

Структура проекта

Server	2
main.....	2
gameSession(class Session).....	3
gameSession(class Scene).....	3
Client	5
main.py.....	5
global_scope.py.....	5
managers.py.....	5
game.py.....	6
server_interface.py.....	6
Resources	7
Текстуры.....	7
Формат карт.....	7
Репозиторий	9

Server

main

Файл `server/main.py` - исполняемый файл для запуска сервера. Содержит единственный класс `Communicator`. При запуске программы запускается метод класса `run`. У класса коммуникатора есть два процесса, главный - принимает пакеты от клиентов и обрабатывает их. Также запускается отдельный процесс, который отвечает за отправку сообщений от сервера клиентам. Если приходит пакет с `action="connection"`, то сервер проверяет есть доступные комнаты с количеством игроков меньше 4. Если такой нет, он создает новую сессию в отдельном процессе (`gameSession.py`). Таким образом, 2 процесса для коммуникации и по одному процессу для каждой сессии.

Коммуникация между классом `Communicator` и сессиями происходит с помощью `multiprocessing.queue` - очередь, в которую можно добавлять объекты и забирать их. Очередь имеет межпроцессорную реализацию, то есть все процессы видят изменения, которые вносят другие процессы. Таким образом, `Communicator` получает пакет, ищет сессию (проверяет в какой сессии хранится адрес игрока, от которого пришел пакет) и передает в очередь этой сессии.

Создание сессии:

```
def createSession(self):  
    q = multiprocessing.Manager().Queue()  
    playersCount = multiprocessing.Manager().list()  
    playersCount.append(0)  
    session = Session(q, self.senderQueue, playersCount)  
    p = multiprocessing.Process(target=session.simulate)  
    p.start()  
    self.gameSessions[p.pid] = [q, playersCount, []]  
    return p.pid
```

gameSession(class Session)

Session - класс, который отвечает за симуляцию игровой комнаты. Содержит в качестве атрибута класс Scene - класс, в котором просчитывается логика и физика игры. Вызываемый метод у класса Session - simulate. Он принимает сообщения от Communicator и обрабатывает их. В методе запущен цикл While, поэтому он бесконечно принимает сообщения и на каждой итерации вызывает метод sceneSimulate.

Обработкой сообщений занимается метод handleCommand, который парсит пакет и принимает определенные действия с возможностью отправки ответа клиенту (передает ответный пакет в очередь, а Communicator его отправляет). В классе есть методы по генерации пакетов (в соответствии с protocol.md). Все что касается с взаимодействием с геймплеем, то Session вызывает методы класса Scene.

gameSession(class Scene)

Данный класс отвечает обработку игровой логики на стороне Сервера. Поскольку, с точки зрения логики, клиент является тонким, данный класс отвечает за:

- Хранение данных о состоянии игры:
 - Хранение игровой карты (данных static)
 - Хранение объектов игры (данных dynamic)
- Обработку физики игры, за это отвечает метод processPhysics()
 - Расчет передвижения игроков
 - Расчет столкновения игроков с картой (static)
 - Расчет полета пуль
 - Расчет столкновения пуль с картой
- Изменение состояния карты, за это отвечают методы removePlayer(), addDynamicShotgunBullet(), deleteDynamicById(), addDynamicPlayer()

- Добавление и удаление игровых объектов
- Обработку пакетов нажатия клавиш, посылаемых клиентами, за это отвечает метод `processPlayerInput()`
- Генерацию пакетов обновления данных о карте, посылаемых клиентам `generateSceneUpdatePacket()`

Client

Клиент игры является тонким и содержит только логику отрисовки на экран данных, приходящих с сервера и код отправки нажатий клавиш игроком.

Программа клиента хранится в директории `./client/` и содержит следующие файлы:

- `main.py`
- `global_scope.py`
- `managers.py`
- `game.py`
- `server_interface.py`

`main.py`

Данный файл является запускаемым файлом клиентского приложения. Код в файле является управляющим кодом для других python файлов в директории. Код содержит цикл приложения, где осуществляется управление разными состояниями приложения: переключение между меню, игровым процессом и прочими меню приложения.

`global_scope.py`

Файл используется для хранения настроек приложения, а также участвует в подгрузке ресурсов приложения в оперативную память. Настройки приложения должны быть доступны из любой точки кода приложения.

`managers.py`

Manager – термин библиотеки `pygame_gui`, который описывает набор элементов графического интерфейса, объединенных одним смыслом, например менеджер для меню приложения, менеджер для кнопки чата и текстового поля в игре.

`game.py`

Файл содержит “тонкую сторону” сервера игры и код, описывающий отрисовку данных, приходящих с сервера.

`server_interface.py`

Файл содержит код сетевого взаимодействия с сервером.

Resources

В данном каталоге хранятся все вспомогательные файлы, необходимые для отображения интерфейса: текстуры, музыка и карты по умолчанию.

Текстуры

Текстуры представляют собой PNG-файлы с разрешением 32x32. Но модель игрока - 32x64, модель мертвого игрока - 64x64. Разрешение фонов для меню и карт - 800x600.

Формат карт

Каждая карта представляет собой файл формата JSON, где описано её название, автор, фон и координаты для спавна игроков и сама структура блоков (ключ static):

В ключе `static` содержится массив строк с одинаковой длиной, где каждый символ обозначает определённый блок:

[illegible]

Соответствие между символами в строках и обрабатываемыми в программе блоками прописаны в файле client/global_scope.py в словаре SYMBOLS:

```
1
2 SYMBOLS = {
3     '=': 'stone',
4     '#': 'stone_darker',
5     'E': 'reinforced_concrete',
6     '$': 'metal',
7     '%': 'metal_beam',
8     '~': 'leaves',
9     '*': 'grass',
10    '&': 'deep_ground',
11    '+': 'bricks',
12    'x': 'box'
13 }
```

Точки в строках - это воздух.

На текущий момент в программе предусмотрены 3 карты по умолчанию: предприятие, паркур и пещера.

Репозиторий

Репозиторий игры располагается на GitHub по ссылке
<https://github.com/losdayver/Internet-Battle>