CHARLES DARWIN UNIVERSITY AUSTRALIA

UNIT CODE: HIT137
Software Now

Assignment-3

Submitted By

Group: CAS-119

Student Name	Student ID	Email
Synthia Islam	# S375728	S375728@students.cdu.edu.au
Hussein Salami	#S371192	S371192@students.cdu.edu.au
Sayeed Anwar	#S384116	S384116@students.cdu.edu.au
A K M Shafiur Rahman	#S372618	S372618@students.cdu.edu.au

Table of Contents

Question 1: - [Synthia Islam, A K M Shafiur Rahman, Sayeed Anwar]

Create a Tkinter application using the concepts of object-oriented programming, such as, multiple inheritance, multiple decorators, encapsulation, polymorphism, and method overriding, etc.
Wherever you use these concepts in code, explain them using # (How they are linked to code).
Examples:

- Youtube Like Interface Applications.
- Using single/multiple Open-Source AI models (smaller) to a create desktop application. Ex: An Image classification application, Language translation app, object detection app, facial recognition app, etc.

(You just need to download the models and add its input points to the buttons in your tkinter application. Whatever the user enters into the application should go the AI model and give a output on the application.)

ANSWER:

(Code files are included under the "tkinter-question-1" folder attached in the zip file; main app file is "translator_app.py", which needs to be run)

Overview

This Translator Application is a simple GUI-based translation tool built using Python's Tkinter for the GUI for the translation functionality. It allows users to translate text from English to French and English to German languages by leveraging pre-trained AI models.

Installation Requirements:

To run the Translator Application, the following libraries need to be installed in the environment.

- **Packages to Install:**

  1. ***tkinter*** (for the GUI)
     - On Windows, *tkinter* usually comes pre-installed with Python.
     - On Linux:

$$sudo\ apt - get\ install\ python3 - tk$$

  2. ***transformers*** (for using pre-trained models from Hugging Face for translations):

$$pip\ install\ transformers$$

How to Run the Application:

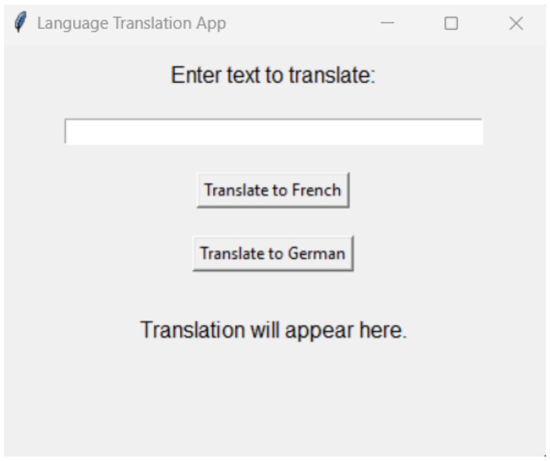
Once all the required packages are installed, you can run the Translator Application by following these steps:

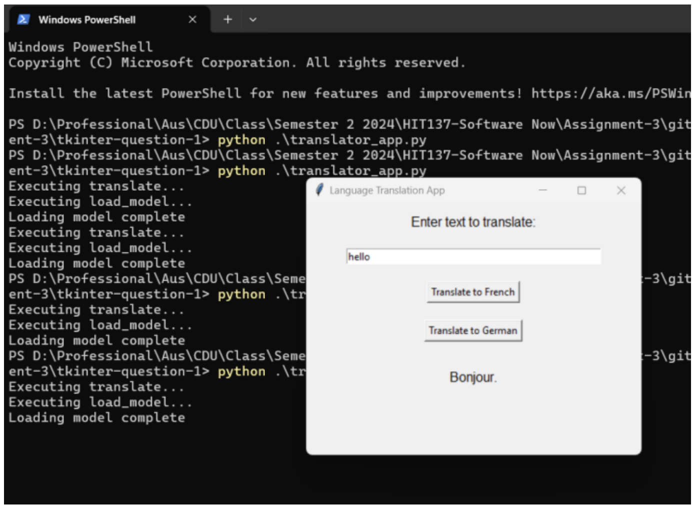
- Open a terminal or command prompt in the project directory.
- Run the following command:

$$python\ translator\_app.py$$

  This will launch the Translator Application's GUI. Here, it needs to be noted that it will take some time to load the application for the first time.

- Input the text you want to translate.
- Click on the "Translate to French" button to check the translated French text.
- Click on the "Translate to German" button to check the translated German text.
- The translated text will be displayed in the designated area.





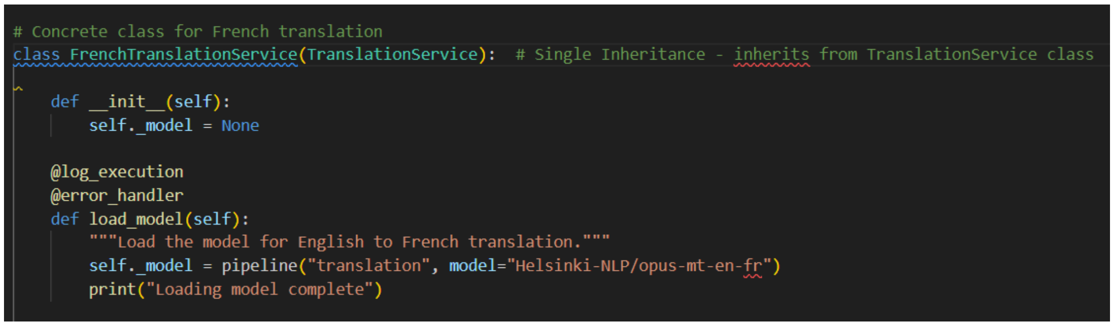
```
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Install the latest PowerShell for new features and improvements! https://aka.ms/PSWin

PS D:\Professional\Aus\CDU\Class\Semester 2 2024\HIT137-Software Now\Assignment-3\git
ent-3\tkinter-question-1> python .\translator_app.py
PS D:\Professional\Aus\CDU\Class\Semester 2 2024\HIT137-Software Now\Assignment-3\git
ent-3\tkinter-question-1> python .\translator_app.py
Executing translate...
Executing load_model...
Loading model complete
Executing translate...
Executing load_model...
Loading model complete
PS D:\Professional\Aus\CDU\Class\Seme                                        -3\git
ent-3\tkinter-question-1> python .\tr
Executing translate...
Executing load_model...
Loading model complete
PS D:\Professional\Aus\CDU\Class\Seme                                        -3\git
ent-3\tkinter-question-1> python .\tr
Executing translate...
Executing load_model...
Loading model complete
```

➢ Features:

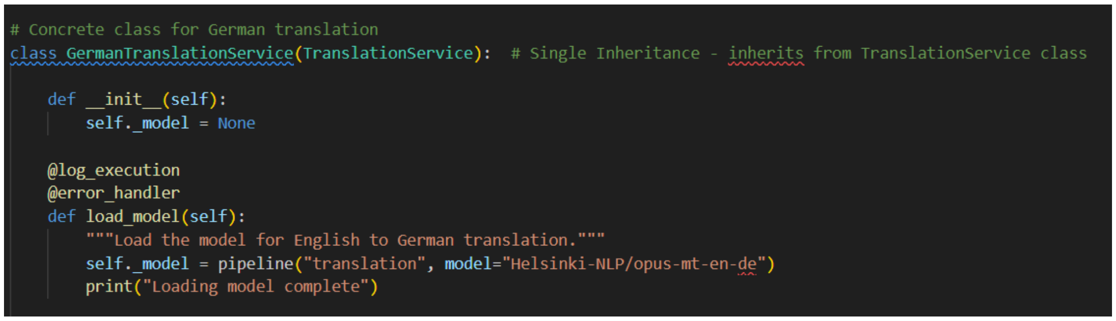
1. **Simple GUI**: The application uses a ***tkinter*** GUI, allowing users to input text and receive translations.

2. **Multiple Language Translation**:

   - The app uses the **"Helsinki-NLP/opus-mt-en-fr"** and **"Helsinki-NLP/opus-mt-en-de"** AI models, which are pre-trained machine translation models provided by the Hugging Face Model Hub. These are loaded through the *"pipeline"* method of the *"transformers"* package.



```python
# Concrete class for French translation
class FrenchTranslationService(TranslationService):  # Single Inheritance - inherits from TranslationService class

    def __init__(self):
        self._model = None

    @log_execution
    @error_handler
    def load_model(self):
        """Load the model for English to French translation."""
        self._model = pipeline("translation", model="Helsinki-NLP/opus-mt-en-fr")
        print("Loading model complete")
```



```python
# Concrete class for German translation
class GermanTranslationService(TranslationService):  # Single Inheritance - inherits from TranslationService class

    def __init__(self):
        self._model = None

    @log_execution
    @error_handler
    def load_model(self):
        """Load the model for English to German translation."""
        self._model = pipeline("translation", model="Helsinki-NLP/opus-mt-en-de")
        print("Loading model complete")
```

3. **Error Handling & Logging**: The app includes multiple decorators for logging execution of methods, logging the result of methods, and handling errors, ensuring a smooth user experience.

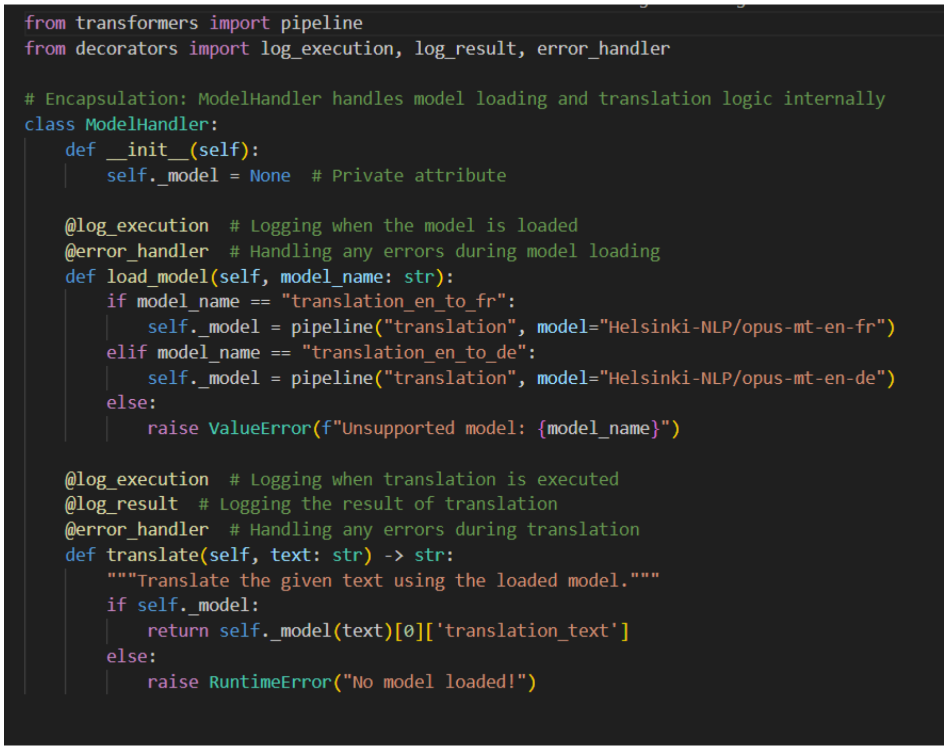
➢ OOP Implementation:

This Translator Application implements several key Object-Oriented Programming (OOP) concepts. Here's how OOP is applied:

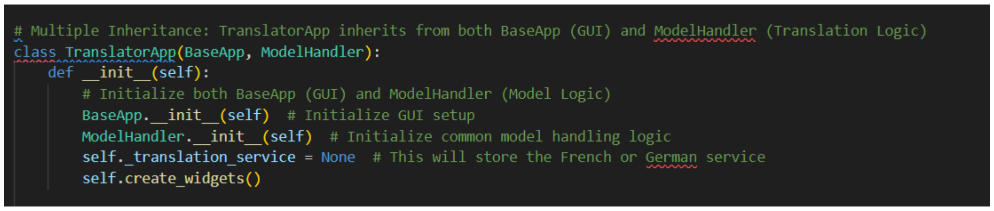
1. Encapsulation

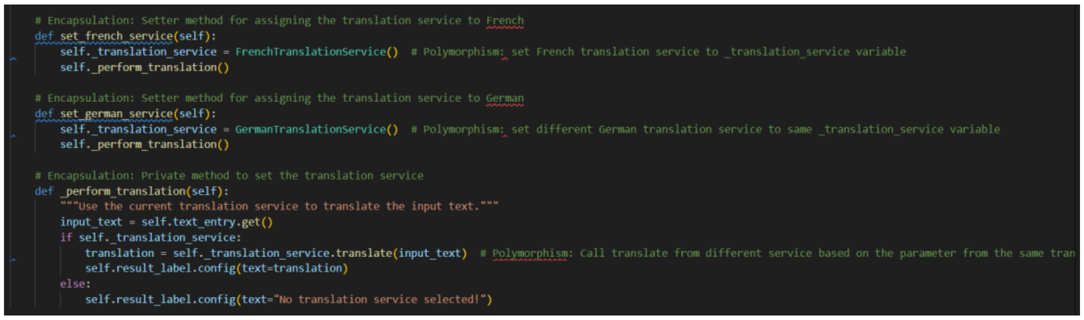
- **Class-Based Design**: The app separates functionality into different classes for modularity and clarity.

