Міністерство освіти і науки України

Харківський національний університет радіоелектроніки

ЦЕНТР ЗАОЧНОЇ ФОРМИ НАВЧАННЯ

КАФЕДРА ПРОГРАМНОЇ ІНЖЕНЕРІЇ

**АТЕСТАЦІЙНА РОБОТА (ПРОЕКТ)**

**Пояснювальна записка**

Бакалавр

(освітньо-кваліфікаційний рівень)

***Мобільна гра з клієнт-серверною архітектурою під ОС Андроїд "Тріо"***

Виконав: студент 5 курсу, групи ПІз-11-1

напряму підготовки (спеціальності)

6.050103 *Програмна інженерія*

Нікулін А.В.

Керівник Самофалов Л. Д.

Рецензент

Допускається до захисту

Зав. кафедри, проф. \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Дудар З. В.

2016

Харківський національний університет радіоелектроніки

Факультет Центр заочної форми навчання

Кафедра Програмної інженерії

Освітньо-кваліфікаційний рівень *бакалавр*

Напрям підготовки *6.050103 Програмна інженерія*

ЗАТВЕРДЖУЮ:

Зав. каф. проф.\_\_\_\_\_\_\_\_\_\_\_\_Дудар З. В.

«\_\_\_\_\_»\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 20 \_\_\_ р.

ЗАВДАННЯ

НА АТЕСТАЦІЙНУ РОБОТУ (ПРОЕКТ)

студентові *Нікуліну Антону Володимировичу*

1. Тема проекту «***Мобільна гра з клієнт-серверною архітектурою під ОС Андроїд "Тріо"***» затверджена наказом по університету від \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_.
2. Термін подання студентом роботи (проекту): 20 *травня* 20*16* р
3. Вихідні дані до проекту :*Розробити кліентське застосування під платформу андроїд, серверну частину для зьеднаня застосувань у мережу. Використовувати мову Java з фреймворком Libgdx для кліентської частини, та мову Python 3,5 для розробки серверу.*
4. Зміст пояснювальної записки (перелік питань, які потрібно розробити) *мета роботи, аналіз користувальницьких і розробка функціональних вимог до програмного продукту, опис прийнятих проектних рішень, методи та алгоритми, що використовувались, структура проекту, опис роботи застосування, тестування ПЗ та аналіз дослідної експлуатації. Додатки:, а) слайди презентації, б) коди програми, в) робочій проект на CD.*
5. Перелік графічного матеріалу (з точним зазначенням обов’язкових креслень) *Слайди презентації: титул, мета проекту, обґрунтування доцільності розробки, постановка задачі, об'єктна модель системи, базові моделі, методи й алгоритми, uml діаграмма класів, структурно-логічна схема взаємодії даних, план захисту інформації, інтерфейс програмної системи, результати тестування програмної системи, висновки.*

Зворотний бік бланку завдання

6. Консультанти розділів роботи (проекту)

|  |  |  |  |
| --- | --- | --- | --- |
| Найменування  розділу | Консультант | Позначка консультанта  про виконання розділу | |
| підпис | дата |
| Спецчастина | доц. Самофалов Л.Д. |  |  |
|  |  |  |  |

7. Дата видачі завдання 21-02-2016

**КАЛЕНДАРНИЙ ПЛАН**

|  |  |  |  |
| --- | --- | --- | --- |
| № | Назва етапів роботи (проекту) | Термін  виконання етапів проекту (роботи) | Примітка |
| 1 | Аналіз предметної галузі | 23-02-16 |  |
| 2 | Розробка специфікації ПЗ | 02-03-16 |  |
| 3 | Об'єктний аналіз поставленої задачі | 09-03-16 |  |
| 4 | Створення коду програми | 16-03-16 |  |
| 5 | Тестування і налагодження програми | 06-04-16 |  |
| 6 | Підготовка пояснювальної записки | 14-04-16 |  |
| 7 | Підготовка презентації та доповіді | 15-05-16 |  |
| 8 | Нормоконтроль, рецензування | 25-05-16 |  |
| 9 | Попередній захист | 27-05-16 |  |
| 10 | Занесення диплома в електронний архів | 29-05-16 |  |
| 11 | Допуск до захисту у зав. кафедри | 01-06-16 |  |

Студент \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Нікулін А.В.

Kepiвник роботи (проекту) \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ доц. Самофалов Л.Д.

РЕФЕРАТ / ABSTRACT

Пояснювальна записка до випускної атестаційної роботи бакалавра: \_\_\_ стор., \_\_ рис., \_\_ табл., \_\_\_ джерел.

Об'єкт розробки – Мобільна гра з клієнт-серверною архітектурою під Андроїд.

Мета розробки– розробити діючу мобільну гру під Андроїд та серверну частину.

Метод проектування – Android Studio, Pycharm, мова Java, Python, фреймворк Libgdx.

В результаті виконання випускної кваліфікаційної роботи бакалавра розроблено мобільну гру під Андроїд, сервер для зьеднання кількох кліентів для гри вдвох.

АТЕСТАЦІЙНА РОБОТА БАКАЛАВРА,МОБІЛЬНА ГРА, КЛІЕНТ-СЕРВЕР, ANDROID STUDIO, МОВА JAVA, PYTHON, LIBGDX, ASYNCIO, TCP.

Explanatory note to the final certification of the bachelor: \_\_\_ pg., \_\_ fig., \_\_\_table, \_\_\_ sources.

The object of development –Mobile application with client server architecture for cooperative gaming.

The purpose of development – to develop mobile Android game that can connect to server.

Method Design– Android Studio, language Java, Python, Libgdx.

As a result of final qualifying Bachelor mobile game developed Android, server for multiple clients to play together.

BACHELOR DEGREE ATTESTATION WORK, MOBILE GAME, CLIENT-SERVER, ANDROID STUDIO, LANGUAGE JAVA, PYTHON, LIBGDX, ASYNCIO, TCP.

ОГЛАВЛЕНИЕ

[Введение 5](#_Toc453185342)

[1 Анализ пользовательских и разработка функциональных требований к программному продукту 8](#_Toc453185343)

[1.1 Обзор существующего рынка мобильных приложений 8](#_Toc453185344)

[1.2 Постановка задачи 9](#_Toc453185345)

[1.3 Требования к системе 9](#_Toc453185346)

[1.4 Выбор языка программирования 10](#_Toc453185347)

[2 Описание принятых проектных решений 13](#_Toc453185348)

[2.1 Структура и функционирование приложения 13](#_Toc453185349)

[2.2 Схема клиента приложения 18](#_Toc453185350)

[3 Описание работы приложения 24](#_Toc453185351)

[3.1 Начало работы с сервером. 24](#_Toc453185352)

[3.2 Работа с клиентом 25](#_Toc453185353)

[4 Тестирование программного продукта, анализ опытной эксплуатации 29](#_Toc453185354)

[4.1 Основные понятия в тестировании 29](#_Toc453185355)

[4.2 Инструментарий 30](#_Toc453185356)

[4.3 Категории тестов 30](#_Toc453185357)

[4.4 Журнал тестирования и таблица проверок 32](#_Toc453185358)

[Выводы 34](#_Toc453185359)

[Список источников 35](#_Toc453185360)

[Приложение А Слайды презентации 36](#_Toc453185361)

[Приложение Б примеры кодов программы 47](#_Toc453185362)

[Приложение В DVD с электронными материалами к проекту](#_Toc453185365)

