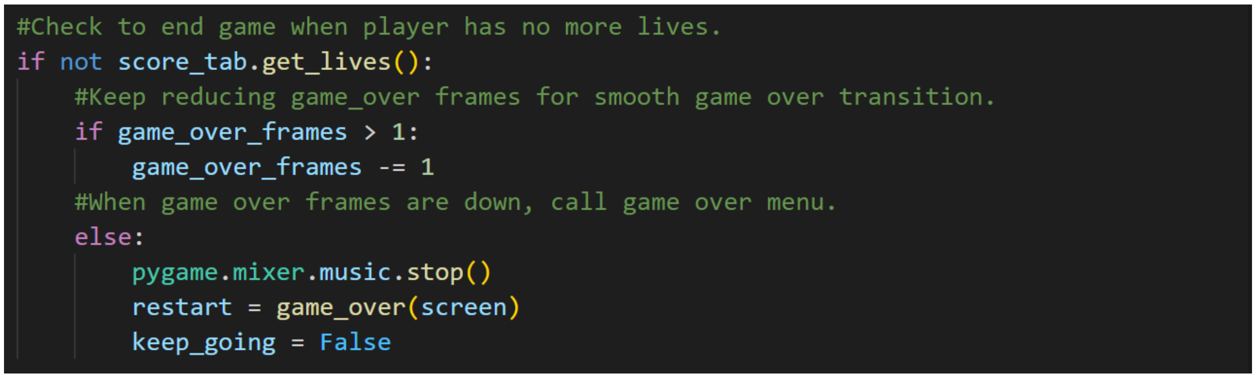


```
#Enemy spawning event, spawn enemy if appropriate, not pass spawn limit.
for spawner in spawners:
    if (spawner.get_type() == 1 and boss_enemies < boss_limit) or\
       (spawner.get_type() == 0 and common_enemies < common_limit):
        spawner.set_lock(0)
        if not spawner.get_spawn_frames():
            enemy_sprites.add(spawner.spawn_enemy())
    if spawner.get_type() == 1 and boss_enemies == boss_limit:
        spawner.set_lock(1)
    elif spawner.get_type() == 0 and common_enemies == common_limit:
        spawner.set_lock(1)
```

Spawning Logic: This section checks each spawner to see if it can spawn more enemies based on the current counts of common and boss enemies.

Locking Spawners: If the limits are reached, the spawner is locked to prevent further spawning until conditions change.

17. Game Over Check

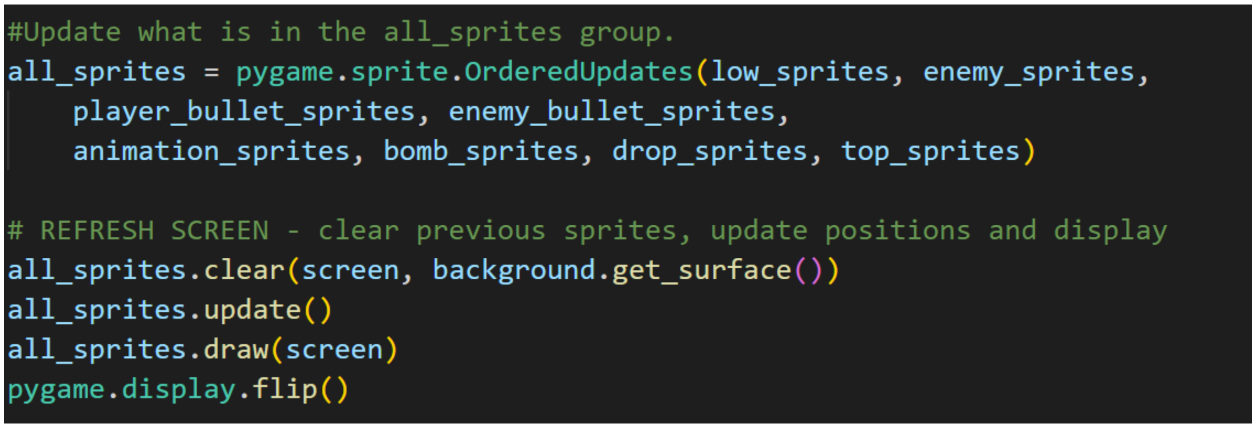


```
#Check to end game when player has no more lives.
if not score_tab.get_lives():
    #Keep reducing game_over frames for smooth game over transition.
    if game_over_frames > 1:
        game_over_frames -= 1
    #When game over frames are down, call game over menu.
    else:
        pygame.mixer.music.stop()
        restart = game_over(screen)
        keep_going = False
```