  - **ModelHandler** class: Encapsulates model loading and translation logic internally by restricting direct access to its internal attribute *_model* and providing controlled access through public methods (*"load_model"* and *"translate"*).


```python
from transformers import pipeline
from decorators import log_execution, log_result, error_handler

# Encapsulation: ModelHandler handles model loading and translation logic internally
class ModelHandler:
    def __init__(self):
        self._model = None  # Private attribute

    @log_execution  # Logging when the model is loaded
    @error_handler  # Handling any errors during model loading
    def load_model(self, model_name: str):
        if model_name == "translation_en_to_fr":
            self._model = pipeline("translation", model="Helsinki-NLP/opus-mt-en-fr")
        elif model_name == "translation_en_to_de":
            self._model = pipeline("translation", model="Helsinki-NLP/opus-mt-en-de")
        else:
            raise ValueError(f"Unsupported model: {model_name}")

    @log_execution  # Logging when translation is executed
    @log_result  # Logging the result of translation
    @error_handler  # Handling any errors during translation
    def translate(self, text: str) -> str:
        """Translate the given text using the loaded model."""
        if self._model:
            return self._model(text)[0]['translation_text']
        else:
            raise RuntimeError("No model loaded!")
```

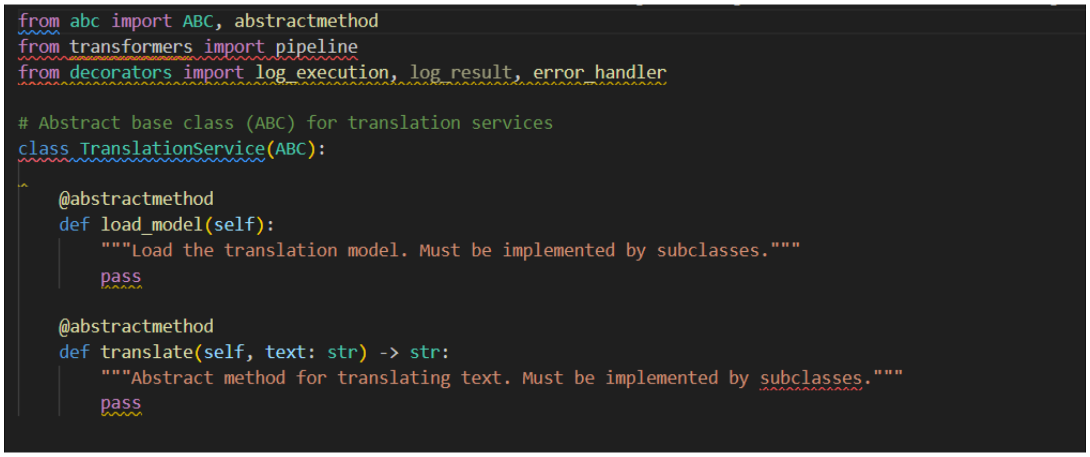
- **TranslatorApp** class:
  - We have used a private method "***_perform_translation***" to assign a new translation service to a private attribute named ***_translation_service***. This maintains encapsulation by preventing direct external modification of the translation service and allows to control how the service is set.
  - Users of the ***TranslatorApp*** class do not need to interact directly with the translation service. They interact via public methods "***set_french_service***" and ***"set_german_service"***, which encapsulate the process of selecting the appropriate service.


```python
# Multiple Inheritance: TranslatorApp inherits from both BaseApp (GUI) and ModelHandler (Translation Logic)
class TranslatorApp(BaseApp, ModelHandler):
    def __init__(self):
        # Initialize both BaseApp (GUI) and ModelHandler (Model Logic)
        BaseApp.__init__(self)  # Initialize GUI setup
        ModelHandler.__init__(self)  # Initialize common model handling logic
        self._translation_service = None  # This will store the French or German service
        self.create_widgets()
```


```python
    # Encapsulation: Setter method for assigning the translation service to French
    def set_french_service(self):
        self._translation_service = FrenchTranslationService()  # Polymorphism: set French translation service to _translation_service variable
        self._perform_translation()

    # Encapsulation: Setter method for assigning the translation service to German
    def set_german_service(self):
        self._translation_service = GermanTranslationService()  # Polymorphism: set different German translation service to same _translation_service variable
        self._perform_translation()

    # Encapsulation: Private method to set the translation service
    def _perform_translation(self):
        """Use the current translation service to translate the input text."""
        input_text = self.text_entry.get()
        if self._translation_service:
            translation = self._translation_service.translate(input_text)  # Polymorphism: Call translate from different service based on the parameter from the same tran
            self.result_label.config(text=translation)
        else:
            self.result_label.config(text="No translation service selected!")
```

2. Abstraction

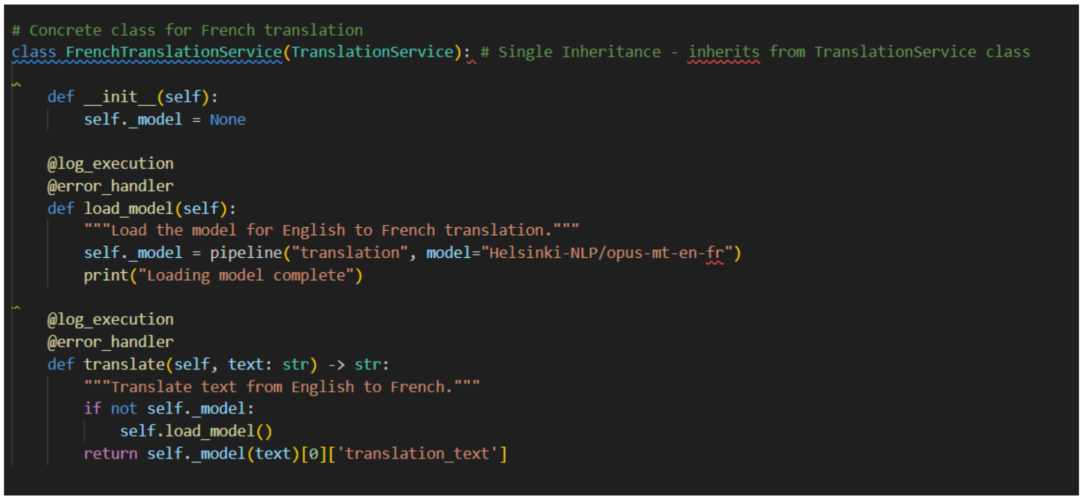
- The ***TranslationService*** class (in *translation_service.py*) is an **abstract base class** (ABC) that defines the structure of any translation service with the "***load_model***" and "***translate***" method as an abstract method. The subclasses **must override** these methods and provide their specific implementation.



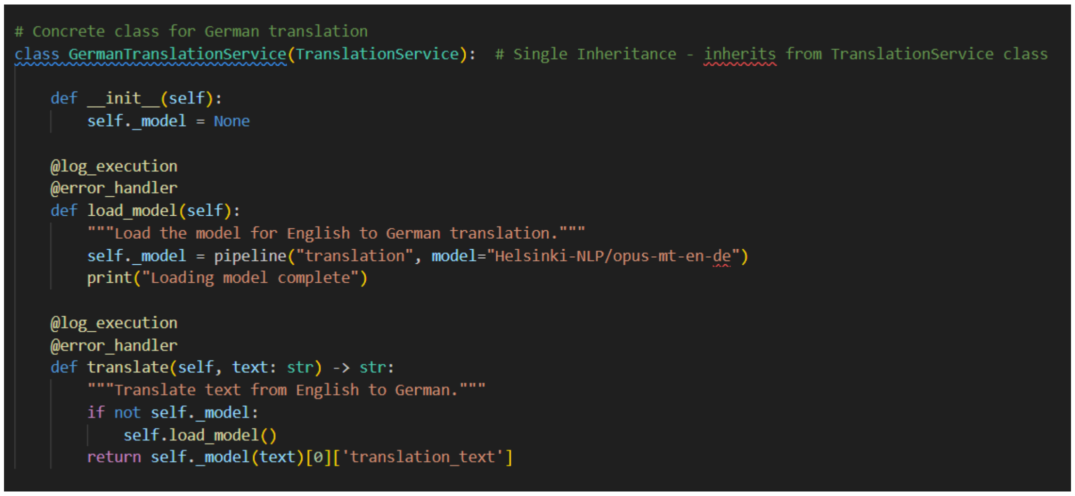
```python
from abc import ABC, abstractmethod
from transformers import pipeline
from decorators import log_execution, log_result, error_handler

# Abstract base class (ABC) for translation services
class TranslationService(ABC):

    @abstractmethod
    def load_model(self):
        """Load the translation model. Must be implemented by subclasses."""
        pass

    @abstractmethod
    def translate(self, text: str) -> str:
        """Abstract method for translating text. Must be implemented by subclasses."""
        pass
```

3. Method Overriding:

- Both ***FrenchTranslationService*** and ***GermanTranslationService*** override the ***translate*** method to provide specific French language and German language behavior, respectively. This is called method overriding.