# Введение

Мобильная игра́ — [игровая](https://ru.wikipedia.org/wiki/%D0%98%D0%B3%D1%80%D0%B0) [программа](https://ru.wikipedia.org/wiki/%D0%9A%D0%BE%D0%BC%D0%BF%D1%8C%D1%8E%D1%82%D0%B5%D1%80%D0%BD%D0%B0%D1%8F_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B0) для [мобильных устройств](https://ru.wikipedia.org/w/index.php?title=%D0%9C%D0%BE%D0%B1%D0%B8%D0%BB%D1%8C%D0%BD%D0%BE%D0%B5_%D1%83%D1%81%D1%82%D1%80%D0%BE%D0%B9%D1%81%D1%82%D0%B2%D0%BE&action=edit&redlink=1), например [сотовых телефонов](https://ru.wikipedia.org/wiki/%D0%A1%D0%BE%D1%82%D0%BE%D0%B2%D1%8B%D0%B9_%D1%82%D0%B5%D0%BB%D0%B5%D1%84%D0%BE%D0%BD), [смартфонов](https://ru.wikipedia.org/wiki/%D0%A1%D0%BC%D0%B0%D1%80%D1%82%D1%84%D0%BE%D0%BD), [коммуникаторов](https://ru.wikipedia.org/wiki/%D0%9A%D0%BE%D0%BC%D0%BC%D1%83%D0%BD%D0%B8%D0%BA%D0%B0%D1%82%D0%BE%D1%80), [КПК](https://ru.wikipedia.org/wiki/%D0%9A%D0%B0%D1%80%D0%BC%D0%B0%D0%BD%D0%BD%D1%8B%D0%B9_%D0%BF%D0%B5%D1%80%D1%81%D0%BE%D0%BD%D0%B0%D0%BB%D1%8C%D0%BD%D1%8B%D0%B9_%D0%BA%D0%BE%D0%BC%D0%BF%D1%8C%D1%8E%D1%82%D0%B5%D1%80) и прочих (за исключением [ноутбуков](https://ru.wikipedia.org/wiki/%D0%9D%D0%BE%D1%83%D1%82%D0%B1%D1%83%D0%BA)).

Android — [операционная система](https://ru.wikipedia.org/wiki/%D0%9E%D0%BF%D0%B5%D1%80%D0%B0%D1%86%D0%B8%D0%BE%D0%BD%D0%BD%D0%B0%D1%8F_%D1%81%D0%B8%D1%81%D1%82%D0%B5%D0%BC%D0%B0) для [смартфонов](https://ru.wikipedia.org/wiki/%D0%A1%D0%BC%D0%B0%D1%80%D1%82%D1%84%D0%BE%D0%BD), [интернет-планшетов](https://ru.wikipedia.org/wiki/%D0%98%D0%BD%D1%82%D0%B5%D1%80%D0%BD%D0%B5%D1%82-%D0%BF%D0%BB%D0%B0%D0%BD%D1%88%D0%B5%D1%82), [электронных книг](https://ru.wikipedia.org/wiki/%D0%AD%D0%BB%D0%B5%D0%BA%D1%82%D1%80%D0%BE%D0%BD%D0%BD%D0%B0%D1%8F_%D0%BA%D0%BD%D0%B8%D0%B3%D0%B0_%28%D1%83%D1%81%D1%82%D1%80%D0%BE%D0%B9%D1%81%D1%82%D0%B2%D0%BE%29), [цифровых проигрывателей](https://ru.wikipedia.org/wiki/%D0%A6%D0%B8%D1%84%D1%80%D0%BE%D0%B2%D0%BE%D0%B9_%D0%BF%D1%80%D0%BE%D0%B8%D0%B3%D1%80%D1%8B%D0%B2%D0%B0%D1%82%D0%B5%D0%BB%D1%8C), [наручных часов](https://ru.wikipedia.org/wiki/%D0%A3%D0%BC%D0%BD%D1%8B%D0%B5_%D1%87%D0%B0%D1%81%D1%8B), [игровых приставок](https://ru.wikipedia.org/wiki/%D0%98%D0%B3%D1%80%D0%BE%D0%B2%D0%B0%D1%8F_%D0%BF%D1%80%D0%B8%D1%81%D1%82%D0%B0%D0%B2%D0%BA%D0%B0), [нетбуков](https://ru.wikipedia.org/wiki/%D0%9D%D0%B5%D1%82%D0%B1%D1%83%D0%BA), основана на [ядре Linux](https://ru.wikipedia.org/wiki/%D0%AF%D0%B4%D1%80%D0%BE_Linux) и собственной реализации виртуальной машины [Java](https://ru.wikipedia.org/wiki/Java_Virtual_Machine) от [Google](https://ru.wikipedia.org/wiki/Google_%28%D0%BA%D0%BE%D0%BC%D0%BF%D0%B0%D0%BD%D0%B8%D1%8F%29). Изначально разрабатывалась компанией [Android Inc.](https://ru.wikipedia.org/wiki/Android_Inc.), которую затем купила [Google](https://ru.wikipedia.org/wiki/Google_%28%D0%BA%D0%BE%D0%BC%D0%BF%D0%B0%D0%BD%D0%B8%D1%8F%29). Впоследствии Google инициировала создание альянса [Open Handset Alliance](https://ru.wikipedia.org/wiki/Open_Handset_Alliance) (OHA), который сейчас занимается поддержкой и дальнейшим развитием платформы. Android позволяет создавать [Java](https://ru.wikipedia.org/wiki/Java)-приложения, управляющие устройством через разработанные Google библиотеки. Android Native Development Kit позволяет портировать библиотеки и компоненты приложений, написанные на [Си](https://ru.wikipedia.org/wiki/%D0%A1%D0%B8_%28%D1%8F%D0%B7%D1%8B%D0%BA_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D1%8F%29) и других [языках](https://ru.wikipedia.org/wiki/%D0%AF%D0%B7%D1%8B%D0%BA_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D1%8F)[1].

В 86 % смартфонов, проданных во втором квартале 2014 года, была установлена операционная система Android. При этом за весь 2014 год было продано более 1 миллиарда Android-устройств О конкуренции мобильных игр с компьютерными и консольными даже речи быть не может. Большая часть таких игр требуют внимания и погружения на долгое время, Андроид приложения же совсем наоборот – не займут у Вас много времени и не заставят вникать в каждую мелочь. В результате своей популярности мобильные игры сформировали несколько игровых жанров, некоторые из которых уникальные и ранее просто не существовали.

Вот несколько наиболее широких игровых жанров для Андроид игр:

- Action (экшн, в пер. с англ. – действие) — один из основных жанров мобильных игр, в котором упор сделан на активные действия игрока и результат зависит от того, насколько игрок способен быстро принимать решения. В экшн-играх действие всегда развивается очень быстро;

- RPG (сокращение от англ. Role-Playing Game) – ролевая игра, отдельный жанр мобильных игр, в котором игрок непосредственно управляет одним или, реже, несколькими персонажами, причём каждый из них имеет свои характеристики: способности, показатели выносливости, здоровья, силы и защиты, может относиться к какому-либо классу, развивать свои навыки и выполнять миссии;

- стратегия (от англ. Strategy) – жанр мобильных игр, требующий от игрока выработки определённой схемы действий — стратегии, которая должна привести к достижению цели. В стратегиях игроку приходится управлять целой группой персонажей, племенем, отрядом, государством и даже галактикой или вселенной;

- симулятор (от англ. simulator – симулятор, имитатор) – жанр игр, в основе которого лежит имитация управления каким-либо процессом из реальной жизни. Симуляторы подразделяются на несколько видов;

- Adventure (адвенчур, в пер. с англ. приключение) – игры с глубоким разветвлённым сюжетом, в которых главный герой должен взаимодействовать с окружающим миром, общаясь с другими персонажами, используя найденные предметы и решая головоломки.