Game Over Condition: This block checks if the player has no lives left. If true, it reduces the game_over_frames to create a smooth transition.

Game Over Menu: When the frames reach zero, it stops the background music, calls the game_over function to display the game over menu, and sets keep_going to False to exit the loop.

18. Updating Sprites

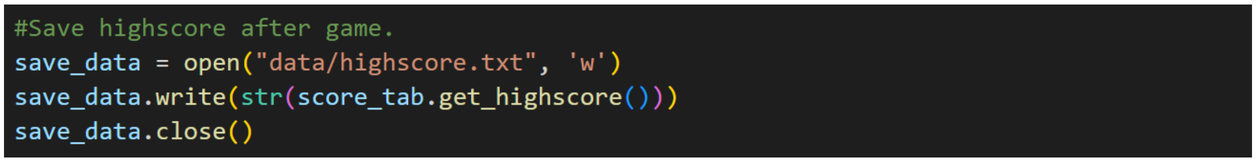


```python
#Update what is in the all_sprites group.
all_sprites = pygame.sprite.OrderedUpdates(low_sprites, enemy_sprites,
    player_bullet_sprites, enemy_bullet_sprites,
    animation_sprites, bomb_sprites, drop_sprites, top_sprites)

# REFRESH SCREEN - clear previous sprites, update positions and display
all_sprites.clear(screen, background.get_surface())
all_sprites.update()
all_sprites.draw(screen)
pygame.display.flip()
```

Updating Sprites: This section updates the all_sprites group by combining all sprite groups in the correct order.

Refreshing Screen: It clears the previous sprites, updates their positions, and redraws them on the screen using pygame.display.flip().

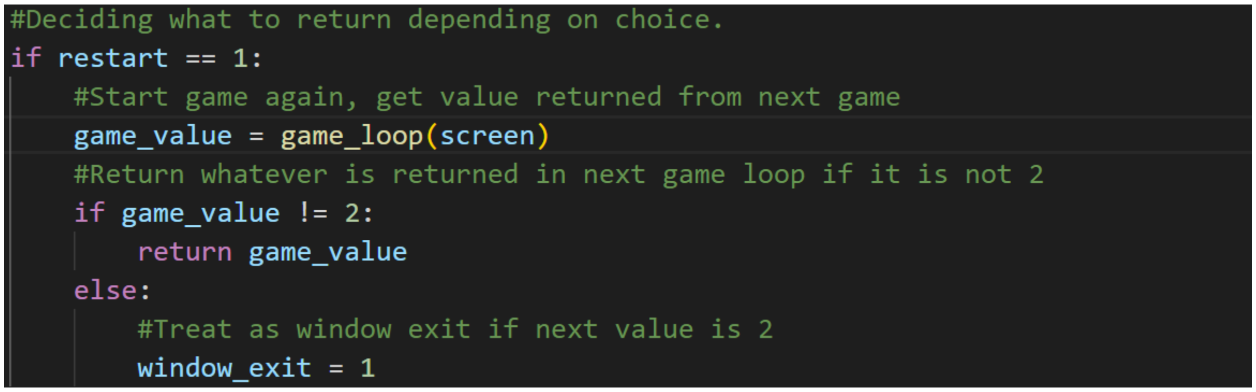
19. Saving High Score



```python
#Save highscore after game.
save_data = open("data/highscore.txt", 'w')
save_data.write(str(score_tab.get_highscore()))
save_data.close()
```