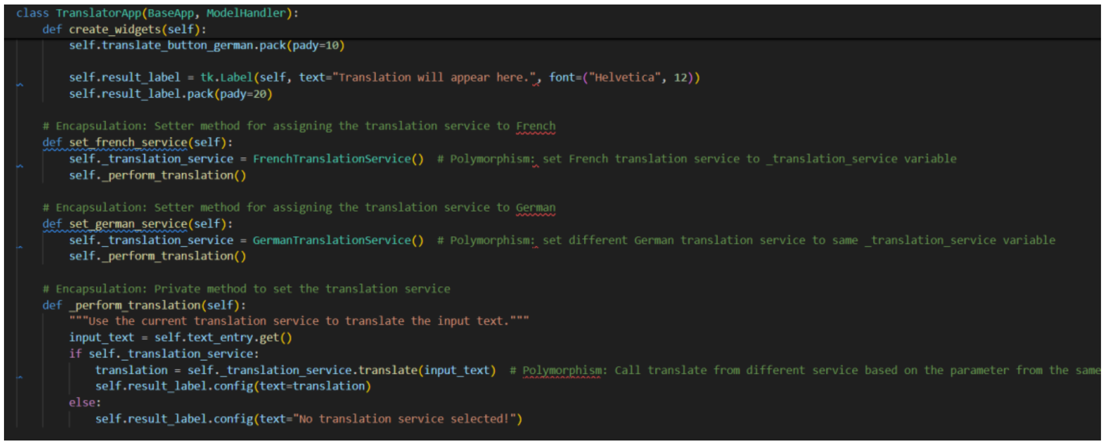
```python
# Concrete class for French translation
class FrenchTranslationService(TranslationService): # Single Inheritance - inherits from TranslationService class

    def __init__(self):
        self._model = None

    @log_execution
    @error_handler
    def load_model(self):
        """Load the model for English to French translation."""
        self._model = pipeline("translation", model="Helsinki-NLP/opus-mt-en-fr")
        print("Loading model complete")

    @log_execution
    @error_handler
    def translate(self, text: str) -> str:
        """Translate text from English to French."""
        if not self._model:
            self.load_model()
        return self._model(text)[0]['translation_text']
```



```python
# Concrete class for German translation
class GermanTranslationService(TranslationService):  # Single Inheritance - inherits from TranslationService class

    def __init__(self):
        self._model = None

    @log_execution
    @error_handler
    def load_model(self):
        """Load the model for English to German translation."""
        self._model = pipeline("translation", model="Helsinki-NLP/opus-mt-en-de")
        print("Loading model complete")

    @log_execution
    @error_handler
    def translate(self, text: str) -> str:
        """Translate text from English to German."""
        if not self._model:
            self.load_model()
        return self._model(text)[0]['translation_text']
```

4. Polymorphism:

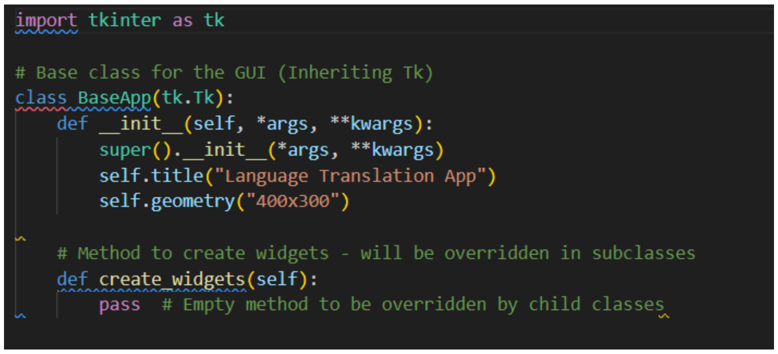
- In the ***TranslatorApp*** class (in *translator_app.py*), the same method ***translate*** is used regardless of which service (French or German) is currently selected. The application does not need to know the exact class. It only calls the translate method and polymorphism ensures that the correct method is executed based on the actual service assigned.



```python
class TranslatorApp(BaseApp, ModelHandler):
    def create_widgets(self):
        self.translate_button_german.pack(pady=10)

        self.result_label = tk.Label(self, text="Translation will appear here.", font=("Helvetica", 12))
        self.result_label.pack(pady=20)

    # Encapsulation: Setter method for assigning the translation service to French
    def set_french_service(self):
        self._translation_service = FrenchTranslationService()  # Polymorphism: set French translation service to _translation_service variable
        self._perform_translation()

    # Encapsulation: Setter method for assigning the translation service to German
    def set_german_service(self):
        self._translation_service = GermanTranslationService()  # Polymorphism: set different German translation service to same _translation_service variable
        self._perform_translation()

    # Encapsulation: Private method to set the translation service
    def _perform_translation(self):
        """Use the current translation service to translate the input text."""
        input_text = self.text_entry.get()
        if self._translation_service:
            translation = self._translation_service.translate(input_text)  # Polymorphism: Call translate from different service based on the parameter from the same
            self.result_label.config(text=translation)
        else:
            self.result_label.config(text="No translation service selected!")
```

5. Inheritance:

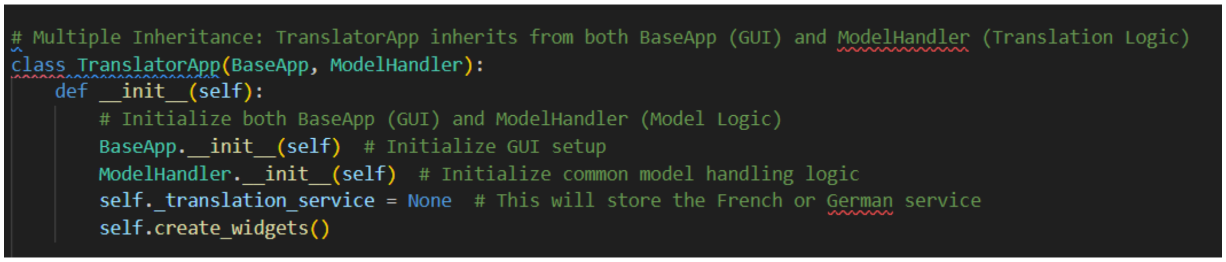
- **Single Inheritance:**
  - Base class for the GUI ***BaseApp*** class is inherited from the ***tk.Tk*** of *tkinter*. This is an example of single inheritance.



```python
import tkinter as tk

# Base class for the GUI (Inheriting Tk)
class BaseApp(tk.Tk):
    def __init__(self, *args, **kwargs):
        super().__init__(*args, **kwargs)
        self.title("Language Translation App")
        self.geometry("400x300")

    # Method to create widgets - will be overridden in subclasses
    def create_widgets(self):
        pass  # Empty method to be overridden by child classes
```

- **Multiple Inheritance:**

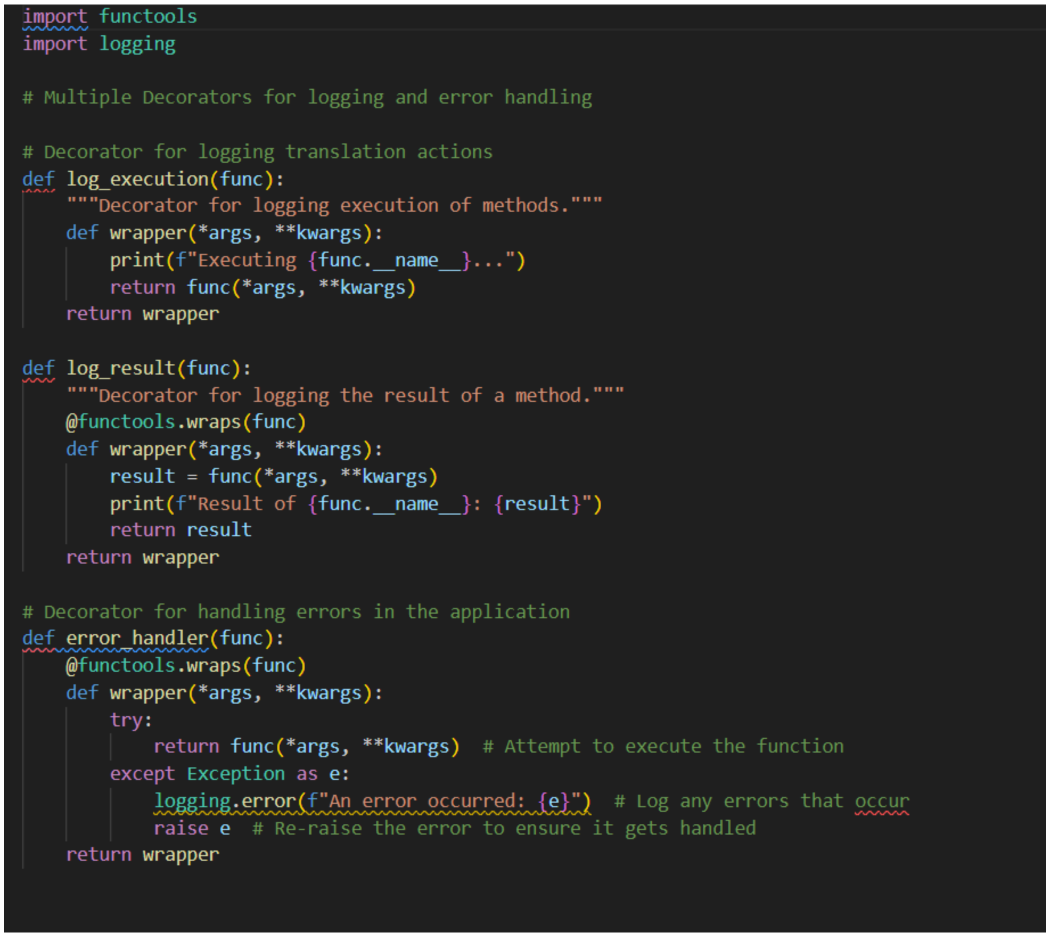
  - ***TranslatorApp*** class inherits from both ***BaseApp*** (for the GUI) and ***ModelHandler*** (for the model logic) classes, which is an example of multiple inheritance.



```python
# Multiple Inheritance: TranslatorApp inherits from both BaseApp (GUI) and ModelHandler (Translation Logic)
class TranslatorApp(BaseApp, ModelHandler):
    def __init__(self):
        # Initialize both BaseApp (GUI) and ModelHandler (Model Logic)
        BaseApp.__init__(self)  # Initialize GUI setup
        ModelHandler.__init__(self)  # Initialize common model handling logic
        self._translation_service = None  # This will store the French or German service
        self.create_widgets()
```

6. Decorators (Enhancing OOP):

- **log_translation, log_result,** and **handle_errors** decorators are used to enhance class methods with additional functionality such as logging for the execution of methods, the result of methods, and error handling respectively. This ensures that object methods are extended without modifying their core functionality.



```python
import functools
import logging

# Multiple Decorators for logging and error handling

# Decorator for logging translation actions
def log_execution(func):
    """Decorator for logging execution of methods."""
    def wrapper(*args, **kwargs):
        print(f"Executing {func.__name__}...")
        return func(*args, **kwargs)
    return wrapper

def log_result(func):
    """Decorator for logging the result of a method."""
    @functools.wraps(func)
    def wrapper(*args, **kwargs):
        result = func(*args, **kwargs)
        print(f"Result of {func.__name__}: {result}")
        return result
    return wrapper

# Decorator for handling errors in the application
def error_handler(func):
    @functools.wraps(func)
    def wrapper(*args, **kwargs):
        try:
            return func(*args, **kwargs)  # Attempt to execute the function
        except Exception as e:
            logging.error(f"An error occurred: {e}")  # Log any errors that occur
            raise e  # Re-raise the error to ensure it gets handled
    return wrapper
```

➢ File Structure:

```
├── translator_app.py        # Main entry point of the application
├── base_app.py              # Base class for the GUI (Inheriting Tk)
├── translation_service.py   # Contains the Abstract base class (ABC) for translation services and
                             its concrete subclasses for French and German Translation Services
├── decorators.py            # Custom decorators for logging and error handling
├── model_handler.py         # For handling model loading and translation logic internally
```

➢ Conclusion

This Translator Application demonstrates a clean, OOP-based design while utilizing modern NLP models from Hugging Face. By adhering to OOP principles, the code is modular, maintainable, and easy to extend. By simply changing the translation service class or adding new features, the app can be scaled to more complex use cases.

Question 2: - [Synthia Islam, Hussein Salami]

Create a simple "side-scrolling" 2D game using Pygame. The game should allow the player to control a character with the ability to run, jump, and shoot projectiles. The game should have enemies, collectibles, and 3 levels. It should also have a scoring system, health, and lives.

The game should include the following, but not limited to:

- Player class (movements, speed, jump, health, lives) - Methods
- Projectile Class (movements, speed, damage) – Methods
- Enemy Class (……………….) – Methods
- Collectible Class (health boost, extra life, etc.,)
- Level Design (3 Levels), Add boss enemy at the end.
- A Scoring system based on enemies defeated, and collectibles collected, health bar for players, and enemies.
- Implement a game over screen with the option to restart.

Bonus: Create a dynamic camera that follows the players smoothly.

You have three game ideas, select one and implement the above requirements.

- A game with human-like characters (hero, enemy)
- A game with an animal (Hero) and human characters (Enemy).
- A tank-based game navigating through a battlefield to engage with enemy tanks or something.