Стратегическая игра — популярный жанр [компьютерных игр](https://ru.wikipedia.org/wiki/%D0%9A%D0%BE%D0%BC%D0%BF%D1%8C%D1%8E%D1%82%D0%B5%D1%80%D0%BD%D0%B0%D1%8F_%D0%B8%D0%B3%D1%80%D0%B0), в котором залогом достижения победы является планирование и [стратегическое](https://ru.wikipedia.org/wiki/%D0%A1%D1%82%D1%80%D0%B0%D1%82%D0%B5%D0%B3%D0%B8%D1%8F) мышление. Смысл таких игр заключается в управлении определёнными ресурсами, которые необходимо преобразовать в преимущество над противником при помощи оперативного плана, разрабатываемого с учётом меняющейся обстановки. Обычными ресурсами в военных стратегиях являются войска (отдельные персонажи, подразделения или армии) и позиция, которые следует развивать и использовать для достижения преимущества и победы. В экономических стратегиях акцент ставится на развитие экономической инфраструктуры подконтрольной игроку стороны. Современные стратегические игры, как правило, соединяют в себе как военные, так и экономические признаки. В большинстве стратегий существуют экономическая (сбор ресурсов, подготовка войск) и военная составляющие части.Различают [пошаговые](https://ru.wikipedia.org/wiki/%D0%9F%D0%BE%D1%88%D0%B0%D0%B3%D0%BE%D0%B2%D0%B0%D1%8F_%D1%81%D1%82%D1%80%D0%B0%D1%82%D0%B5%D0%B3%D0%B8%D1%8F) стратегические игры ([англ.](https://ru.wikipedia.org/wiki/%D0%90%D0%BD%D0%B3%D0%BB%D0%B8%D0%B9%D1%81%D0%BA%D0%B8%D0%B9_%D1%8F%D0%B7%D1%8B%D0%BA) Turn-Based Strategy, TBS), где игроки поочерёдно делают ходы, и каждому игроку отводится неограниченное или ограниченное (в зависимости от типа и сложности игры) время на свой ход, и стратегические игры [в реальном времени](https://ru.wikipedia.org/wiki/%D0%A1%D1%82%D1%80%D0%B0%D1%82%D0%B5%D0%B3%D0%B8%D1%8F_%D0%B2_%D1%80%D0%B5%D0%B0%D0%BB%D1%8C%D0%BD%D0%BE%D0%BC_%D0%B2%D1%80%D0%B5%D0%BC%D0%B5%D0%BD%D0%B8) ([англ.](https://ru.wikipedia.org/wiki/%D0%90%D0%BD%D0%B3%D0%BB%D0%B8%D0%B9%D1%81%D0%BA%D0%B8%D0%B9_%D1%8F%D0%B7%D1%8B%D0%BA) Real Time Strategy, RTS), в которых все игроки выполняют свои действия одновременно, и ход времени не прерывается. Также существуют онлайн-стратегии, предназначенные только для игры в интернете. Среди них можно выделить браузерные игры и игры, требующие использования клиента. В области стратегий, использующих клиент, ранее существовали только игры, выпускавшиеся с однопользовательским режимом. Поша́говая страте́гия ([англ.](https://ru.wikipedia.org/wiki/%D0%90%D0%BD%D0%B3%D0%BB%D0%B8%D0%B9%D1%81%D0%BA%D0%B8%D0%B9_%D1%8F%D0%B7%D1%8B%D0%BA) Turn-Based Strategy, TBS) — поджанр [компьютерных стратегических игр](https://ru.wikipedia.org/wiki/%D0%9A%D0%BE%D0%BC%D0%BF%D1%8C%D1%8E%D1%82%D0%B5%D1%80%D0%BD%D0%B0%D1%8F_%D1%81%D1%82%D1%80%D0%B0%D1%82%D0%B5%D0%B3%D0%B8%D1%87%D0%B5%D1%81%D0%BA%D0%B0%D1%8F_%D0%B8%D0%B3%D1%80%D0%B0), в которых игровой процесс состоит из последовательности фиксированных моментов времени, именуемых ходами (или шагами), во время которых игроки совершают свои действия[2].

Основной характеристикой пошаговых стратегических игр является [дискретность](https://ru.wikipedia.org/wiki/%D0%94%D0%B8%D1%81%D0%BA%D1%80%D0%B5%D1%82%D0%BD%D0%BE%D1%81%D1%82%D1%8C) игрового процесса. Игра состоит из фиксированных во времени моментов («шагов» или «ходов»), которые завершаются только по команде игрока. Во время этих ходов игрок совершает свои действия. Один ход может соответствовать промежутку во много лет в игровом мире, за которые игрок успевает управиться с событиями в каждом городе империи и отдать приказы сотням военных отрядов

Компьютерные пошаговые стратегии происходят от настольных стратегических игр, в которых игроки, как правило, совершали действия по очереди. До 1990 года почти все стратегические компьютерные игры были пошаговыми. Большинство первых пошаговых стратегий были либо вариациями существующих настольных игр, либо в той или иной степени были вдохновлены ими

Существует тип стратегических игр, которые обычно относят к пошаговым, но игроки в них делают ходы одновременно. Иногда их еще называют пошаговыми стратегиями в реальном времени или tick-based strategy (от [англ.](https://ru.wikipedia.org/wiki/%D0%90%D0%BD%D0%B3%D0%BB%D0%B8%D0%B9%D1%81%D0%BA%D0%B8%D0%B9_%D1%8F%D0%B7%D1%8B%D0%BA) tick — метка). В таких играх все игроки совершают игровые действия одновременно и в реальном времени. Когда игрок заканчивает возможные или желаемые действия, он отмечает, что готов к следующему ходу. После того, как все игроки просигнализировали о завершении хода, начинается следующий ход. Игры данного типа часто относят к пошаговым стратегиям, потому что в основе игрового процесса лежит «пошаговость», несмотря на то, что действия совершаются одновременно Компьютерные пошаговые стратегии происходят от настольных стратегических игр, в которых игроки, как правило, совершали действия по очереди. До 1990 года почти все стратегические компьютерные игры были пошаговыми. Большинство первых пошаговых стратегий были либо вариациями существующих настольных игр, либо в той или иной степени были вдохновлены ими.

Текстовую игру конца 1960-х годов [Hamurabi](https://ru.wikipedia.org/wiki/Hamurabi) часто называют прародителем компьютерных стратегических игр. В ней игроку предстояло управлять ресурсами и землями [шумерского](https://ru.wikipedia.org/wiki/%D0%A8%D1%83%D0%BC%D0%B5%D1%80) государства, а сам игровой процесс был пошаговым.

В 1971 году [Питер Лэнгстон](https://ru.wikipedia.org/w/index.php?title=%D0%9B%D1%8D%D0%BD%D0%B3%D1%81%D1%82%D0%BE%D0%BD,_%D0%9F%D0%B8%D1%82%D0%B5%D1%80&action=edit&redlink=1) создал многопользовательскую пошаговую стратегическую игру [Empire](https://ru.wikipedia.org/w/index.php?title=Empire_%28%D0%B8%D0%B3%D1%80%D0%B0,_1971%29&action=edit&redlink=1), которая была основана на одноименной настольной игре, разработанной студентами [Рид-колледжа](https://ru.wikipedia.org/wiki/%D0%A0%D0%B8%D0%B4-%D0%BA%D0%BE%D0%BB%D0%BB%D0%B5%D0%B4%D0%B6). В 1977 году [Уолтер Брайт](https://ru.wikipedia.org/wiki/%D0%91%D1%80%D0%B0%D0%B9%D1%82,_%D0%A3%D0%BE%D0%BB%D1%82%D0%B5%D1%80) создал пошаговую стратегию, которая тоже называлась [Empire](https://ru.wikipedia.org/wiki/Empire_%28%D0%B8%D0%B3%D1%80%D0%B0%29). По словам Брайта, игра не имела никакого отношения к игре из Рид-колледжа и базировалась на настольной игре собственного авторства, созданной по мотивам игры [Риск](https://ru.wikipedia.org/wiki/%D0%A0%D0%B8%D1%81%D0%BA_%28%D0%BD%D0%B0%D1%81%D1%82%D0%BE%D0%BB%D1%8C%D0%BD%D0%B0%D1%8F_%D0%B8%D0%B3%D1%80%D0%B0%29)

Некоторые стратегические компьютерные игры комбинируют в себе элементы пошаговой стратегии и стратегии в реальном времени. В серии игр [Total War](https://ru.wikipedia.org/wiki/Total_War) от [Creative Assembly](https://ru.wikipedia.org/wiki/Creative_Assembly) игровой процесс делится на две части. Основная игра идет в пошаговом режиме, но во время сражений игра переходит в режим реального времени.

