Пояснительная записка к проекту PyUnoGame

ОБЩАЯ ИНФОРМАЦИЯ

## Технологии

В качестве игрового фреймворка используется библиотека PyGame. Для реализации интерфейса используется дополнительная библиотека pygame\_gui. Сетевая часть реализована с помощью TCP сокетов.

## Основные классы

Стоит понимать, что игра является сетевой, поэтому существует две разных реализации – сервер и клиент, работающие отдельно (и, в идеале, на отдельных компьютерах). Но, клиент и сервер использует для работы одни и те же классы (почему – см. протокол связи). Поэтому, при дистрибуции игры стоит также включать классы (т. е. для клиента можно не включать только сервер, для сервера только клиент + картинки). Большинство классов, для ускорения разработки прототипа, реализованы с помощью dataclass. Вот короткий список основных классов:

* Game

Класс игры, который синхронизируется между всеми клиентами. Содержит в себе всех пользователей в текущей игре, игровую колоду и хранит текущего пользователя и направление движения. Также реализует полезные методы, к примеру:

next\_player – учитывая направление движения, переходит к следующему пользователю

append\_user – добавляет пользователя в игру

* auth.User

Класс пользователя. Содержит в себе имя, id в базе, адрес и колоду игрока в текущей игре. Не реализует в себе никаких методов.

* decks.GameDeck

Класс игровой колоды карт. Хранит в себе только карты игры (игрового поля). Реализует полезные для колоды методы – append\_card, который добавляет колоду в карту и init\_random, который инициализирует колоду одной случайной картой.

* decks.PlayerDeck

Класс колоды карт игрока. Хранит в себе только карты игрока. Также реализует полезные методы – append, который добавляет карту в колоду, init\_random, который инициализирует колоду семью случайными картами и pop, который удаляет карту по индексу и возвращает её

Так же стоит отметить все классы из модуля cards. Достаточно знать о реализации одного из классов – card. Содержит в себе цвет (см. класс enums.colors) карты, и её номинал (должен быть инициализирован самостоятельно). Реализует заглушку метода possible\_move, которая проверяет возможен ли ход, если эта карта верхняя в колоде игры, и метода move, который вызывается, когда происходит ход картой.

Все остальные классы, наследованные от card, переопределяют в себе данные методы и добавляют новые атрибуты, необходимые для реализации определённого типа карт.

КЛИЕНТ

Особых отличий от стандартной реализации игры с помощью PyGame не имеет, за исключением одной. Для удобства разработки существует деление игры на “Экраны”, которые хранятся в отдельном модуле screens. Существует стандартный AbcScreen, от которого наследуются все остальные экраны. Основные атрибуты:

surface – surface игры

manager – класс manager библиотеки pygame\_gui, необходимый для отрисовки интерфейса

networking – класс сетевого взаимодействия клиента (см. подробнее ниже.)

is\_running – атрибут, хранящий в себе состояние экрана – запущен он или нет.

next\_screen – следующий экран, который будет показан после этого экрана

## Основные методы:

run – основной цикл игры. Вызывается каждый тик в игровом цикле игры. Именно в этом методе необходимо отрисовывать всё что нужно. Аргументами так же передаётся список со всеми необработанными в основном цикле игры событиями.

\_handle\_events – метод для обработки событий. Правилом хорошего тона является именно в нём обрабатывать все события.

Так же стоит отметить, что как только is\_running становится False, то происходит смена на экран, указанный в next\_screen.