ANSWER:

(Codes and other files are included under the folder "pygame-question-2", attached in the main zip file; the main app file is in the "side-scrolling-pygame.py" file, which needs to be run)

- **Precondition:**

  "**pygame**" needs to be installed through the command:

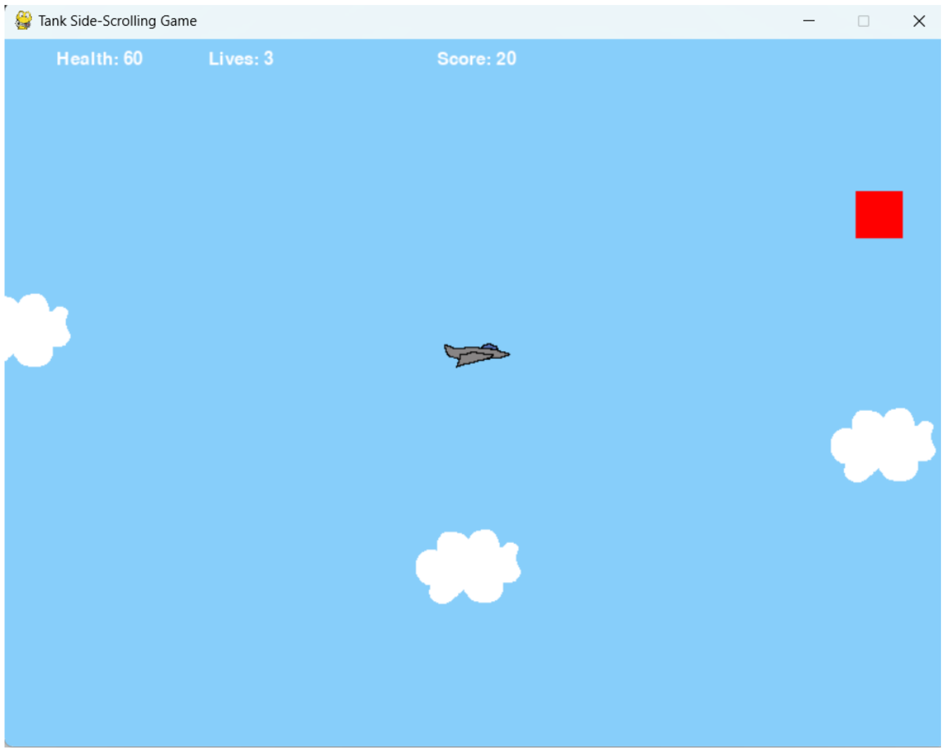
  pip install pygame

- **How to run the game:**
  - Open a terminal or command prompt in the project directory.
  - Run the following command:

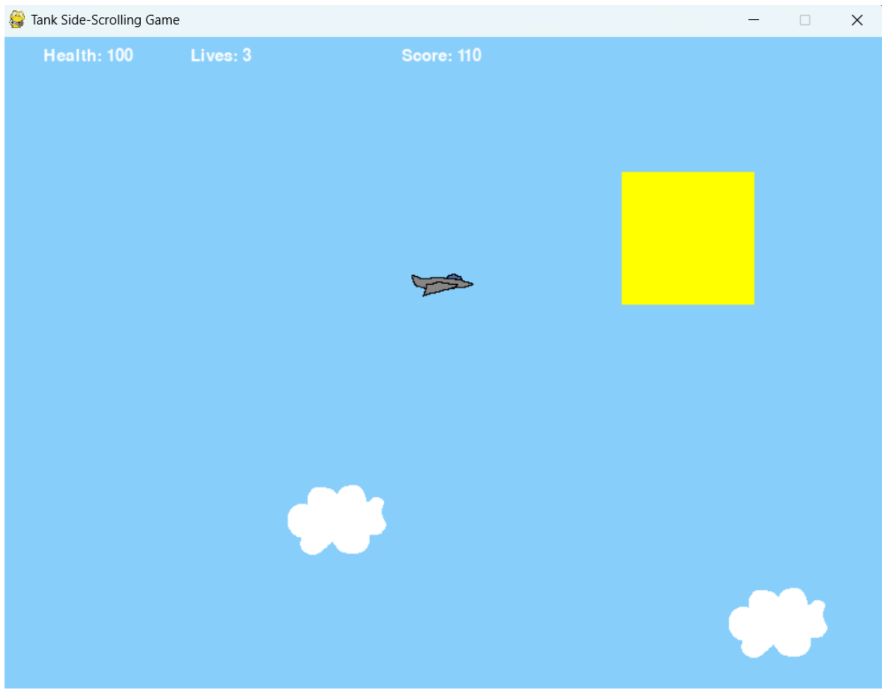
  python side-scrolling-pygame.py

- **Theme:** For this game, we have chosen the third theme, but in the air battle, to engage with the enemies for uniqueness.

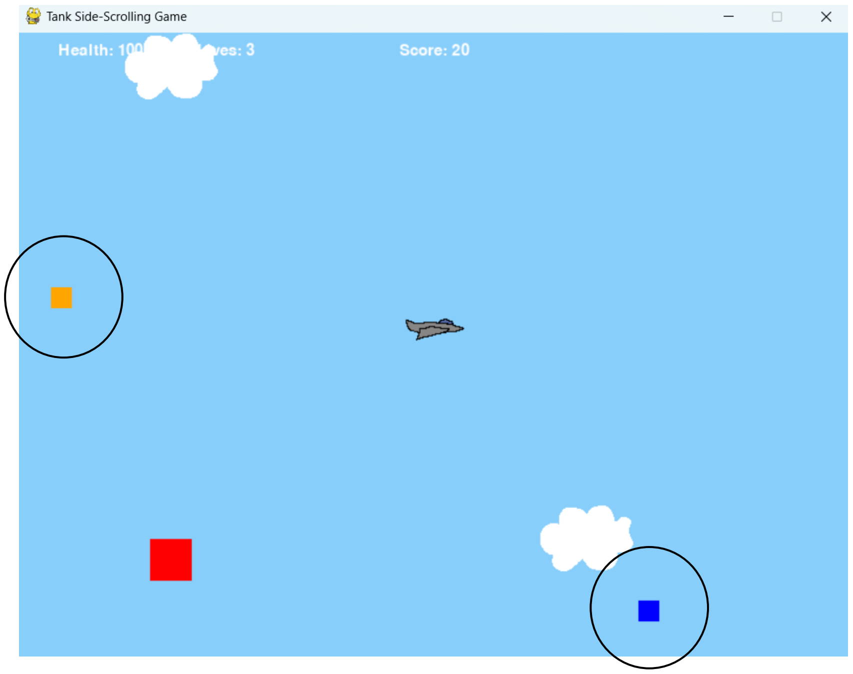
- **Representation:** Here, we have shown different visuals for different entities.
    - The airplane represents the Player.
    - Red boxes represent the Enemies.



- The yellow box represents the Boss Enemy.



- The blue box (highlighted in the image below with a circle) represents the Extra Life increased by 1 if the Player can collect it.
- The orange box (highlighted in the image below with a circle) represents the Health Booster, increased by 20 if the Player can collect it.



- **Game Rules:**
  - **Navigation:**

    i) Up and Down Arrows must be pressed to move the Player to the top and bottom positions respectively.

    ii) Space Bar needs to be pressed to shoot bullets to kill the enemies.

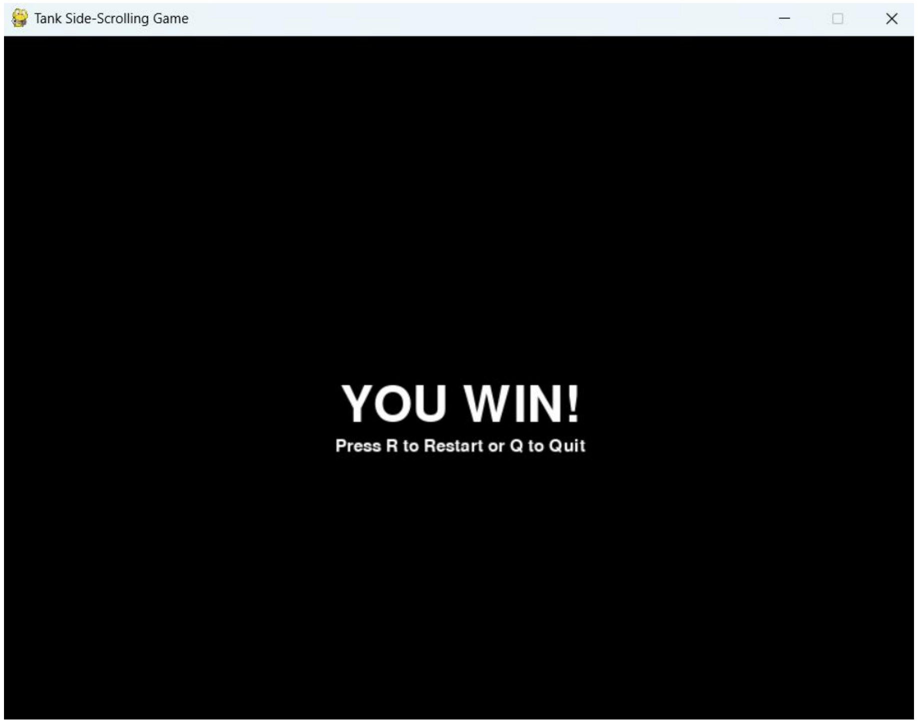
    iii) R (case insensitive): After finishing the game, **R** needs to be pressed to restart the game.

    iv) Q (case insensitive): After finishing the game, **Q** must be pressed to quit.
  - **Features:**

    i) Levels: There are 3 levels.

    - **Level 1**:
      - The player needs to kill **5 enemies** to complete the level.
      - The player needs to shoot **once** every time to kill each enemy.
      - Collectibles (Health booster and extra life) will spawn one by one.
    - **Level 2**:
      - The player needs to kill **3 enemies** to complete the level.

    - The player needs to shoot **twice** every time to kill each enemy.
    - Collectibles (Health booster and extra life) will spawn one by one.
- **Level 3**:
    - The player needs to kill the Boss Enemy **to win the Game**.
    - The player needs to shoot **thrice** to kill the Boss Enemy.



ii) Enemy:

- Enemies will be spawned one after another in Level 1 and 2.
- The Boss Enemy will appear in Level 3. It will reappear again and again until it's killed completely or all the lives of the Player are lost.
- The strength and size of enemies will be increased level-wise. The enemy size and strength will be higher in level 2 than in level 1. The ultimate Boss Enemy in the 3rd level will be the biggest and strongest than the other enemies in Level 1 and 2.

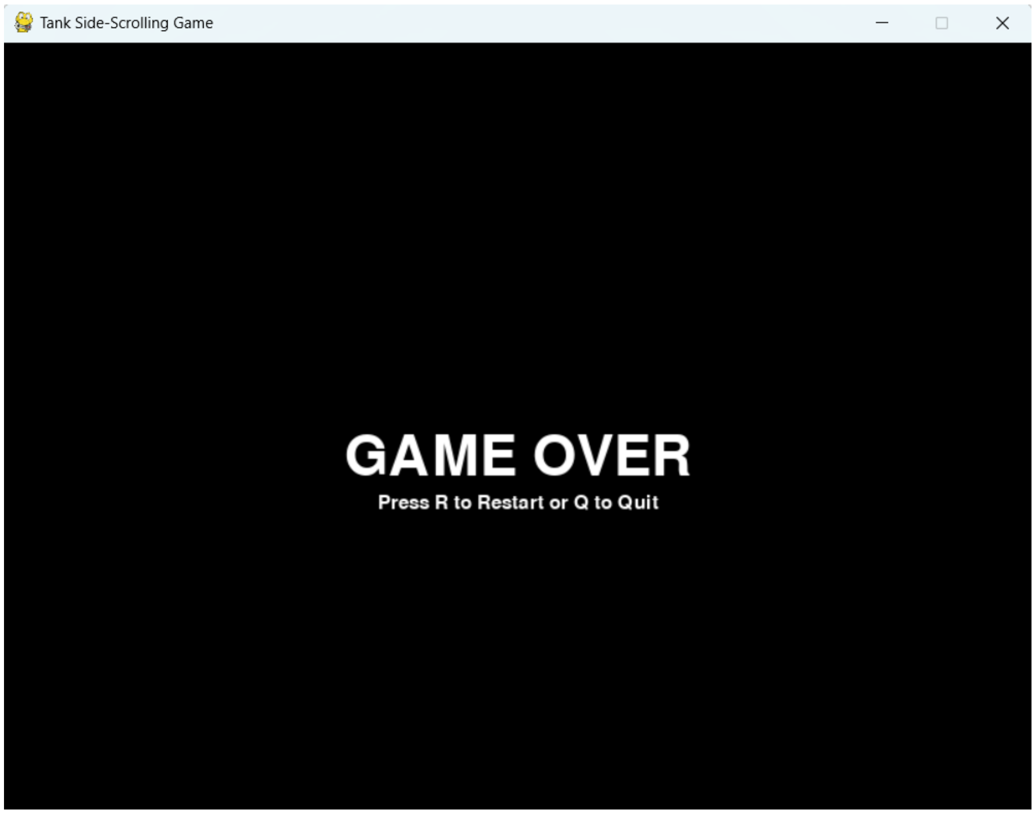
iii) Health:

- Health is shown as 100 for each life of the player.
- Enemy health is not shown on the screen for avoiding confusion.
    - In Level 1, enemy health is 1.
    - In Level 2, enemy health is 2.
    - In Level 3, enemy health is 3.