1 АНАЛИЗ ПОЛЬЗОВАТЕЛЬСКИХ И РАЗРАБОТКА ФУНКЦИОНАЛЬНЫХ ТРЕБОВАНИЙ К ПРОГРАММНОМУ ПРОДУКТУ

## Анализ пользовательских требований в разрабатываемой системе

В 2012 году рынок мобильных приложений оценивался в 53 миллиарда долларов, а прогноз на 2016 год гласил, что предполагаемый рост составит около 100 миллиардов долларов. Эти цифры немного отличаются у разных исследователей, но очевидным остается то, что мобильный рынок действительно масштабен. Доход разработчики получают с помощью внутренних in-app покупок, рекламы внутри приложений, а также сбора больших данных (big data). Самые многообещающие категории – это социальные сети, производительность, рекламные сервисы, а также полезные приложения для различных целей. Самые быстрорастущие рынки – Юго-Восточная Азия и Латинская Америка. [1]

Приблизительное число разработчиков мобильных приложений – 2.3 миллиона человек, а это означает, что каждый восьмой из всех разработчиков в мире создает мобильные приложения. В 2013 году компания Apple во время WorldWide Developer Conference объявила, что в AppStore опубликовано уже 1.25 миллионов приложений, которые пользователи скачали 50 миллиардов раз, а разработчики получили доход в 5 миллиардов долларов. [3]

Ожидается, что скачка мобильных приложений вырастет до 200 миллиардов в год, а доходы в 2017 году будут равняться 63.5 млрд. дол.. Аналитические прогнозы показывают, что рыночная стоимость мобильных платежей во всем мире вырастет от 235 млрд. дол. в 2013 году до 721.3 млрд. дол. в 2017. Основной причиной такого бурного роста мобильных рынков является небывалый рост продаж планшетов, смартфоном и других мобильных устройств. В данный момент большинство мобильных приложений относятся к категории B2C, т.к. B2B приложения еще только-только начинают выходить на рынок. Но потенциальный рост их очень велик. Несмотря на различные цифры в прогнозируемых показателях динамики рынка, все исследователи соглашаются, что рынок мобильных приложений является одним из самых перспективных направлений.[4]

Для мобильных игр характерны частые, но короткие игровые сессии. Мобильные игры – другие, и это требует от разработчиков новых подходов к их разработке и монетизации

## Постановка задачи

В рамках выпускной работы бакалавра основной задачей является создание программного продукта игры под платформу андроид, с возможностью подключения к удаленному серверу приложений для сетевой игры. Игрок должен управлять отрядом персонажей в пошаговом режиме, против команды управляемой другим игроком. Целью игры является устраниения всех персонажей протвиположной команды. Игрок должен иметь возможность играть по сети против другого игрока, или на одном устройстве в автономном режиме. Игра разработана в жанре пошаговой стратегии.

Необходимо разработать структуру мобильного клиента, с основным меню для выбора последующих действий пользователя, сетевой модуль для подключения к серверу.

Экран меню для задания конфигурационных параметров и выборе предопределённых опций.

В том числе экран для выбора подключения, и задания адреса и портов принимающего сервера в асинхронном режиме осуществляющим подключение и обмен данными.

Сервер приложения должен осуществлять прием подключаемых клиентов, и обработки запросов клиентов в реальном времени реализуя шаблон «реактор» в асинхронном режиме.

В случае невозможности организовать клиент серверное подключение, предположительно из-за отсутствия сетевого соединения. Клиент должен иметь возможность осуществлять работу в офлайновом режиме реализуя подход «hot seat» при котором несколько игроков играют с одного устройства.

## Требования к системе

Для запуска мобильного клиента требуется мобильное устройство на котором установлена версия операционной системы андроид версии 4 и старше, желательно с наличием системы вывода звуков и процессором, содержащим графическое ускорение, с диагональю экрана не менее 7 дюймов и разрешением экрана не менее 480 на 854 пикселей.

Для запуска серверной части требуется 32х или 64х разрядная операционная система работающая под управление ОС Windows/Linux 1гб оперативной памяти и установленным python35, или другой версией в которой установлен модуль asyncio и существует поддержка асинхронных операций.

## Выбор языка программирования

Java — [объектно-ориентированный язык программирования](https://ru.wikipedia.org/wiki/%D0%9E%D0%B1%D1%8A%D0%B5%D0%BA%D1%82%D0%BD%D0%BE-%D0%BE%D1%80%D0%B8%D0%B5%D0%BD%D1%82%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%BD%D1%8B%D0%B9_%D1%8F%D0%B7%D1%8B%D0%BA_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D1%8F), разработанный компанией [Sun Microsystems](https://ru.wikipedia.org/wiki/Sun_Microsystems) (в последующем приобретённой компанией [Oracle](https://ru.wikipedia.org/wiki/Oracle)). Приложения Java обычно [транслируются](https://ru.wikipedia.org/wiki/%D0%A2%D1%80%D0%B0%D0%BD%D1%81%D0%BB%D1%8F%D1%82%D0%BE%D1%80) в специальный [байт-код](https://ru.wikipedia.org/wiki/%D0%91%D0%B0%D0%B9%D1%82-%D0%BA%D0%BE%D0%B4), поэтому они могут работать на любой [виртуальной Java-машине](https://ru.wikipedia.org/wiki/Java_Virtual_Machine) вне зависимости от [компьютерной архитектуры](https://ru.wikipedia.org/wiki/%D0%9A%D0%BE%D0%BC%D0%BF%D1%8C%D1%8E%D1%82%D0%B5%D1%80%D0%BD%D0%B0%D1%8F_%D0%B0%D1%80%D1%85%D0%B8%D1%82%D0%B5%D0%BA%D1%82%D1%83%D1%80%D0%B0). Дата официального выпуска — 23 мая 1995 года. Программы на Java [транслируются](https://ru.wikipedia.org/wiki/%D0%A2%D1%80%D0%B0%D0%BD%D1%81%D0%BB%D1%8F%D1%82%D0%BE%D1%80) в [байт-код](https://ru.wikipedia.org/wiki/%D0%91%D0%B0%D0%B9%D1%82-%D0%BA%D0%BE%D0%B4), выполняемый [виртуальной машиной Java](https://ru.wikipedia.org/wiki/Java_Virtual_Machine) (JVM) — программой, обрабатывающей байтовый код и передающей инструкции оборудованию как [интерпретатор](https://ru.wikipedia.org/wiki/%D0%98%D0%BD%D1%82%D0%B5%D1%80%D0%BF%D1%80%D0%B5%D1%82%D0%B0%D1%82%D0%BE%D1%80).

Программный код, написанный на Java, [виртуальная машина Java](https://ru.wikipedia.org/wiki/Java_Virtual_Machine) преобразует в [байт-код Java](https://ru.wikipedia.org/wiki/%D0%91%D0%B0%D0%B9%D1%82-%D0%BA%D0%BE%D0%B4_Java). Однако есть компиляторы [байт-кода](https://ru.wikipedia.org/wiki/%D0%91%D0%B0%D0%B9%D1%82-%D0%BA%D0%BE%D0%B4) для других языков программирования, таких как [Ada](https://ru.wikipedia.org/wiki/%D0%90%D0%B4%D0%B0_%28%D1%8F%D0%B7%D1%8B%D0%BA_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D1%8F%29), [JavaScript](https://ru.wikipedia.org/wiki/JavaScript), [Python](https://ru.wikipedia.org/wiki/Python), и [Ruby](https://ru.wikipedia.org/wiki/Ruby). Также есть несколько новых языков программирования, разработанных для работы с виртуальной машиной Java. Это такие языки как [Scala](https://ru.wikipedia.org/wiki/Scala_%28%D1%8F%D0%B7%D1%8B%D0%BA_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D1%8F%29), [Clojure](https://ru.wikipedia.org/wiki/Clojure) and [Groovy](https://ru.wikipedia.org/wiki/Groovy). [Синтаксис Java](https://ru.wikipedia.org/w/index.php?title=%D0%A1%D0%B8%D0%BD%D1%82%D0%B0%D0%BA%D1%81%D0%B8%D1%81_Java&action=edit&redlink=1) в основном заимствован из [Си](https://ru.wikipedia.org/wiki/%D0%A1%D0%B8_%28%D1%8F%D0%B7%D1%8B%D0%BA_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D1%8F%29) и [C++](https://ru.wikipedia.org/wiki/C%2B%2B), но объектно-ориентированные возможности основаны на модели, используемой в [Smalltalk](https://ru.wikipedia.org/wiki/Smalltalk) и [Objective-C](https://ru.wikipedia.org/wiki/Objective-C). В Java отсутствуют определённые [низкоуровневые](https://ru.wikipedia.org/wiki/%D0%9D%D0%B8%D0%B7%D0%BA%D0%BE%D1%83%D1%80%D0%BE%D0%B2%D0%BD%D0%B5%D0%B2%D1%8B%D0%B9_%D1%8F%D0%B7%D1%8B%D0%BA_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D1%8F) конструкции, такие как [указатели](https://ru.wikipedia.org/wiki/%D0%A3%D0%BA%D0%B0%D0%B7%D0%B0%D1%82%D0%B5%D0%BB%D1%8C_%28%D1%82%D0%B8%D0%BF_%D0%B4%D0%B0%D0%BD%D0%BD%D1%8B%D1%85%29), также Java имеет очень простую модель памяти, где каждый объект расположен [в куче](https://ru.wikipedia.org/wiki/%D0%94%D0%B8%D0%BD%D0%B0%D0%BC%D0%B8%D1%87%D0%B5%D1%81%D0%BA%D0%BE%D0%B5_%D1%80%D0%B0%D1%81%D0%BF%D1%80%D0%B5%D0%B4%D0%B5%D0%BB%D0%B5%D0%BD%D0%B8%D0%B5_%D0%BF%D0%B0%D0%BC%D1%8F%D1%82%D0%B8) и все переменные [объектного типа](https://ru.wikipedia.org/w/index.php?title=%D0%9E%D0%B1%D1%8A%D0%B5%D0%BA%D1%82%D0%BD%D1%8B%D0%B9_%D1%82%D0%B8%D0%BF&action=edit&redlink=1) являются [ссылками](https://ru.wikipedia.org/wiki/%D0%A1%D1%81%D1%8B%D0%BB%D0%BA%D0%B0_%28%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5%29). Управление памятью осуществляется с помощью интегрированной автоматической [сборки мусора](https://ru.wikipedia.org/wiki/%D0%A1%D0%B1%D0%BE%D1%80%D0%BA%D0%B0_%D0%BC%D1%83%D1%81%D0%BE%D1%80%D0%B0_%28%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5%29), которую выполняет [JVM](https://ru.wikipedia.org/wiki/Java_Virtual_Machine). [5]