Saving High Score: After the game ends, this block saves the current high score to a file named highscore.txt in the data directory

20. Handling Restart Logic



```python
#Deciding what to return depending on choice.
if restart == 1:
    #Start game again, get value returned from next game
    game_value = game_loop(screen)
    #Return whatever is returned in next game loop if it is not 2
    if game_value != 2:
        return game_value
    else:
        #Treat as window exit if next value is 2
        window_exit = 1
```

Restarting the Game: This block checks if the restart variable is set to 1, indicating that the player chose to restart the game.

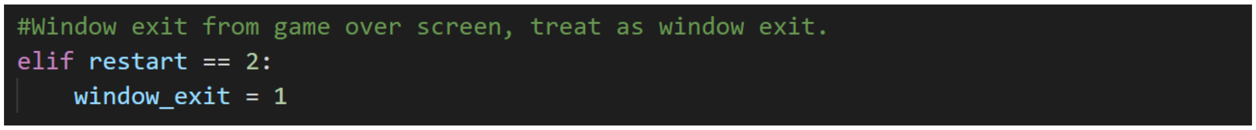
Calling the Game Loop: The game_loop(screen) function is called, which starts a new game session. The return value from this function is stored in game_value.

Return Logic:

If game_value is not 2, it returns the value directly. This value could represent various states (like continuing the game, going back to the main menu, etc.).

If game_value is 2, it sets window_exit to 1, indicating that the player has exited the window.

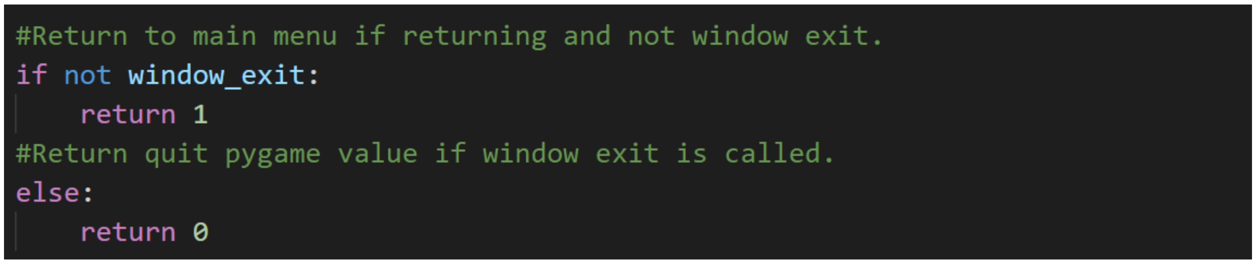
21. Handling Exit Logic from Game Over Screen



```
#Window exit from game over screen, treat as window exit.
elif restart == 2:
    window_exit = 1
```

Exiting the Game: If the restart variable is set to 2, it means the player chose to exit the game from the game over screen. In this case, it sets window_exit to 1 to indicate that the game should close.

22. Final Return Logic

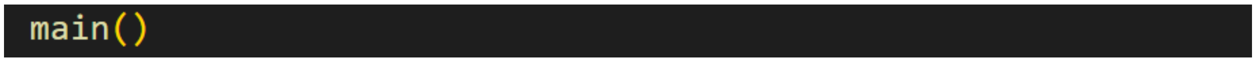


```
#Return to main menu if returning and not window exit.
if not window_exit:
    return 1
#Return quit pygame value if window exit is called.
else:
    return 0
```

Return to Main Menu: If window_exit is not set (i.e., not window_exit evaluates to True), it returns 1, which likely indicates that the game should return to the main menu.

Exiting the Game: If window_exit is set to 1, it returns 0, which typically signals that the game should quit.

23. Calling the Main Function



```
main()
```

Starting the Game: This line calls the main() function, which is the entry point of the game. This function initializes everything and starts the game loop.

Michael Arianno Chandrarieta - 2802499711