- If a collision happens between the player and the enemy, then:
    - Enemy health will decrease by 1.
    - In Level 1, the player's health will be decreased by 20.
    - In Level 2, the player's health will be decreased by 30.
    - In Level 3, the player's health will be decreased by 40.

iv) Lives:

- There will be 3 lives for players by default.
- When health becomes 0, lives will be decreased by 1. **When all lives are gone, then the game will be over.**


Tank Side-Scrolling Game

GAME OVER

Press R to Restart or Q to Quit

v) Collectibles:

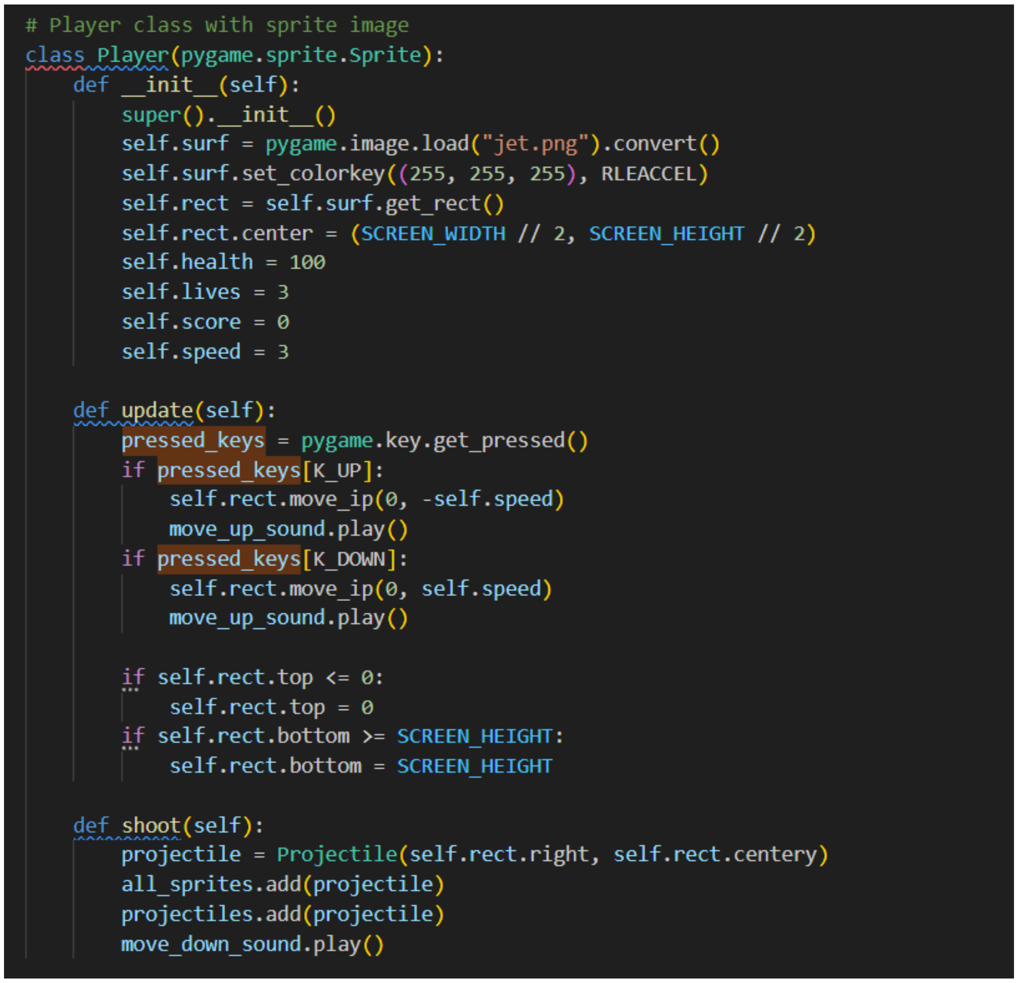
- **Health Booster:** A health booster will come after killing a few enemies in Level 1 and 2.
- **Extra Life:** An extra life will come after killing several enemies in Level 1 and 2.

vi) Scores:

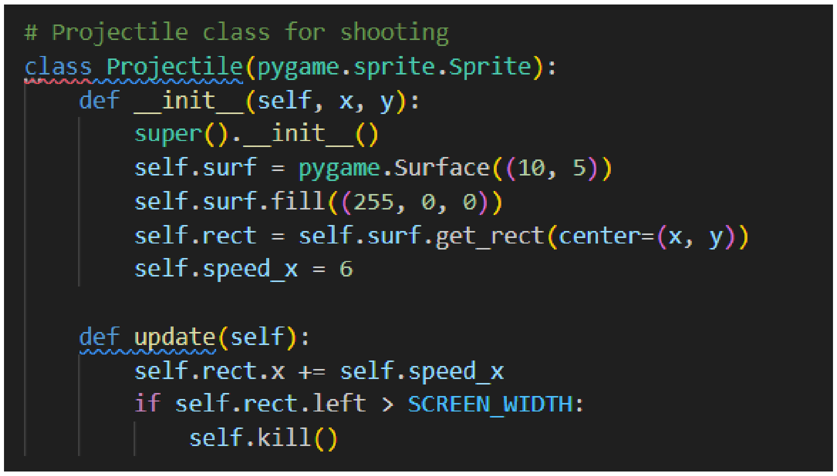
- After killing each enemy, the score will be increased by 10 points.

➢ Implementation:

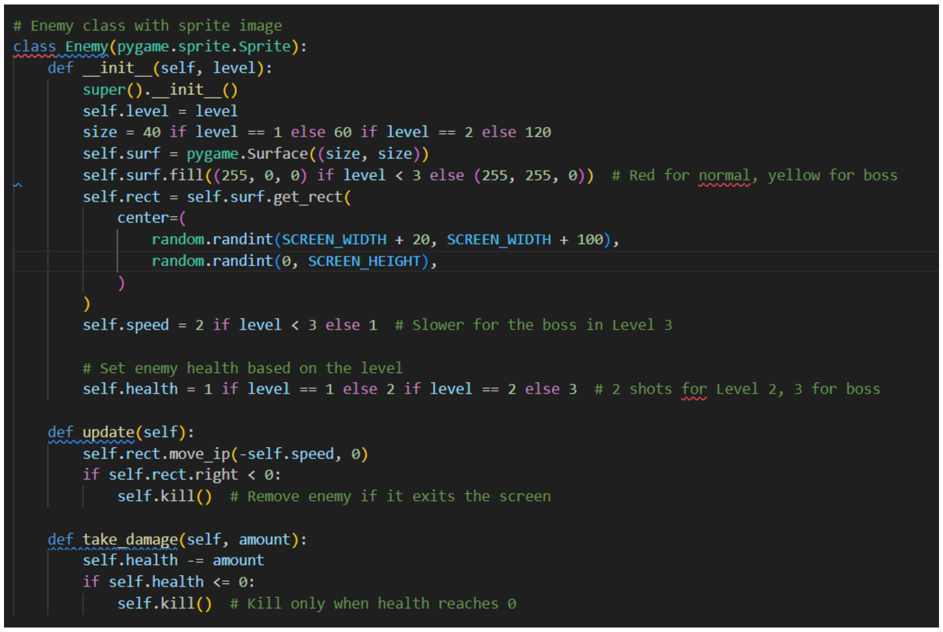
- We implemented Player, Enemy, Projectile, Collectible classes as well as Cloud class for background.



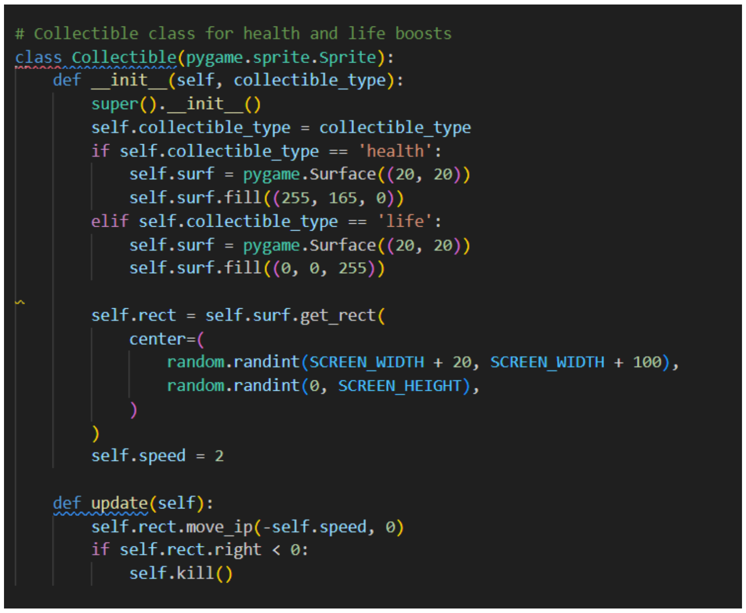
```python
# Player class with sprite image
class Player(pygame.sprite.Sprite):
    def __init__(self):
        super().__init__()
        self.surf = pygame.image.load("jet.png").convert()
        self.surf.set_colorkey((255, 255, 255), RLEACCEL)
        self.rect = self.surf.get_rect()
        self.rect.center = (SCREEN_WIDTH // 2, SCREEN_HEIGHT // 2)
        self.health = 100
        self.lives = 3
        self.score = 0
        self.speed = 3

    def update(self):
        pressed_keys = pygame.key.get_pressed()
        if pressed_keys[K_UP]:
            self.rect.move_ip(0, -self.speed)
            move_up_sound.play()
        if pressed_keys[K_DOWN]:
            self.rect.move_ip(0, self.speed)
            move_up_sound.play()

        if self.rect.top <= 0:
            self.rect.top = 0
        if self.rect.bottom >= SCREEN_HEIGHT:
            self.rect.bottom = SCREEN_HEIGHT

    def shoot(self):
        projectile = Projectile(self.rect.right, self.rect.centery)
        all_sprites.add(projectile)
        projectiles.add(projectile)
        move_down_sound.play()
```