Достоинством подобного способа выполнения программ является полная независимость байт-кода от [операционной системы](https://ru.wikipedia.org/wiki/%D0%9E%D0%BF%D0%B5%D1%80%D0%B0%D1%86%D0%B8%D0%BE%D0%BD%D0%BD%D0%B0%D1%8F_%D1%81%D0%B8%D1%81%D1%82%D0%B5%D0%BC%D0%B0) и [оборудования](https://ru.wikipedia.org/wiki/%D0%90%D0%BF%D0%BF%D0%B0%D1%80%D0%B0%D1%82%D0%BD%D0%B0%D1%8F_%D0%BF%D0%BB%D0%B0%D1%82%D1%84%D0%BE%D1%80%D0%BC%D0%B0), что позволяет выполнять Java-приложения на любом устройстве, для которого существует соответствующая виртуальная машина. Другой важной особенностью технологии Java является гибкая система безопасности, в рамках которой исполнение программы полностью контролируется виртуальной машиной. Любые операции, которые превышают установленные полномочия программы (например, попытка несанкционированного доступа к данным или соединения с другим компьютером), вызывают немедленное прерывание.

Часто к недостаткам концепции виртуальной машины относят снижение производительности. Ряд усовершенствований несколько увеличил скорость выполнения программ на Java:

* применение технологии трансляции байт-кода в машинный код непосредственно во время работы программы ([JIT](https://ru.wikipedia.org/wiki/JIT)-технология) с возможностью сохранения версий класса в машинном коде;
* широкое использование [платформенно-ориентированного кода](https://ru.wikipedia.org/wiki/%D0%9F%D0%BB%D0%B0%D1%82%D1%84%D0%BE%D1%80%D0%BC%D0%B5%D0%BD%D0%BD%D0%BE-%D0%BE%D1%80%D0%B8%D0%B5%D0%BD%D1%82%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%BD%D1%8B%D0%B9_%D0%BA%D0%BE%D0%B4) (native-код) в стандартных библиотеках;
* аппаратные средства, обеспечивающие ускоренную обработку байт-кода (например, технология [Jazelle](https://ru.wikipedia.org/wiki/Jazelle), поддерживаемая некоторыми процессорами фирмы [ARM](https://ru.wikipedia.org/wiki/ARM_%28%D0%BA%D0%BE%D0%BC%D0%BF%D0%B0%D0%BD%D0%B8%D1%8F%29)).

По данным для семи разных задач время выполнения на Java составляет в среднем в полтора-два раза больше, чем для C/C++, в некоторых случаях Java быстрее, а в отдельных случаях в 7 раз медленнее. С другой стороны, для большинства из них потребление памяти Java-машиной было в 10—30 раз больше, чем программой на C/C++. Также примечательно исследование, проведённое компанией [Google](https://ru.wikipedia.org/wiki/Google_%28%D0%BA%D0%BE%D0%BC%D0%BF%D0%B0%D0%BD%D0%B8%D1%8F%29), согласно которому отмечается существенно более низкая производительность и бо́льшее потребление памяти в тестовых примерах на Java в сравнении с аналогичными программами на [C++](https://ru.wikipedia.org/wiki/C%2B%2B). [6]

Программы, написанные на Java, имеют репутацию более медленных и занимающих больше оперативной памяти, чем написанные на языке C. Тем не менее, скорость выполнения программ, написанных на языке Java, была существенно улучшена с выпуском в 1997—1998 годах так называемого JIT-компилятора в версии 1.1 в дополнение к другим особенностям языка для поддержки лучшего анализа кода (такие, как внутренние классы, класс StringBuffer, упрощенные логические вычисления и т. д.). Кроме того, была произведена оптимизация виртуальной машины Java — с 2000 года для этого используется виртуальная машина [HotSpot](https://ru.wikipedia.org/wiki/HotSpot). По состоянию на февраль 2012 года, код Java 7 приблизительно в 1.8 раза медленнее кода, написанного на языке Си. [7]

Некоторые платформы предлагают аппаратную поддержку выполнения для Java. К примеру, микроконтроллеры, выполняющие код Java на аппаратном обеспечении вместо программной JVM, а также основанные на ARM процессоры, которые поддерживают выполнение байткода Java через опцию Jazelle.

Python— [высокоуровневый язык программирования](https://ru.wikipedia.org/wiki/%D0%92%D1%8B%D1%81%D0%BE%D0%BA%D0%BE%D1%83%D1%80%D0%BE%D0%B2%D0%BD%D0%B5%D0%B2%D1%8B%D0%B9_%D1%8F%D0%B7%D1%8B%D0%BA_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D1%8F) общего назначения, ориентированный на повышение производительности разработчика и читаемости кода. [Синтаксис](https://ru.wikipedia.org/wiki/%D0%A1%D0%B8%D0%BD%D1%82%D0%B0%D0%BA%D1%81%D0%B8%D1%81_%28%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5%29) ядра Python минималистичен. В то же время [стандартная библиотека](https://ru.wikipedia.org/wiki/%D0%A1%D1%82%D0%B0%D0%BD%D0%B4%D0%B0%D1%80%D1%82%D0%BD%D0%B0%D1%8F_%D0%B1%D0%B8%D0%B1%D0%BB%D0%B8%D0%BE%D1%82%D0%B5%D0%BA%D0%B0_Python) включает большой объём полезных функций.

