OOP cousework

Domas Stagniūnas EAF – 23

# Introduction

## The Goal of the Coursework

The primary goal of this coursework is to design and implement a Hangman game application using Python and the `pygame` library. The project aims to demonstrate an understanding of basic programming concepts, user interaction through a graphical interface.

## The Topic

The topic of this coursework is the creation of a word-guessing game, Hangman, where players attempt to guess a word by suggesting letters within a certain number of guesses.

## What is Your Application?

The application is a Hangman game developed in Python. It features graphical user interface elements made with `pygame` and uses JSON files for saving historical score data.

## How to Run the Program?

To run the Hangman game, ensure Python and `pygame` are installed on your system, and the main file is executed in visual code studio.

## How to Use the Program?

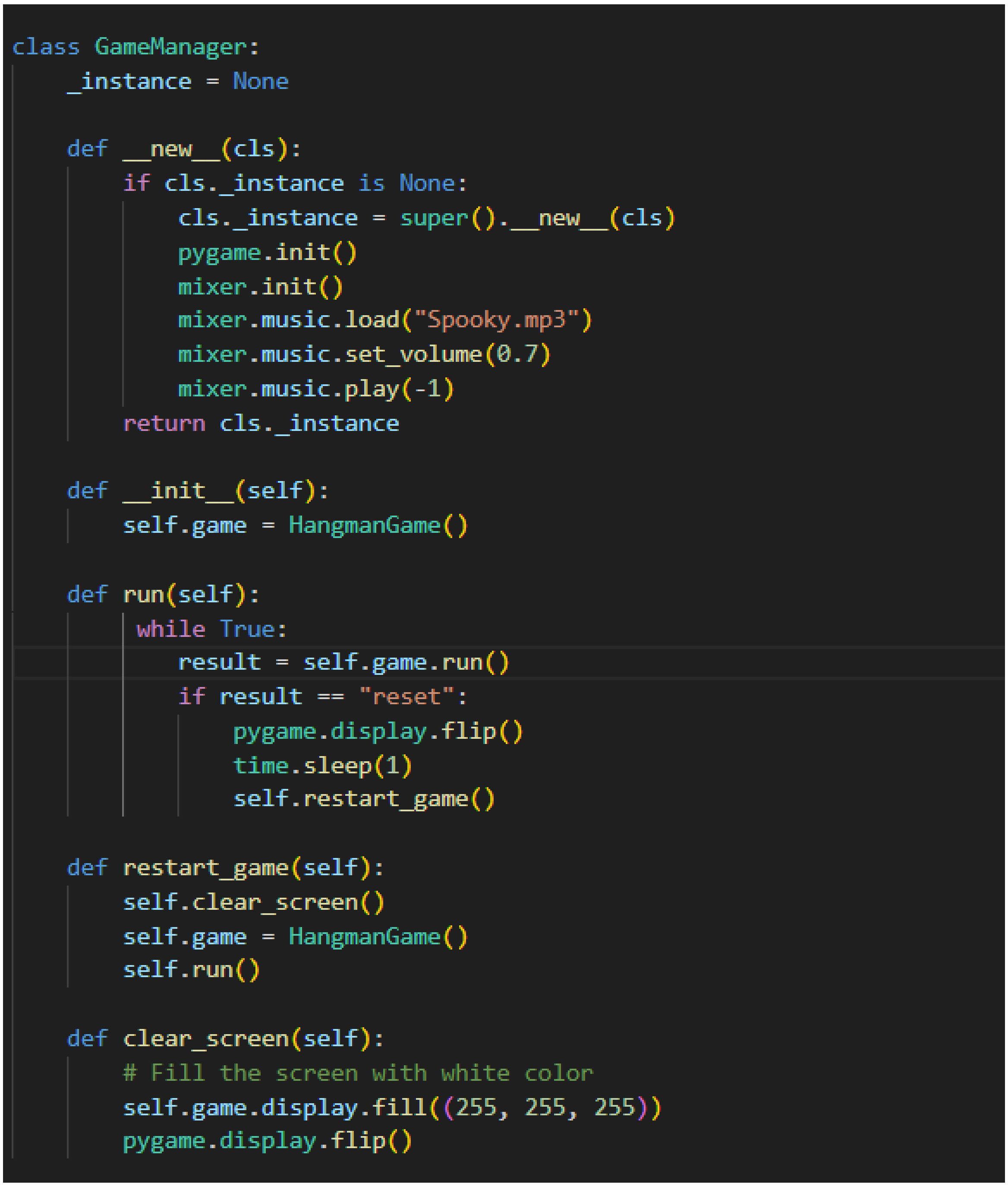
Execute the main code. The game window will display blank space representing the letters of a randomly chosen word. Players press on the billboard to enter letters using the keyboard. Correct guesses reveal the letter in the word, while incorrect guesses result losing a life. The game continues until the word is completed or life number reaches zero.