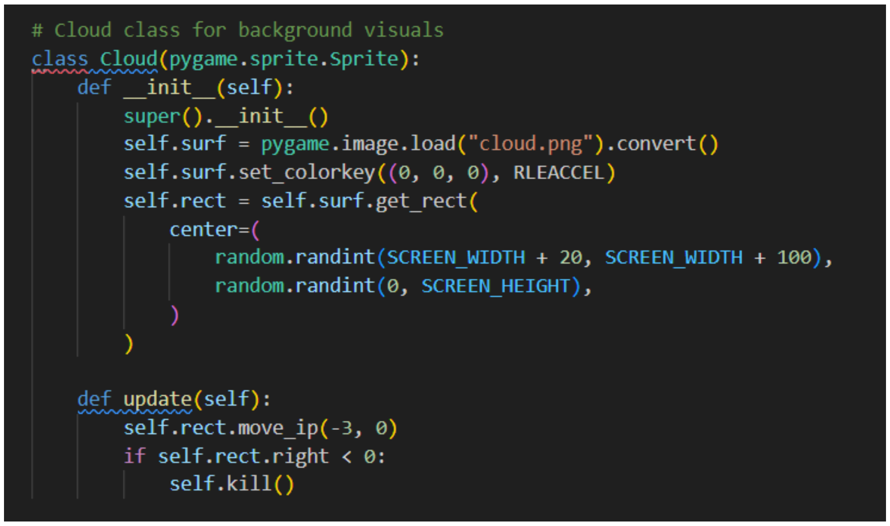
```python
# Projectile class for shooting
class Projectile(pygame.sprite.Sprite):
    def __init__(self, x, y):
        super().__init__()
        self.surf = pygame.Surface((10, 5))
        self.surf.fill((255, 0, 0))
        self.rect = self.surf.get_rect(center=(x, y))
        self.speed_x = 6

    def update(self):
        self.rect.x += self.speed_x
        if self.rect.left > SCREEN_WIDTH:
            self.kill()
```



```python
# Enemy class with sprite image
class Enemy(pygame.sprite.Sprite):
    def __init__(self, level):
        super().__init__()
        self.level = level
        size = 40 if level == 1 else 60 if level == 2 else 120
        self.surf = pygame.Surface((size, size))
        self.surf.fill((255, 0, 0) if level < 3 else (255, 255, 0))  # Red for normal, yellow for boss
        self.rect = self.surf.get_rect(
            center=(
                random.randint(SCREEN_WIDTH + 20, SCREEN_WIDTH + 100),
                random.randint(0, SCREEN_HEIGHT),
            )
        )
        self.speed = 2 if level < 3 else 1  # Slower for the boss in Level 3

        # Set enemy health based on the level
        self.health = 1 if level == 1 else 2 if level == 2 else 3  # 2 shots for Level 2, 3 for boss

    def update(self):
        self.rect.move_ip(-self.speed, 0)
        if self.rect.right < 0:
            self.kill()  # Remove enemy if it exits the screen

    def take_damage(self, amount):
        self.health -= amount
        if self.health <= 0:
            self.kill()  # Kill only when health reaches 0
```

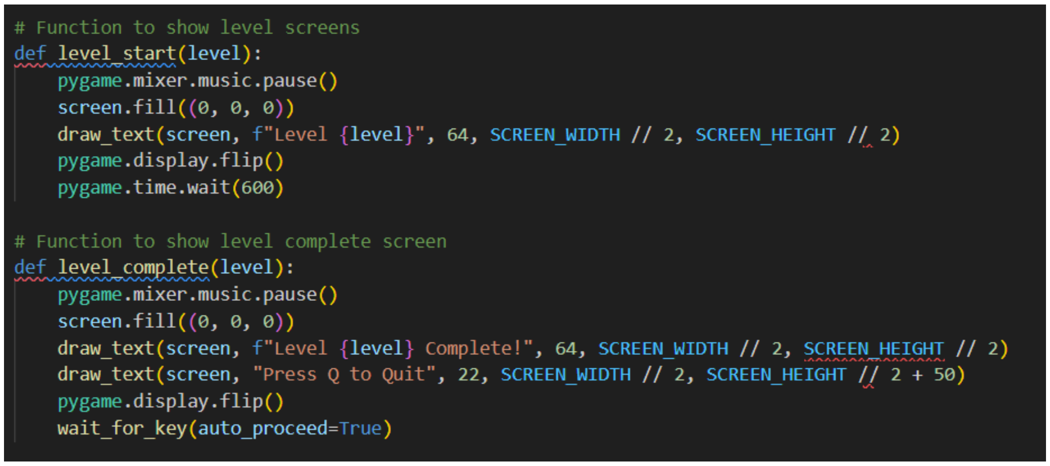


```python
# Collectible class for health and life boosts
class Collectible(pygame.sprite.Sprite):
    def __init__(self, collectible_type):
        super().__init__()
        self.collectible_type = collectible_type
        if self.collectible_type == 'health':
            self.surf = pygame.Surface((20, 20))
            self.surf.fill((255, 165, 0))
        elif self.collectible_type == 'life':
            self.surf = pygame.Surface((20, 20))
            self.surf.fill((0, 0, 255))

        self.rect = self.surf.get_rect(
            center=(
                random.randint(SCREEN_WIDTH + 20, SCREEN_WIDTH + 100),
                random.randint(0, SCREEN_HEIGHT),
            )
        )
        self.speed = 2

    def update(self):
        self.rect.move_ip(-self.speed, 0)
        if self.rect.right < 0:
            self.kill()
```



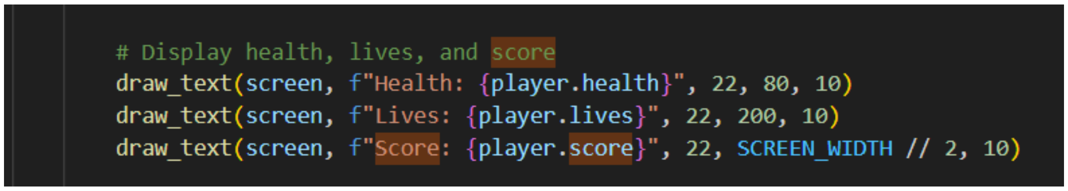
```python
# Cloud class for background visuals
class Cloud(pygame.sprite.Sprite):
    def __init__(self):
        super().__init__()
        self.surf = pygame.image.load("cloud.png").convert()
        self.surf.set_colorkey((0, 0, 0), RLEACCEL)
        self.rect = self.surf.get_rect(
            center=(
                random.randint(SCREEN_WIDTH + 20, SCREEN_WIDTH + 100),
                random.randint(0, SCREEN_HEIGHT),
            )
        )

    def update(self):
        self.rect.move_ip(-3, 0)
        if self.rect.right < 0:
            self.kill()
```

- We implemented Level Complete and Next Level (1,2,3) Start Screen:



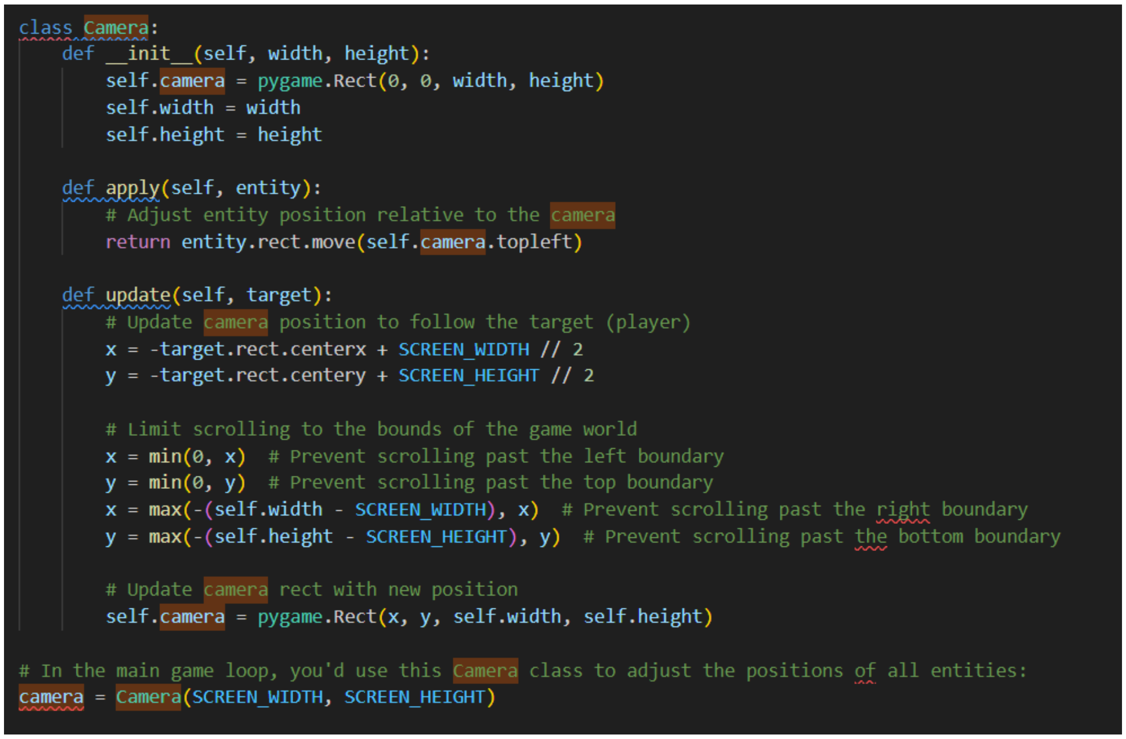
```python
# Function to show level screens
def level_start(level):
    pygame.mixer.music.pause()
    screen.fill((0, 0, 0))
    draw_text(screen, f"Level {level}", 64, SCREEN_WIDTH // 2, SCREEN_HEIGHT // 2)
    pygame.display.flip()
    pygame.time.wait(600)

# Function to show level complete screen
def level_complete(level):
    pygame.mixer.music.pause()
    screen.fill((0, 0, 0))
    draw_text(screen, f"Level {level} Complete!", 64, SCREEN_WIDTH // 2, SCREEN_HEIGHT // 2)
    draw_text(screen, "Press Q to Quit", 22, SCREEN_WIDTH // 2, SCREEN_HEIGHT // 2 + 50)
    pygame.display.flip()
    wait_for_key(auto_proceed=True)
```

- We implemented a Scoring system based on enemies defeated and collectibles collected, and for simplicity, health is shown on the screen for players only. Health is counted for enemies in the backend, which is not shown on the screen to avoid confusion.



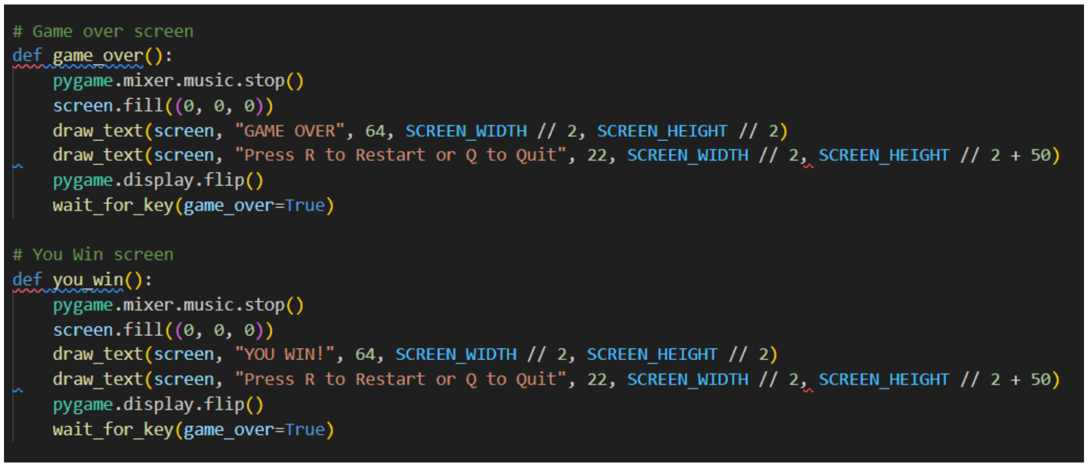
```python
        # Display health, lives, and score
        draw_text(screen, f"Health: {player.health}", 22, 80, 10)
        draw_text(screen, f"Lives: {player.lives}", 22, 200, 10)
        draw_text(screen, f"Score: {player.score}", 22, SCREEN_WIDTH // 2, 10)
```

- We implemented the bonus part, which is a dynamic camera that follows the players smoothly.