Python поддерживает несколько [парадигм программирования](https://ru.wikipedia.org/wiki/%D0%9F%D0%B0%D1%80%D0%B0%D0%B4%D0%B8%D0%B3%D0%BC%D0%B0_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D1%8F), в том числе [структурное](https://ru.wikipedia.org/wiki/%D0%A1%D1%82%D1%80%D1%83%D0%BA%D1%82%D1%83%D1%80%D0%BD%D0%BE%D0%B5_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5), [объектно-ориентированное](https://ru.wikipedia.org/wiki/%D0%9E%D0%B1%D1%8A%D0%B5%D0%BA%D1%82%D0%BD%D0%BE-%D0%BE%D1%80%D0%B8%D0%B5%D0%BD%D1%82%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%BD%D0%BE%D0%B5_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5), [функциональное](https://ru.wikipedia.org/wiki/%D0%A4%D1%83%D0%BD%D0%BA%D1%86%D0%B8%D0%BE%D0%BD%D0%B0%D0%BB%D1%8C%D0%BD%D0%BE%D0%B5_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5), [императивное](https://ru.wikipedia.org/wiki/%D0%98%D0%BC%D0%BF%D0%B5%D1%80%D0%B0%D1%82%D0%B8%D0%B2%D0%BD%D0%BE%D0%B5_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5) и [аспектно-ориентированное](https://ru.wikipedia.org/wiki/%D0%90%D1%81%D0%BF%D0%B5%D0%BA%D1%82%D0%BD%D0%BE-%D0%BE%D1%80%D0%B8%D0%B5%D0%BD%D1%82%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%BD%D0%BE%D0%B5_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5). Основные архитектурные черты — [динамическая типизация](https://ru.wikipedia.org/wiki/%D0%94%D0%B8%D0%BD%D0%B0%D0%BC%D0%B8%D1%87%D0%B5%D1%81%D0%BA%D0%B0%D1%8F_%D1%82%D0%B8%D0%BF%D0%B8%D0%B7%D0%B0%D1%86%D0%B8%D1%8F), [автоматическое управление памятью](https://ru.wikipedia.org/wiki/%D0%A1%D0%B1%D0%BE%D1%80%D0%BA%D0%B0_%D0%BC%D1%83%D1%81%D0%BE%D1%80%D0%B0_%28%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5%29), полная [интроспекция](https://ru.wikipedia.org/wiki/%D0%98%D0%BD%D1%82%D1%80%D0%BE%D1%81%D0%BF%D0%B5%D0%BA%D1%86%D0%B8%D1%8F_%28%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5%29), механизм [обработки исключений](https://ru.wikipedia.org/wiki/%D0%9E%D0%B1%D1%80%D0%B0%D0%B1%D0%BE%D1%82%D0%BA%D0%B0_%D0%B8%D1%81%D0%BA%D0%BB%D1%8E%D1%87%D0%B5%D0%BD%D0%B8%D0%B9), поддержка [многопоточных вычислений](https://ru.wikipedia.org/wiki/%D0%9C%D0%BD%D0%BE%D0%B3%D0%BE%D0%BF%D0%BE%D1%82%D0%BE%D1%87%D0%BD%D0%BE%D1%81%D1%82%D1%8C) и удобные высокоуровневые [структуры данных](https://ru.wikipedia.org/wiki/%D0%A1%D1%82%D1%80%D1%83%D0%BA%D1%82%D1%83%D1%80%D0%B0_%D0%B4%D0%B0%D0%BD%D0%BD%D1%8B%D1%85). Код в Python организовывается в функции и [классы](https://ru.wikipedia.org/wiki/%D0%9A%D0%BB%D0%B0%D1%81%D1%81_%28%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5%29), которые могут объединяться в [модули](https://ru.wikipedia.org/wiki/%D0%9C%D0%BE%D0%B4%D1%83%D0%BB%D1%8C_%28%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5%29) (они в свою очередь могут быть объединены в пакеты).

Эталонной реализацией Python является интерпретатор [CPython](https://ru.wikipedia.org/wiki/CPython), поддерживающий большинство активно используемых платформ. Он распространяется под [свободной лицензией](https://ru.wikipedia.org/wiki/%D0%A1%D0%B2%D0%BE%D0%B1%D0%BE%D0%B4%D0%BD%D0%BE%D0%B5_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%BD%D0%BE%D0%B5_%D0%BE%D0%B1%D0%B5%D1%81%D0%BF%D0%B5%D1%87%D0%B5%D0%BD%D0%B8%D0%B5) Python Software Foundation License, позволяющей использовать его без ограничений в любых приложениях, включая [проприетарные](https://ru.wikipedia.org/wiki/%D0%9F%D1%80%D0%BE%D0%BF%D1%80%D0%B8%D0%B5%D1%82%D0%B0%D1%80%D0%BD%D0%BE%D0%B5_%D0%9F%D0%9E). Есть реализации интерпретаторов для [JVM](https://ru.wikipedia.org/wiki/JVM) (с возможностью [компиляции](https://ru.wikipedia.org/wiki/%D0%9A%D0%BE%D0%BC%D0%BF%D0%B8%D0%BB%D1%8F%D1%82%D0%BE%D1%80)), [MSIL](https://ru.wikipedia.org/wiki/MSIL) (с возможностью [компиляции](https://ru.wikipedia.org/wiki/%D0%9A%D0%BE%D0%BC%D0%BF%D0%B8%D0%BB%D1%8F%D1%82%D0%BE%D1%80)), [LLVM](https://ru.wikipedia.org/wiki/LLVM) и других. Проект [PyPy](https://ru.wikipedia.org/wiki/PyPy) предлагает реализацию Python на самом Python, что уменьшает затраты на изменения языка и постановку экспериментов над новыми возможностями. [8]

Python — активно развивающийся [язык программирования](https://ru.wikipedia.org/wiki/%D0%AF%D0%B7%D1%8B%D0%BA_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D1%8F), новые версии (с добавлением/изменением языковых свойств) выходят примерно раз в два с половиной года. Вследствие этого и некоторых других причин на Python отсутствуют [стандарт ANSI](https://ru.wikipedia.org/wiki/%D0%90%D0%BC%D0%B5%D1%80%D0%B8%D0%BA%D0%B0%D0%BD%D1%81%D0%BA%D0%B8%D0%B9_%D0%BD%D0%B0%D1%86%D0%B8%D0%BE%D0%BD%D0%B0%D0%BB%D1%8C%D0%BD%D1%8B%D0%B9_%D0%B8%D0%BD%D1%81%D1%82%D0%B8%D1%82%D1%83%D1%82_%D1%81%D1%82%D0%B0%D0%BD%D0%B4%D0%B0%D1%80%D1%82%D0%BE%D0%B2), [ISO](https://ru.wikipedia.org/wiki/ISO) или другие официальные стандарты, их роль выполняет [CPython](https://ru.wikipedia.org/wiki/CPython).Python [портирован](https://ru.wikipedia.org/wiki/%D0%9F%D0%BE%D1%80%D1%82%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%BD%D0%BE%D0%B3%D0%BE_%D0%BE%D0%B1%D0%B5%D1%81%D0%BF%D0%B5%D1%87%D0%B5%D0%BD%D0%B8%D1%8F) и работает почти на всех известных платформах — от [КПК](https://ru.wikipedia.org/wiki/%D0%9A%D0%B0%D1%80%D0%BC%D0%B0%D0%BD%D0%BD%D1%8B%D0%B9_%D0%BF%D0%B5%D1%80%D1%81%D0%BE%D0%BD%D0%B0%D0%BB%D1%8C%D0%BD%D1%8B%D0%B9_%D0%BA%D0%BE%D0%BC%D0%BF%D1%8C%D1%8E%D1%82%D0%B5%D1%80) до [мейнфреймов](https://ru.wikipedia.org/wiki/%D0%9C%D0%B5%D0%B9%D0%BD%D1%84%D1%80%D0%B5%D0%B9%D0%BC). Существуют порты под [Microsoft Windows](https://ru.wikipedia.org/wiki/Microsoft_Windows), практически все варианты [UNIX](https://ru.wikipedia.org/wiki/UNIX) (включая [FreeBSD](https://ru.wikipedia.org/wiki/FreeBSD) и [Linux](https://ru.wikipedia.org/wiki/Linux)), [Plan 9](https://ru.wikipedia.org/wiki/Plan_9), [Mac OS](https://ru.wikipedia.org/wiki/Mac_OS) и [Mac OS X](https://ru.wikipedia.org/wiki/Mac_OS_X), [iPhone OS](https://ru.wikipedia.org/wiki/IPhone_OS) 2.0 и выше, [Palm OS](https://ru.wikipedia.org/wiki/Palm_OS), [OS/2](https://ru.wikipedia.org/wiki/OS/2), [Amiga](https://ru.wikipedia.org/wiki/Amiga), [HaikuOS](https://ru.wikipedia.org/wiki/HaikuOS), [AS/400](https://ru.wikipedia.org/wiki/AS/400) и даже [OS/390](https://ru.wikipedia.org/wiki/OS/390), [Windows Mobile](https://ru.wikipedia.org/wiki/Windows_Mobile), [Symbian](https://ru.wikipedia.org/wiki/Symbian) и [Android](https://ru.wikipedia.org/wiki/Android).По мере устаревания платформы её поддержка в основной ветви языка прекращается. Например, с серии 2.6 прекращена поддержка [Windows 95](https://ru.wikipedia.org/wiki/Windows_95), [Windows 98](https://ru.wikipedia.org/wiki/Windows_98) и [Windows ME](https://ru.wikipedia.org/wiki/Windows_ME). Однако на этих платформах можно использовать предыдущие версии Python — на данный момент сообщество активно поддерживает версии Python начиная от 2.3 (для них выходят исправления).

