Виселица

Введение

1. **Общая структура игры "Виселица"**:
   - **Описание игры**: "Виселица" — это игра на отгадывание слов. Игроки пытаются угадать слово, предлагая буквы. За каждую неправильную попытку к рисунку виселицы добавляется элемент, и если весь рисунок завершен до того, как слово будет отгадано, игрок проигрывает.
   - **Основные компоненты**:
     - **Выбор слова**: Случайное слово выбирается из списка.
     - **Попытки угадать буквы**: Игрок вводит буквы, чтобы попытаться угадать слово.
     - **Отслеживание угаданных букв**: Игра отслеживает, какие буквы уже были угаданы.
     - **Отображение состояния игры**: Состояние слова (угадываемое слово с подчеркиваниями для неугаданных букв) и виселицы.

Основные классы и их роли

2. **Класс `WordLoader`**:
   - **Назначение**: Этот класс загружает слова из файла и выбирает случайное слово.
   - **Методы**:
     - `__init__(self, words_file="words.txt")`: Конструктор, загружает слова из файла.
     - `load_words(self, words_file)`: Считывает слова из файла и возвращает список слов.
     - `get_random_word(self)`: Возвращает случайное слово из списка слов.
   - **Пример кода**:

```
class WordLoader:
    def __init__(self, words_file="words.txt"):
        self.words = self.load_words(words_file)

    def load_words(self, words_file):
        with open(words_file, "r") as f:
            return [word.strip() for word in f.readlines()]

    def get_random_word(self):
        return random.choice(self.words)
```

   - **Объяснение**:
     - Конструктор загружает слова из файла при создании экземпляра класса.
     - Метод `load_words` считывает слова из файла и возвращает их в виде списка.
     - Метод `get_random_word` выбирает и возвращает случайное слово из списка.
3. **Класс `HangmanState`**:
   - **Назначение**: Этот класс отслеживает состояние игры, включая текущее слово, количество оставшихся попыток и угаданные буквы.
   - **Методы**:
     - `__init__(self, word, max_attempts)`: Инициализирует состояние игры.
     - `guess_letter(self, letter)`: Обрабатывает угадывание буквы, обновляет состояние игры.
     - `get_game_state(self)`: Возвращает текущее состояние игры.
     - `is_won(self)`: Проверяет, выиграна ли игра.
     - `is_lost(self)`: Проверяет, проиграна ли игра.
   - **Пример кода**:

```
class HangmanState:
    def __init__(self, word, max_attempts):
        self.word = word
        self.max_attempts = max_attempts
```

```
        self.attempts = max_attempts
        self.guessed_letters = []

    def guess_letter(self, letter):
        if letter in self.word and letter not in self.guessed_letters:
            self.guessed_letters.append(letter)
            return True
        elif letter not in self.word and letter not in self.guessed_letters:
            self.guessed_letters.append(letter)
            self.attempts -= 1
            return False
        else:
            return None

    def get_game_state(self):
        word_state = ["_" if letter not in self.guessed_letters else letter for letter in
self.word]
        return {
            "word": " ".join(word_state),
            "attempts": self.attempts,
            "guessed_letters": " ".join(sorted(self.guessed_letters)),
        }

    def is_won(self):
        return all([letter in self.guessed_letters for letter in self.word])

    def is_lost(self):
        return self.attempts == 0
```

- **Объяснение**:
  - Конструктор инициализирует игру, устанавливает максимальное количество попыток и текущее слово.
  - Метод `guess_letter` обрабатывает ввод пользователя, проверяя, присутствует ли буква в слове, и обновляет количество попыток и список угаданных букв.
  - Метод `get_game_state` возвращает текущее состояние игры, включая текущее состояние слова, количество оставшихся попыток и угаданные буквы.
  - Методы `is_won` и `is_lost` проверяют, выиграна или проиграна игра.

4. **Класс** `HangmanGraphics`:
   - **Назначение**: Этот класс отвечает за отображение псевдографики виселицы в зависимости от количества оставшихся попыток.
   - **Содержимое**:
     - `stages`: Список строк, представляющих различные стадии рисунка виселицы.
     - `display(attempts_left)`: Статический метод, который принимает количество оставшихся попыток и отображает соответствующую стадию виселицы.
   - **Пример кода**:

```
class HangmanGraphics:
    stages = [
        """
           ____
           |   |
               |
               |
               |
               |
        ========""",
        """
           ____
           |   |
           O   |
               |
               |
               |
```

```
        ========""",
        """
           ____
           |   |
           0   |
           |   |
               |
               |
        ========""",
        """
           ____
           |   |
           0   |
          /|   |
               |
               |
        ========""",
        """
           ____
           |   |
           0   |
          /|\  |
               |
               |
        ========""",
        """
           ____
           |   |
           0   |
          /|\  |
          /    |
               |
        ========""",
        """
           ____
           |   |
           0   |
          /|\  |
          / \  |
               |
        ========"""
    ]

    @staticmethod
    def display(attempts_left):
        index = max(0, len(HangmanGraphics.stages) - (attempts_left + 1))
        print(HangmanGraphics.stages[index])
```

- **Объяснение**:
  - Список `stages` содержит различные стадии рисунка виселицы, каждая из которых представляет состояние игры.
  - Метод `display` выбирает и отображает соответствующую стадию в зависимости от количества оставшихся попыток.