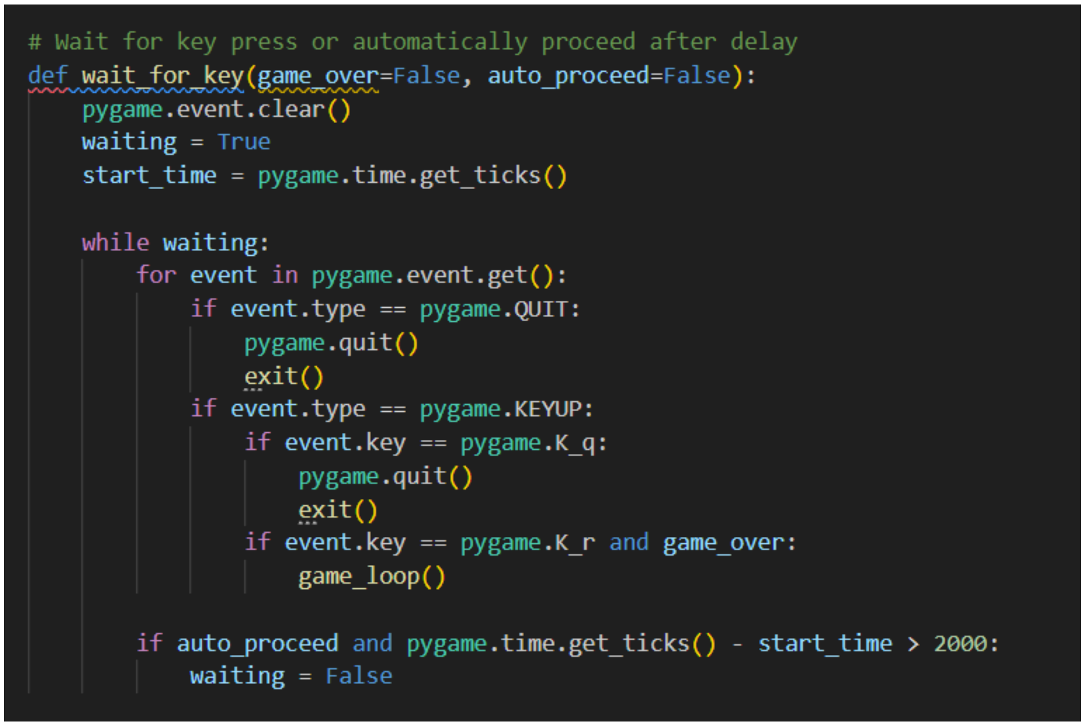
```python
class Camera:
    def __init__(self, width, height):
        self.camera = pygame.Rect(0, 0, width, height)
        self.width = width
        self.height = height

    def apply(self, entity):
        # Adjust entity position relative to the camera
        return entity.rect.move(self.camera.topleft)

    def update(self, target):
        # Update camera position to follow the target (player)
        x = -target.rect.centerx + SCREEN_WIDTH // 2
        y = -target.rect.centery + SCREEN_HEIGHT // 2

        # Limit scrolling to the bounds of the game world
        x = min(0, x)  # Prevent scrolling past the left boundary
        y = min(0, y)  # Prevent scrolling past the top boundary
        x = max(-(self.width - SCREEN_WIDTH), x)  # Prevent scrolling past the right boundary
        y = max(-(self.height - SCREEN_HEIGHT), y)  # Prevent scrolling past the bottom boundary

        # Update camera rect with new position
        self.camera = pygame.Rect(x, y, self.width, self.height)

# In the main game loop, you'd use this Camera class to adjust the positions of all entities:
camera = Camera(SCREEN_WIDTH, SCREEN_HEIGHT)
```

- We implemented a Game Over and Win screen with the option of Restart or Quit the game by pressing a specific key.



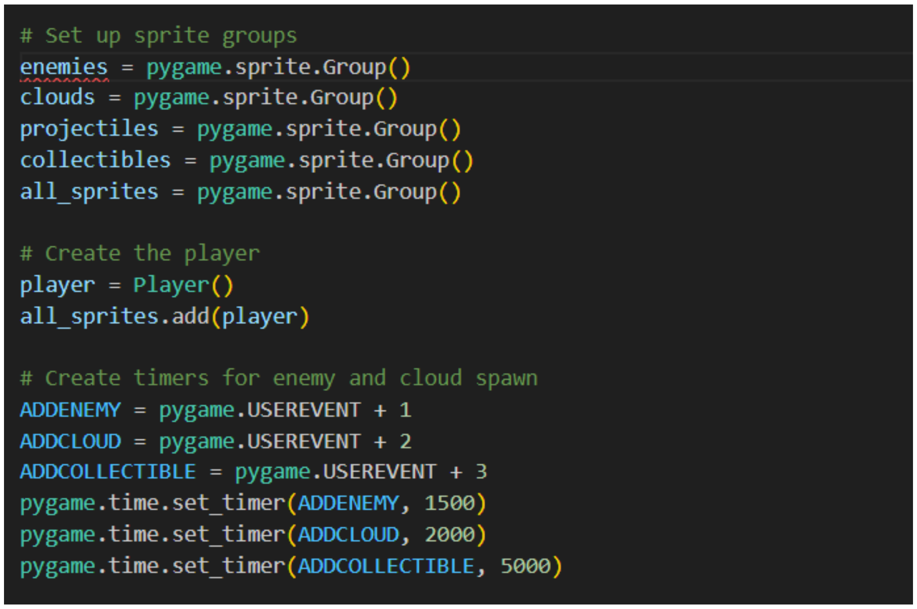
```python
# Game over screen
def game_over():
    pygame.mixer.music.stop()
    screen.fill((0, 0, 0))
    draw_text(screen, "GAME OVER", 64, SCREEN_WIDTH // 2, SCREEN_HEIGHT // 2)
    draw_text(screen, "Press R to Restart or Q to Quit", 22, SCREEN_WIDTH // 2, SCREEN_HEIGHT // 2 + 50)
    pygame.display.flip()
    wait_for_key(game_over=True)

# You Win screen
def you_win():
    pygame.mixer.music.stop()
    screen.fill((0, 0, 0))
    draw_text(screen, "YOU WIN!", 64, SCREEN_WIDTH // 2, SCREEN_HEIGHT // 2)
    draw_text(screen, "Press R to Restart or Q to Quit", 22, SCREEN_WIDTH // 2, SCREEN_HEIGHT // 2 + 50)
    pygame.display.flip()
    wait_for_key(game_over=True)
```

- We implemented wait for key event for restart or quit and automatically transition to next level after completing one level:



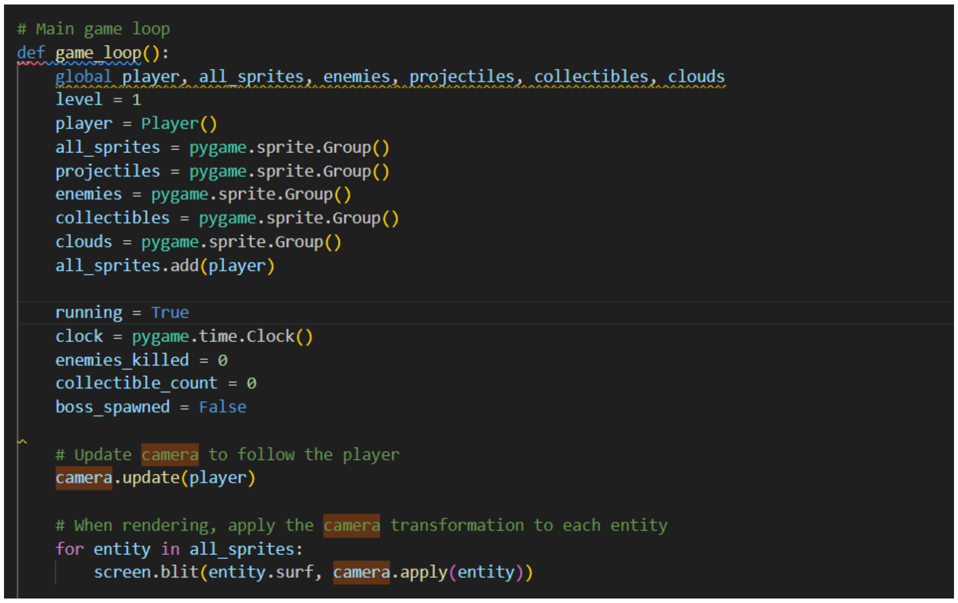
```python
# Wait for key press or automatically proceed after delay
def wait_for_key(game_over=False, auto_proceed=False):
    pygame.event.clear()
    waiting = True
    start_time = pygame.time.get_ticks()

    while waiting:
        for event in pygame.event.get():
            if event.type == pygame.QUIT:
                pygame.quit()
                exit()
            if event.type == pygame.KEYUP:
                if event.key == pygame.K_q:
                    pygame.quit()
                    exit()
                if event.key == pygame.K_r and game_over:
                    game_loop()

        if auto_proceed and pygame.time.get_ticks() - start_time > 2000:
            waiting = False
```

- We setup the sprite groups for all the features and set the timers.

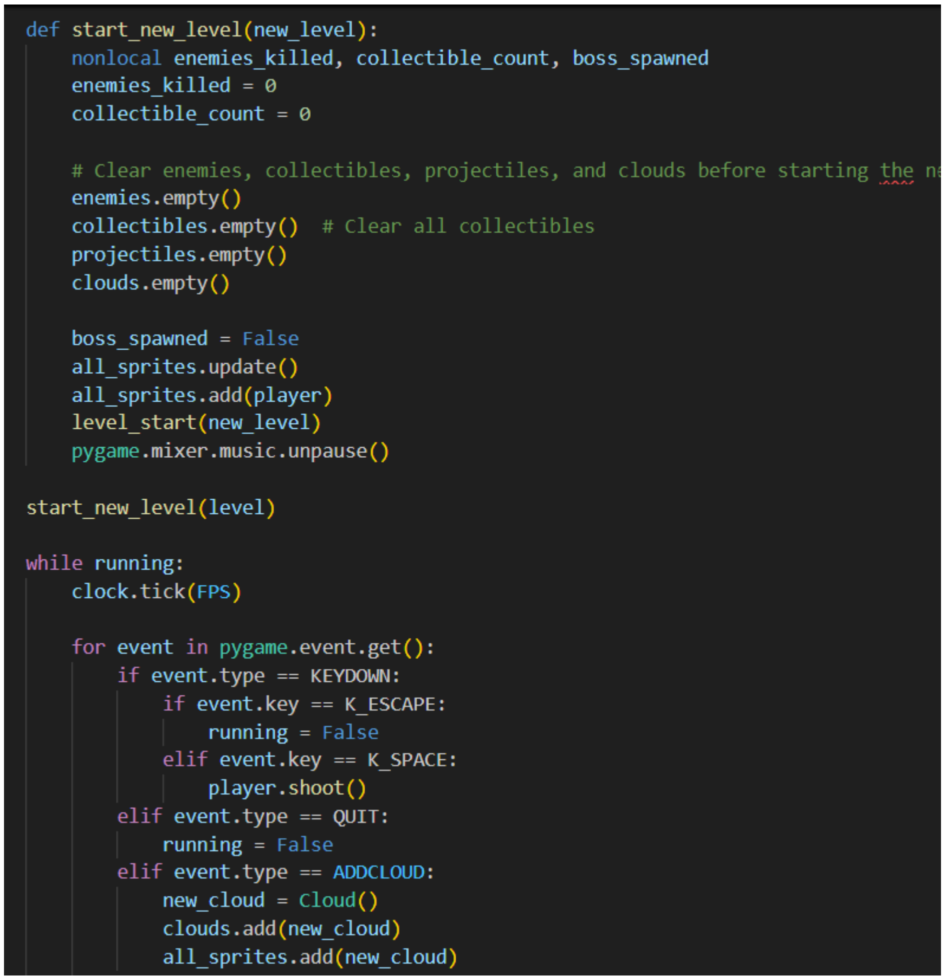


```python
# Set up sprite groups
enemies = pygame.sprite.Group()
clouds = pygame.sprite.Group()
projectiles = pygame.sprite.Group()
collectibles = pygame.sprite.Group()
all_sprites = pygame.sprite.Group()

# Create the player
player = Player()
all_sprites.add(player)

# Create timers for enemy and cloud spawn
ADDENEMY = pygame.USEREVENT + 1
ADDCLOUD = pygame.USEREVENT + 2
ADDCOLLECTIBLE = pygame.USEREVENT + 3
pygame.time.set_timer(ADDENEMY, 1500)
pygame.time.set_timer(ADDCLOUD, 2000)
pygame.time.set_timer(ADDCOLLECTIBLE, 5000)
```