При этом, в отличие от многих портируемых систем, для всех основных платформ Python имеет поддержку характерных для данной платформы технологий (например, Microsoft [COM](https://ru.wikipedia.org/wiki/Microsoft_Component_Object_Model)/[DCOM](https://ru.wikipedia.org/wiki/Microsoft_Distributed_Component_Object_Model)). Более того, существует специальная версия Python для [виртуальной машины Java](https://ru.wikipedia.org/wiki/%D0%92%D0%B8%D1%80%D1%82%D1%83%D0%B0%D0%BB%D1%8C%D0%BD%D0%B0%D1%8F_%D0%BC%D0%B0%D1%88%D0%B8%D0%BD%D0%B0_Java) — [Jython](https://ru.wikipedia.org/wiki/Jython), что позволяет интерпретатору выполняться на любой системе, поддерживающей Java, при этом классы Java могут непосредственно использоваться из Python и даже быть написанными на Python. Python поддерживает [динамическую типизацию](https://ru.wikipedia.org/wiki/%D0%94%D0%B8%D0%BD%D0%B0%D0%BC%D0%B8%D1%87%D0%B5%D1%81%D0%BA%D0%B0%D1%8F_%D1%82%D0%B8%D0%BF%D0%B8%D0%B7%D0%B0%D1%86%D0%B8%D1%8F), то есть тип переменной определяется только во время исполнения. Поэтому вместо «присваивания значения переменной» лучше говорить о «связывании значения с некоторым именем». Все значения являются объектами, в том числе функции, методы, модули, классы.

# Описание принятых проектных решений

## Структура и функционирование приложения

Приложение использует клиент серверную архитектуру по TCP протоколу для реализации сетевого режима игры. Сервер синхронизируют данные между клиентскими устройствами.

Общая структура клиент-серверного взаимодействия представлена на рисунке 2.1. при которой серверное приложение, работающее на выделенной машине принимает запросы от сервера и обрабатывает их в асинхронном режиме, реализуя шаблон реактора.

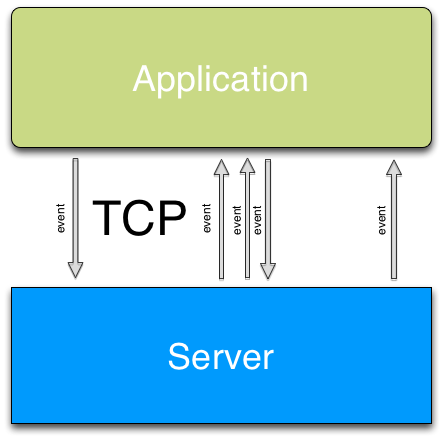


Рисунок 2.1. Клиент серверное взаимодействие

После успешной обработки происходит сериализация измененного состояние системы в json обьект и передается на сторону клиента позволяя клиенту отобразить новое состояние для пользователя.

В случае возникновения ошибки при вычислении состояния сервера, или попытки пользователем выполнения недопустимой операции, клиенту отправляется код ошибки совместно с сообщением.

На самом примитивном уровне абстракции приложение, ориентированное на работу с сервером состоит из следующих архитектурных слоев:

* ядро приложения, которое включает в себя компоненты системы, не доступные для взаимодействия с пользователем;
* графический пользователь интерфейс;
* компоненты повторного использования: библиотеки, визуальные компоненты и другое;
* ресурсы приложения: графические файлы, звуки, необходимые бинарные файлы.

Наиважнейшим условием построение стрессоустойчивой архитектуры является отделение ядра системы от GUI, настолько, что б одно, могло успешно функционировать без другого. Ядро приложения, состоит из следующих слоев:

* (Start layer) Стартовый слой, определяющий рабочий процесс, начала исполнения программы;
* (Network layer) Сетевой слой, обеспечивающий механизм транспортного взаимодействия;
* (API layer) Слой API, обеспечивающий единую систему команд взаимодействия между клиентом и сервером;
* (Network Cache Layer) Слой сетевого кэширования, обеспечивающий ускорения сетевого взаимодействия клиента и сервера;
* (Validation Items Layer) Слой валидации данных полученных из сети;
* (Network Items Layer) Слой сущности данных передаваемых по сети;
* (Data Model) Модель данных, обеспечивающая взаимодействие сущностей данных;
* (Local cache layer) Слой локального кеширования, обеспечивающий локальный доступ к уже полученным сетевым ресурсам;
* (Workflow layer) Слой рабочих процессов, включающий классы и алгоритмы специфичные для данного приложения;
* (Local storage) Локальное хранилище.

Одна из основных задач заключается в том, чтобы обеспечить взаимно независимое функционирование указанных слоев. Каждый слой должным обеспечивать только выполнение возложенных на него функций. [9] Как правило, слой находящийся на более высоком уровне иерархии не имеет представление о специфике реализации других слоев. В том числе и обработка ошибок сетевого соединения которая осуществляется виртуальными методами на следующих уровнях приложения.Постановка задачи позволяет выделить несколько подзадач, которые могут быть описаны отдельными классами, такими как класс загрузки данных из сети:

- проверка полученных данных;

- сохранение данных в постоянном хранилище;

- вычисление данных;

- фильтрация данных по указанным критериям (настройки приложения);

- класс старта приложения.

После запуска приложения на выполнения, производится создание (инстанциирование) объекта, отвечающего за загрузку данных (создание независимого потока под управлением ОС) и начинает процесс. Главный контроллер приложения отображает главное меню,

По окончании загрузки данных, создается объект-валидатор и объект-провайдер локального хранилища. В случае если данные прошли необходимую валидацию, они могут быть переданы провайдеру локального хранилища. Для отображения графика, создается объект локального хранилища и объект настроек данных. Настройки данных передаются в провайдер локального хранилища для извлечения данных с установленными фильтрами. Для проведения вычислений создается главный игровой объект (mainGameObject), и объекты операций. В объект передаются данные полученные с в результате операций пользователя, и один из двух объектов операций, которые знают, как именно осуществить вычисления над игровым объектом.

Приложение начинает функционирование с запуска объекта делегатного класса. Его назначение — принять и передать вызовы системы приложению, а также, осуществить первоначальную конфигурацию GUI приложения. Все алгоритмы и механизмы, которые не относятся к старту приложения, или получения сообщений от системы в отдельные классы. Сразу после завершения первоначальной конфигурации управление передано классу, который осуществляет остальные операции настройки приложения: реконфигурирование интерфейса в зависимости от условий, первоначальную загрузку данных.

Сетевой уровень (Network Layer) абстракции обеспечивает базовые алгоритмы транспортного уровня передачи сообщений от клиента к серверу, и получению от него необходимой информации. Как правило, сообщения могут передаваться в форматах JSON. Кроме того, каждое сообщение может иметь заголовок со служебной информацией. Например, там может быть описана длительность хранения запроса / ответа в кеше приложения. Network Layer не имеет никакого представления об приложением серверах, или о его системе команд.