5. **Класс** `HangmanGame`:
   - **Назначение**: Этот класс представляет саму игру и объединяет загрузку слов и отслеживание состояния игры.
   - **Методы**:
     - `__init__(self, words_file="words.txt", max_attempts=7)`: Конструктор, который загружает слова и устанавливает максимальное количество попыток.
     - `reset(self)`: Сбрасывает состояние игры, выбирая новое случайное слово.
     - `guess_letter(self, letter)`: Делегирует обработку угадывания буквы методу `guess_letter` класса `HangmanState`.
     - `get_game_state(self)`: Возвращает текущее состояние игры, делегируя вызов методу `get_game_state` класса `HangmanState`.

- `is_won(self)`: Проверяет, выиграна ли игра.
- `is_lost(self)`: Проверяет, проиграна ли игра.

- **Пример кода**:

```
class HangmanGame:
    def __init__(self, words_file="words.txt", max_attempts=7):
        self.word_loader = WordLoader(words_file)
        self.max_attempts = max_attempts
        self.reset()

    def reset(self):
        word = self.word_loader.get_random_word()
        self.state = HangmanState(word, self.max_attempts)

    def guess_letter(self, letter):
        return self.state.guess_letter(letter)

    def get_game_state(self):
        return self.state.get_game_state()

    def is_won(self):
        return self.state.is_won()

    def is_lost(self):
        return self.state.is_lost()
```

- **Объяснение**:
  - Конструктор инициализирует игру, загружает слова и устанавливает максимальное количество попыток.
  - Метод `reset` сбрасывает состояние игры, выбирая новое случайное слово.
  - Методы `guess_letter`, `get_game_state`, `is_won` и `is_lost` делегируют работу соответствующим методам класса `HangmanState`.

6. **Класс** `HangmanConsoleUI`:
   - **Назначение**: Этот класс отвечает за взаимодействие с пользователем через консоль.
   - **Методы**:
     - `__init__(self)`: Инициализирует игру и устанавливает начальное состояние.
     - `show_game_state(self)`: Отображает текущее состояние игры, используя `HangmanGraphics` для отображения виселицы.
     - `get_user_input(self)`: Обрабатывает ввод пользователя и проверяет его корректность.
     - `update_game_state(self)`: Обновляет состояние игры.
     - `run(self)`: Запускает основной игровой цикл, в котором отображается текущее состояние игры, обрабатывается ввод пользователя и проверяется состояние игры на победу или поражение.
   - **Пример кода**:

```
class HangmanConsoleUI:
    def __init__(self):
        self.game = HangmanGame()
        self.is_running = True

    def show_game_state(self):
        game_state = self.game.get_game_state()
        HangmanGraphics.display(game_state['attempts'])
        print(f"Word: {game_state['word']}")
        print(f"Attempts left: {game_state['attempts']}")
        print(f"Guessed letters: {game_state['guessed_letters']}")

    def get_user_input(self):
        letter = input("Enter a letter: ").lower().strip()
        if len(letter) != 1 or not letter.isalpha():
            print("Invalid input. Please enter a single letter.")
            return self.get_user_input()
        return letter
```

```
    def update_game_state(self):
        game_state = self.game.get_game_state()
        HangmanGraphics.display(game_state['attempts'])

    def run(self):
        while self.is_running:
            self.show_game_state()
            letter = self.get_user_input()
            if self.game.guess_letter(letter) is None:
                print(f"You have already guessed the letter '{letter}'. Please try again.")
            elif self.game.guess_letter(letter):
                print(f"You have guessed the letter '{letter}'. Good job!")
                if self.game.is_won():
                    print("Congratulations! You have won the game!")
                    self.is_running = False
            else:
                print(f"You have missed the letter '{letter}'. Try again.")
                if self.game.is_lost():
                    print("Sorry, you have lost the game. The word was", self.game.state.word)
                    self.is_running = False

if __name__ == "__main__":
    ui = HangmanConsoleUI()
    ui.run()
```

- **Объяснение**:
  - Конструктор инициализирует игру и устанавливает начальное состояние.
  - Метод `show_game_state` отображает текущее состояние игры, используя `HangmanGraphics` для отображения виселицы.
  - Метод `get_user_input` обрабатывает ввод пользователя и проверяет его корректность.
  - Метод `update_game_state` обновляет состояние игры.
  - Метод `run` запускает основной игровой цикл, в котором отображается текущее состояние игры, обрабатывается ввод пользователя и проверяется состояние игры на победу или поражение.