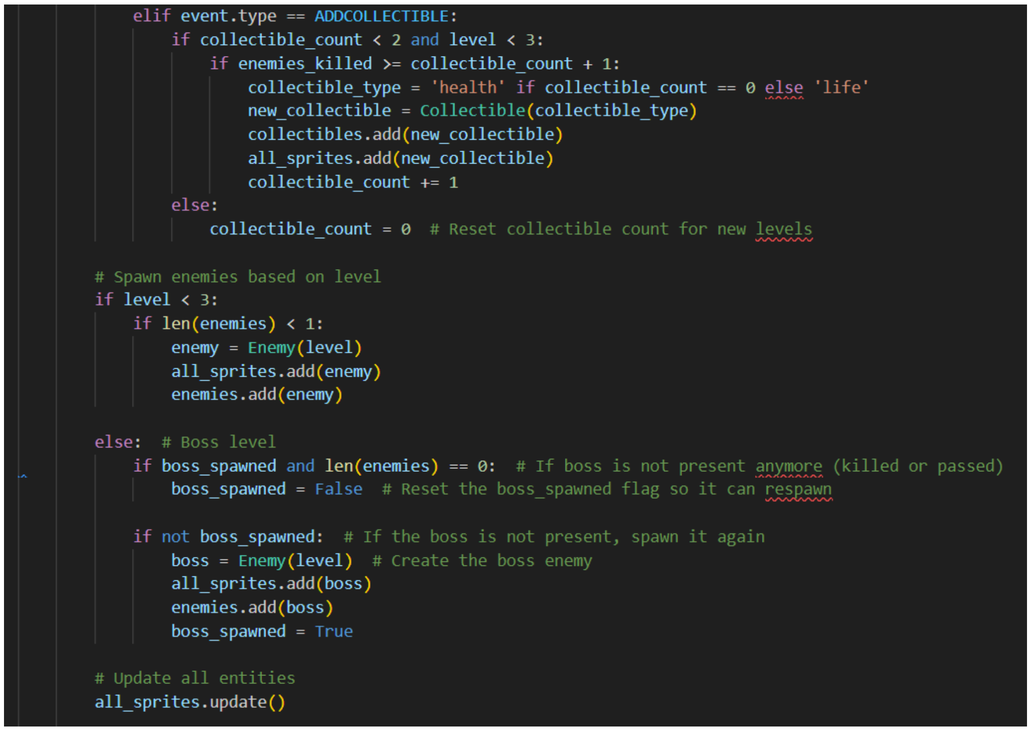
- We implemented the main game loop:



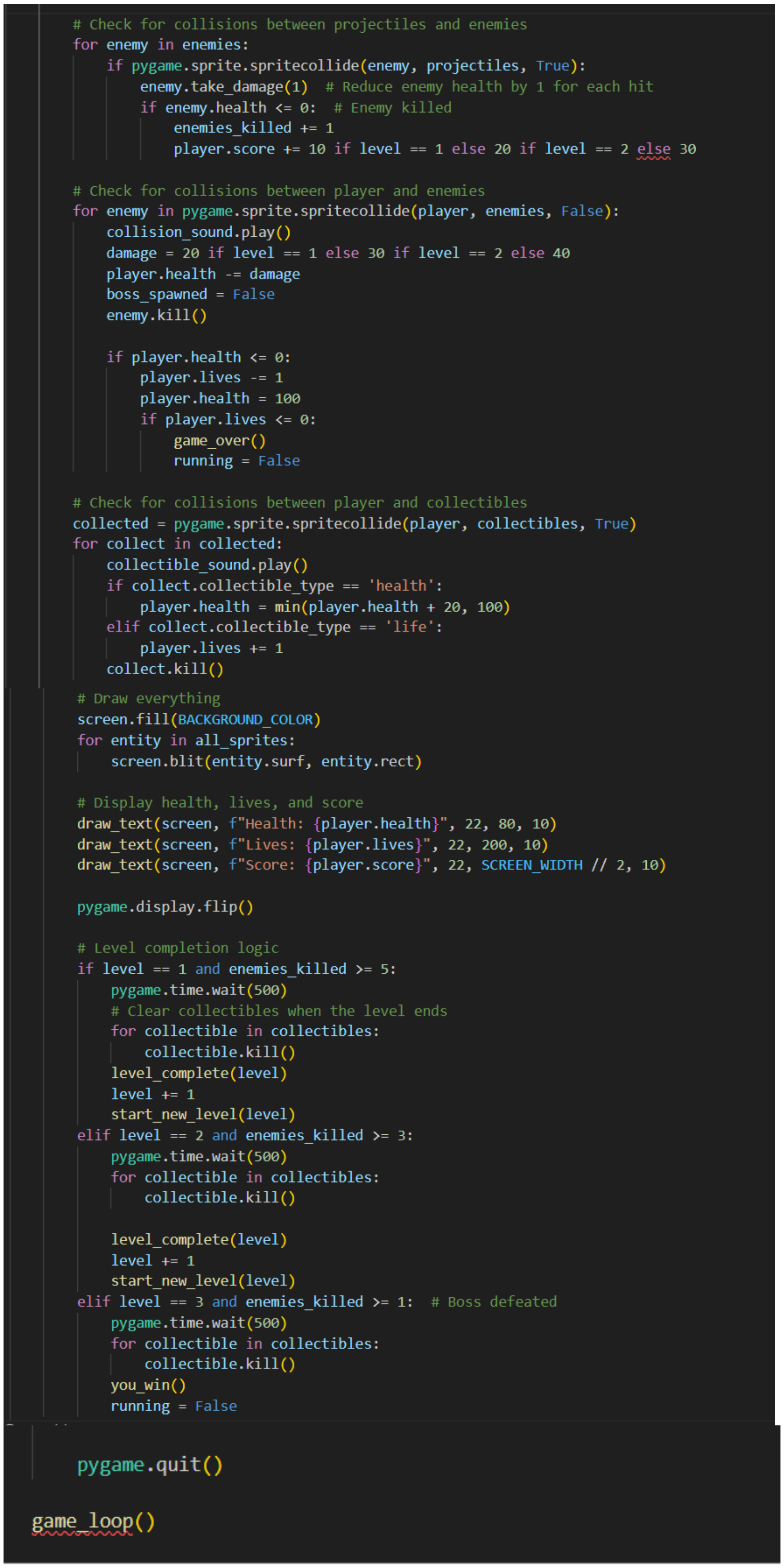
```python
# Main game loop
def game_loop():
    global player, all_sprites, enemies, projectiles, collectibles, clouds
    level = 1
    player = Player()
    all_sprites = pygame.sprite.Group()
    projectiles = pygame.sprite.Group()
    enemies = pygame.sprite.Group()
    collectibles = pygame.sprite.Group()
    clouds = pygame.sprite.Group()
    all_sprites.add(player)

    running = True
    clock = pygame.time.Clock()
    enemies_killed = 0
    collectible_count = 0
    boss_spawned = False

    # Update camera to follow the player
    camera.update(player)

    # When rendering, apply the camera transformation to each entity
    for entity in all_sprites:
        screen.blit(entity.surf, camera.apply(entity))
```



```python
def start_new_level(new_level):
    nonlocal enemies_killed, collectible_count, boss_spawned
    enemies_killed = 0
    collectible_count = 0

    # Clear enemies, collectibles, projectiles, and clouds before starting the n
    enemies.empty()
    collectibles.empty()  # Clear all collectibles
    projectiles.empty()
    clouds.empty()

    boss_spawned = False
    all_sprites.update()
    all_sprites.add(player)
    level_start(new_level)
    pygame.mixer.music.unpause()

start_new_level(level)

while running:
    clock.tick(FPS)

    for event in pygame.event.get():
        if event.type == KEYDOWN:
            if event.key == K_ESCAPE:
                running = False
            elif event.key == K_SPACE:
                player.shoot()
        elif event.type == QUIT:
            running = False
        elif event.type == ADDCLOUD:
            new_cloud = Cloud()
            clouds.add(new_cloud)
            all_sprites.add(new_cloud)
```



```python
        elif event.type == ADDCOLLECTIBLE:
            if collectible_count < 2 and level < 3:
                if enemies_killed >= collectible_count + 1:
                    collectible_type = 'health' if collectible_count == 0 else 'life'
                    new_collectible = Collectible(collectible_type)
                    collectibles.add(new_collectible)
                    all_sprites.add(new_collectible)
                    collectible_count += 1
            else:
                collectible_count = 0  # Reset collectible count for new levels

    # Spawn enemies based on level
    if level < 3:
        if len(enemies) < 1:
            enemy = Enemy(level)
            all_sprites.add(enemy)
            enemies.add(enemy)

    else:  # Boss level
        if boss_spawned and len(enemies) == 0:  # If boss is not present anymore (killed or passed)
            boss_spawned = False  # Reset the boss_spawned flag so it can respawn

        if not boss_spawned:  # If the boss is not present, spawn it again
            boss = Enemy(level)  # Create the boss enemy
            all_sprites.add(boss)
            enemies.add(boss)
            boss_spawned = True

    # Update all entities
    all_sprites.update()
```



```python
        # Check for collisions between projectiles and enemies
        for enemy in enemies:
            if pygame.sprite.spritecollide(enemy, projectiles, True):
                enemy.take_damage(1)  # Reduce enemy health by 1 for each hit
                if enemy.health <= 0:  # Enemy killed
                    enemies_killed += 1
                    player.score += 10 if level == 1 else 20 if level == 2 else 30

        # Check for collisions between player and enemies
        for enemy in pygame.sprite.spritecollide(player, enemies, False):
            collision_sound.play()
            damage = 20 if level == 1 else 30 if level == 2 else 40
            player.health -= damage
            boss_spawned = False
            enemy.kill()

            if player.health <= 0:
                player.lives -= 1
                player.health = 100
                if player.lives <= 0:
                    game_over()
                    running = False

        # Check for collisions between player and collectibles
        collected = pygame.sprite.spritecollide(player, collectibles, True)
        for collect in collected:
            collectible_sound.play()
            if collect.collectible_type == 'health':
                player.health = min(player.health + 20, 100)
            elif collect.collectible_type == 'life':
                player.lives += 1
            collect.kill()
        # Draw everything
        screen.fill(BACKGROUND_COLOR)
        for entity in all_sprites:
            screen.blit(entity.surf, entity.rect)

        # Display health, lives, and score
        draw_text(screen, f"Health: {player.health}", 22, 80, 10)
        draw_text(screen, f"Lives: {player.lives}", 22, 200, 10)
        draw_text(screen, f"Score: {player.score}", 22, SCREEN_WIDTH // 2, 10)

        pygame.display.flip()

        # Level completion logic
        if level == 1 and enemies_killed >= 5:
            pygame.time.wait(500)
            # Clear collectibles when the level ends
            for collectible in collectibles:
                collectible.kill()
            level_complete(level)
            level += 1
            start_new_level(level)
        elif level == 2 and enemies_killed >= 3:
            pygame.time.wait(500)
            for collectible in collectibles:
                collectible.kill()

            level_complete(level)
            level += 1
            start_new_level(level)
        elif level == 3 and enemies_killed >= 1:  # Boss defeated
            pygame.time.wait(500)
            for collectible in collectibles:
                collectible.kill()
            you_win()
            running = False

    pygame.quit()

game_loop()
```

Question 3: - [Synthia Islam]

Welcome to the final task of this assignment. You are required to create a GitHub repository and add all your group mates to it (make sure to keep it public, not private). You should do this before you start the assignment.

All the answers and contributions should be recorded in GitHub till you submit the assignment.

Answer:

GitHub public repository is created for the team, and members are properly added:

Link: https://github.com/cas119/HIT137-software-now-cas119/

Folder: assignment-3