# Body/Analysis

## Program Implementation and Analysis

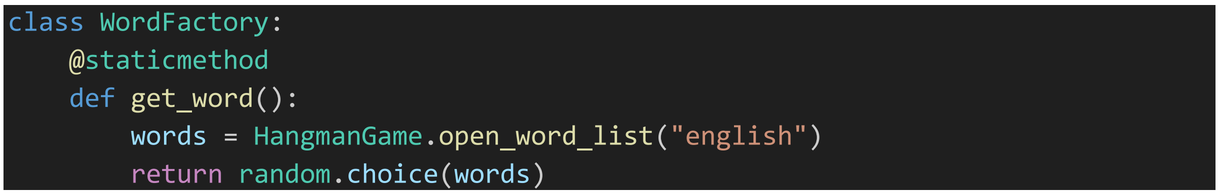
The Hangman game is implemented in Python using the `pygame` library, adhering to the principles of event-driven programming. The game meets its functional requirements as follows:

Usage of Singelton



```python
class GameManager:
    _instance = None

    def __new__(cls):
        if cls._instance is None:
            cls._instance = super().__new__(cls)
            pygame.init()
            mixer.init()
            mixer.music.load("Spooky.mp3")
            mixer.music.set_volume(0.7)
            mixer.music.play(-1)
        return cls._instance

    def __init__(self):
        self.game = HangmanGame()

    def run(self):
         while True:
            result = self.game.run()
            if result == "reset":
                pygame.display.flip()
                time.sleep(1)
                self.restart_game()

    def restart_game(self):
        self.clear_screen()
        self.game = HangmanGame()
        self.run()

    def clear_screen(self):
        # Fill the screen with white color
        self.game.display.fill((255, 255, 255))
        pygame.display.flip()
```

Usage of decorator



```python
class WordFactory:
    @staticmethod
    def get_word():
        words = HangmanGame.open_word_list("english")
        return random.choice(words)
```

Abstraction

The class provides methods like toggle_language, reveal_letter, and run, which allow interaction with the game without exposing the internal details of how these operations are performed.

```
    def reveal_letter(self, letter):
        # Abstracting the letter reveal functionality
        if letter in self.missed_letters:
            return True
        revealed = False
        for i, char in enumerate(self.random_word):
            if char == letter:
                self.hidden_word[i] = letter
                revealed = True
        if not revealed:
            self.missed_letters.append(letter)
        return revealed


    def run(self):
        # Abstracting the game loop
        while True:
            self.display.fill((255, 255, 255))
            self.display.blit(self.bg, (0, 0))
            self.display.blit(self.billboard, (100, 318))
            # Handle events and game logic
            # ...
```

Encapsulation

Attributes: The attributes such as language, words, random_word, separated_word, hidden_word, missed_letters, lifes, game_over, display, font, input_box, color_inactive, color_active, bg, etc. are encapsulated within the HangmanGame class.

Methods: Methods like toggle_language, open_word_list, separate_word, hide_word, reveal_letter, is_word_revealed, display_hidden_word, display_hearts, display_text, save_score_history, load_score, and run operate on the data encapsulated within the class.

```
class HangmanGame:
    def __init__(self):
        # Initialization code
        self.language = "Lietuvių"
        self.words = self.open_word_list(self.language)
        # Other attributes
        # ...

    def toggle_language(self):
        # Method to toggle language
        # ...

    def open_word_list(self, value):
        # Method to open word list
        # ...

    def separate_word(self, word):
        # Method to separate word
        # ...

    def hide_word(self, letters):
        # Method to hide word
        # ...

    # Other methods
    # ...
```

# Results

- Successfully implemented a playable Hangman game with graphical user interface using Python and `pygame`.

- Faced challenges with implementing efficient word selection and managing game states.

- Integrating sound effects and managing game over scenarios required careful event handling and state checks.

# Conclusions

## Key Findings and Outcomes

- The coursework successfully resulted in creating fully functional Hangman game.

- The usage of `pygame` for graphical output and event handling.

- I learned how to make basic 2d game with interactive interface.

## Result om my program

- The outcame of my work is simple 2d game.

## Future Prospects

- The application can be expanded with additional features such as multiplayer options, more complex word lists, and different difficulty levels.

- Potential integration with an online database for global high score tracking and community engagement.