PYTHON329 HW №22
Домашнее задание

Мы возвращаемся к игре в города! В этот раз мы сделаем её рефакторинг на ООП!
Вам пригодится сет городов, который вы получили в виде JSON файла в одной из прошлых реализаций игры.

Я предлагаю вам такую структуру программы. Ниже описана структура классов и методов.
Классы желательно оставить в этом же количестве, методы могут называться иначе и работать иначе.

1. **Класс** `JsonFile`: по работе с JSON который мы писали на прошлом занятии. Имеет методы по чтению и записи JSON. Поправьте аннотацию типов (мы читаем сет строк - наши города)
2. **Класс** `Cities`: Этот класс будет использоваться для представления данных о городах из JSON-файла. Он будет содержать список всех городов.
   - `__init__(self, city_data)`: Конструктор класса `Cities`, который принимает список городов `city_data` и инициализирует состояние объекта. В этом конструкторе происходит наполнение городами.
3. **Класс** `CityGame`: Этот класс будет управлять самой игрой. Он будет принимать экземпляр класса `Cities` в качестве аргумента. Методы класса `CityGame` включают в себя:
   - `__init__(self, cities)`: Конструктор класса `CityGame`, который принимает экземпляр класса `Cities` и инициализирует состояние игры.
   - `start_game(self)`: Метод для начала игры, который включает первый ход компьютера.
   - `human_turn(self, city_input)`: Метод для хода человека, который будет обрабатывать ввод пользователя.
   - `computer_turn(self)`: Метод для хода компьютера, который будет выбирать город на основе правил игры.
   - `check_game_over(self)`: Метод для проверки завершения игры и определения победителя.
   - `save_game_state(self)`: Метод для сохранения состояния игры, если это необходимо.
4. **Управляющий класс** `GameManager`: Этот класс принимает экземпляры классов `JsonFile`, `Cities` и `CityGame` в качестве аргументов. Методы класса `GameManager` включают в себя:
   - `__call__(self)`: Метод `__call__`, который позволяет вызывать объекты этого класса, как если бы они были функциями, и который будет запускать всю игру.
   - `run_game(self)`: Метод для запуска игры, который будет вызывать методы `start_game()`, `human_turn()` и `computer_turn()` поочередно до завершения игры.
   - `display_game_result(self)`: Метод для отображения результата игры после завершения. (По желанию. Опционально)

Общие преимущества структуры программы:

- Модульность: Каждый класс выполняет свою специфическую функцию, что делает код более модульным и понятным.
- Улучшенная читаемость: Хорошо организованный код легче понимать и поддерживать.
- Разделение обязанностей: Каждый класс отвечает за свои обязанности, что позволяет изолировать изменения и уменьшить связность между компонентами.
- Возможность повторного использования: Классы могут быть повторно использованы в других проектах или частях программы.

Каждый из этих классов выполняет свою роль в системе, что помогает создать более структурированный и поддерживаемый код для вашей игры.

Критерии проверки 👌

- Вся работа в одном файле
- Работа сдается в виде файла `.py`
- Аннотации типов
- Игра запускается вызовом экземпляра класса `GameManager`
- Количество классов соблюдено, методы могут отличаться
- Игра запускается в конце документа примерно в такой конструкции

```
if __name__ == "__main__":
    # Создайте экземпляры необходимых классов
    json_file = JsonFile("data.json")  # Замените "data.json" на имя вашего JSON-файла
    cities = Cities(json_file.read_data())
    game = CityGame(cities)

    # Создайте экземпляр GameManager и вызовите его, чтобы начать игру
    game_manager = GameManager(json_file, cities, game)
    game_manager()
```