## Сетевое взаимодействие.

Всё сетевое взаимодействие реализовано в классе Networking. Именно он занимается подключением к серверу, обработкой и отправкой сообщений. Предполагается, что программисту не придётся использовать приватные атрибуты данного класса и инициализировать его самостоятельно. Методы:

login и register – входят или регистрируют пользователя на сервере соответственно. Возвращают класс только что полученного от сервера пользователя. Если войти не получилось, вызывают исключение WrongCredentials с сообщением об ошибке, полученным от сервера.

fetch – получение данных игры от сервера. Предполагается, что вызывать данный метод программисту самостоятельно не придётся. Но важно понимать, что данный метод каждый FETCH\_RATE тиков получает свежие данные от сервера. Таким образом, синхронизируются данные на всех клиентах.

throw\_card – кинуть карту. Атрибутом передаётся индекс карты в колоде игрока. Ответом является булево значение, обозначающие удалось ли или нет кинуть карту.

get\_card – получить карту. Добавляет одну случайную карту в колоду игрока

get\_user\_from\_game – вспомогательный метод, необходимый для получение экземпляра класса User текущего пользователя.

user\_id – вспомогательный метод, необходимый для нахождения индекса переданного аргументом пользователя в списке пользователей игры.

СЕРВЕР

Сервер реализован как простейший Singleton, который реализует в себе основной цикл обработки событий с помощью деления на отдельные процессы для упрощения реализации в рамках прототипа игры. Предполагается, что программист не будет никаким образом задействовать существующие методы из данного класса. Но стоит отметить добавление новых обработчиков событий в сервер – для этого необходимо дописать в метод \_client\_thread новое ветвление с новым типом события, в идеале создать отдельный приватный метод с таким же типом, принимающий аргументом переданные клиентом данные и уже в нём взаимодействовать с классом сервера.

*Примечание автора: мне Вас искренне жаль, если вы захотите туда что-то добавить. На вашем бы месте я занялся чем-то более простым и интересным, к примеру, порешал бы олимпиадные задачки и попробовал себя в алгоритмике. Оставьте это для мазохист*ов, как я.

Также для сервера реализован класс взаимодействия с базой данных Authorization. Он принимает при инициализации путь до базы данных. Реализует два метода: register и login, принимающие аргументами логин и пароль пользователя, и возвращающие только что созданный экземпляр класса User. По наименованием методов понятно, что реализует каждый из них.

ПРОТОКОЛ СВЯЗИ

Сервер-клиент общается по такой схеме: клиент отправляет сообщение определённого типа, сервер на него отвечает. Все сообщения передаются в виде словаря, сериализированного с помощью стандартной библиотеки pickle. Словарь обязательно содержит в себе ключ type со значением типа сообщение. Ниже приведены все реализованные типы сообщений:

|  |  |  |  |
| --- | --- | --- | --- |
| НАИМЕНОВАНИЕ ТИПА | ПОЯСНЕНИЕ К НАИМЕНОВАНИЮ | ПОЛУЧАЕМЫЕ АРГУМЕНТЫ | ВОЗВРАЩАЕТ |
| register | Регистрация пользователя | login: str, password: str | User |
| login | Вход пользователя | login: str, password: str | User |
| fetch | Получение свежих данных | *Ничего* | Game |
| throw | Кинутая карта от клиента | card: Card | bool |
| get\_card | Получить случайную карту | *Ничего* | bool |

УТИЛИТЫ

Для удобства разработки также существует пакет utilities, содержащий в себе полезные функции.

Разберём большинство из них:

* password\_utility

Содержит в себе утилиты для хэширования паролей. hash\_password принимает аргументами пароль, возвращает хэш. Также есть функция для проверки пароля – check\_password, которая принимает пароль и хэш в байтах, возвращает булево значение, обозначающие правда ли что пароль верен.

* text\_utility

Содержит в себе утилиты для удобства отрисовки текста. Реализует функцию text\_on\_center, принимающую surface, на котором необходимо нарисовать текст, шрифт и сам текст, а также координату y текста. center\_rect это вспомогательная функция, которая центрирует rect по центру переданного в неё surface.

* image\_utility

Содержит в себе утилиты для работы с изображениями. В данном прототипе реализует только одну функцию– load\_image.

* card\_utility

Полезные утилиты для работы с карточками. Функция random\_cards возвращает amount случайных карт. Функция card\_image возвращает surface с картинкой, переданной аргументом карточки.

Надеюсь, что в данной пояснительной записке я сумел описать всё, что необходимо для работы над данной программой. По всем вопросам советую обратиться ко мне с помощью электронной почты cralix@cralix.ru