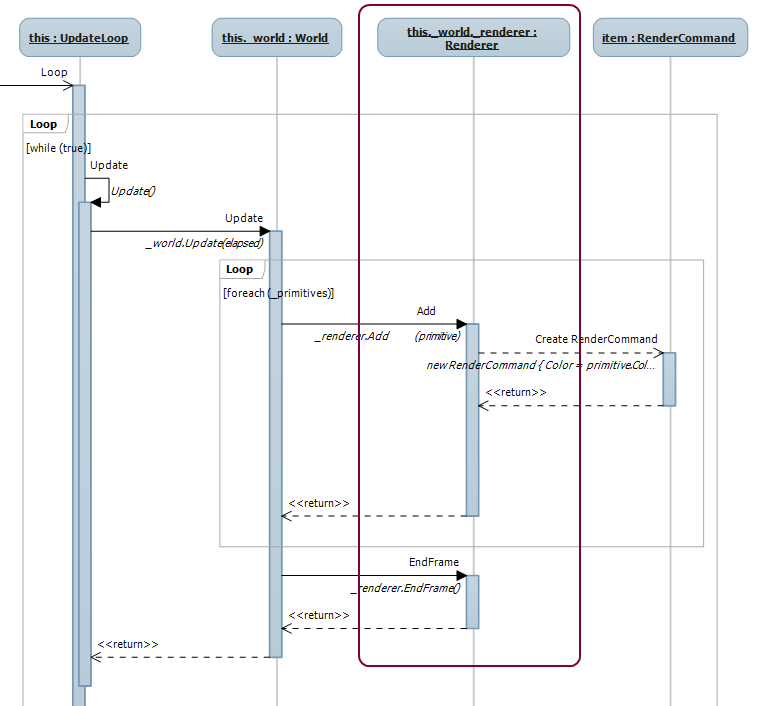


Рисунок 2.2 Диаграмма последовательности графической системы приложения

Диаграмма последовательности ([*англ.*](https://ru.wikipedia.org/wiki/%D0%90%D0%BD%D0%B3%D0%BB%D0%B8%D0%B9%D1%81%D0%BA%D0%B8%D0%B9_%D1%8F%D0%B7%D1%8B%D0%BA) sequence diagram)  на рисунке 2.2 [диаграмма](https://ru.wikipedia.org/wiki/%D0%94%D0%B8%D0%B0%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B0), на которой для набора классов и игровых объектов на примере системы отображения графики и рендера графических спрайтов приложения на единой временной оси. Показаны жизненный цикл (создание-деятельность-уничтожение) и взаимодействие (отправка запросов и получение ответов) внутри бесконечного цикла while True loop. Основными элементами диаграммы последовательности являются Frame обозначения [объектов](https://ru.wikipedia.org/wiki/%D0%9E%D0%B1%D1%8A%D0%B5%D0%BA%D1%82_%28%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5%29), вертикальные «линии жизни» ([англ.](https://ru.wikipedia.org/wiki/%D0%90%D0%BD%D0%B3%D0%BB%D0%B8%D0%B9%D1%81%D0%BA%D0%B8%D0%B9_%D1%8F%D0%B7%D1%8B%D0%BA) lifeline) и уничтожения обьектов, помеченых как return, отображающие течение времени, прямоугольники, отражающие деятельность объекта или исполнение им определенной функции, и стрелки, показывающие обмен сигналами или сообщениями между объектами. На данной диаграмме объекты располагаются слева направо. Внутри while loop происходит отображение (World.Render) обьектов после их обновления внутри UpdateLoop.

Обработка ошибок сетевого соединения осуществляется виртуальными методами на следующих уровнях приложения. Задача этого слоя только осуществить вызов метода обработки и передать в него полученную из сети информацию.  
 Кроме того, перед непосредственным запросом информации из сети, network layer опрашивает локальный кеш, и в случае присутствия там ответа сразу же возвращает его пользователю. TCP Socket — наиболее низкоуровневый подход, включающий в себя синхронные и асинхронные запросы, и имеющий возможность работать как с TCP, так и с UDP подключениями [10]. Реализации TCP обычно встроены в [ядра](https://ru.wikipedia.org/wiki/%D0%AF%D0%B4%D1%80%D0%BE_%D0%BE%D0%BF%D0%B5%D1%80%D0%B0%D1%86%D0%B8%D0%BE%D0%BD%D0%BD%D0%BE%D0%B9_%D1%81%D0%B8%D1%81%D1%82%D0%B5%D0%BC%D1%8B) [ОС](https://ru.wikipedia.org/wiki/%D0%9E%D0%BF%D0%B5%D1%80%D0%B0%D1%86%D0%B8%D0%BE%D0%BD%D0%BD%D0%B0%D1%8F_%D1%81%D0%B8%D1%81%D1%82%D0%B5%D0%BC%D0%B0). Существуют реализации TCP, работающие в [пространстве пользователя](https://ru.wikipedia.org/wiki/%D0%9F%D1%80%D0%BE%D1%81%D1%82%D1%80%D0%B0%D0%BD%D1%81%D1%82%D0%B2%D0%BE_%D0%BF%D0%BE%D0%BB%D1%8C%D0%B7%D0%BE%D0%B2%D0%B0%D1%82%D0%B5%D0%BB%D1%8F). Порт источника идентифицирует приложение клиента, с которого отправлены пакеты. Ответные данные передаются клиенту на основании этого номера. Когда осуществляется передача от компьютера к компьютеру через Интернет, TCP работает на верхнем уровне между двумя конечными системами, например, [браузером](https://ru.wikipedia.org/wiki/%D0%91%D1%80%D0%B0%D1%83%D0%B7%D0%B5%D1%80) и веб-сервером. TCP осуществляет надежную передачу потока байтов от одной программы на некотором компьютере к другой программе на другом компьютере. TCP контролирует длину сообщения, скорость обмена сообщениями, сетевой трафик. Позволяет выполнять любые сетевые операции, и не ограничивает ни по формату, ни по размеру данных. Не добавляет дополнительного веса сообщениям.

Слой кеширования задействуется для ускорения сетевого обмена между клиентом и сервером на уровне Android SDK. Выбор ответов осуществляется стороной лежащей за пределами контроля системы, и не гарантирует снижение сетевого трафика, но ускоряет его.

Именно на этот слое лежит ответственность за сериализация данных из JSON в десериализированное JAVA представление. Данный слой используется для описания классов, осуществляющих объектное или объектно-реляционное преобразование. Используется стандартный набор библиотек доступных в языках.

Все реализованные алгоритмы, которые не относятся к слоям ядра, и не представляют собой GUI вынесены в классы специфических последовательностей рабочих процессов. Каждый из этих процессов оформляется в стиле, и подключается к основной части приложения путем добавления ссылок на экземпляр соответствующего класса в GUI. В подавляющем большинстве случаев, все эти процессы являются не визуальными. Однако имеются некоторые исключения, например, когда необходимо осуществить длинную последовательность предопределенных кадров анимации, с заданными алгоритмами отображения Вызывающий код имеет минимальные знания об этой функциональности существуют довольно развитые рабочие процессы, логика функционирования которых зависит от внутреннего состояния. Такие процессы реализованы при помощи паттернов «Стратегия» и «Машина состояний». Один из часто используемых процессов — процесс перемещения персонажа — очевидный претендент на использование паттерна «машины состояний».

Каждый вызов слоев ядра сопровождается передачей объекта обратного вызова (callback), и именно через него должно быть возвращено управление в приложение при успешном выполнении команды или возникновения ошибок. Ни в коем случае не должен допускаться неявный вызов слоев ядра, объектами рабочей последовательности.

Учитывая, что нельзя допускать, чтоб какие-либо универсальные визуальные элементы управления зависели от состояния рабочих последовательностей. Так критически необходимо, такие элементы управления должны быть объявлены приватными последовательностью. Доступ к состоянию управления может осуществляться через свойства самих элементов, наследование и переопределение методов, и, в крайнем случае, через реализацию методов.

## C:\Users\tsh\AppData\Local\Microsoft\Windows\INetCache\Content.Word\seq_diag_libgdx_screen-manager-uml.pngСхема клиента приложения

Рисунок 2.3 Управление экранами в приложении

На рисунке 2.3 изображенна реализацию управления экрана в игре. Интерфейс представляет автономные экраны, используемые в игре, и переключение между ними может быть автоматическим или может зависеть от некоторого взаимодействия с пользователем. Вместе с экраном наследуется возможность использовать абстрактный класс как точку входа основного проекта LibGDX.  
 Подход заключается в создании особого класса ответсвенного за создание, доступа и переключения различных игровых экранов.

Реализазия интерфейса Screen дает доступ к контролируемости, и параллельному доступу будущего класса MyGdxGame, который использует перечисление экранов и переключается между по необходимости.

Конструктор класа реализации сделан приватным и создание обьекта выполняется вне пределах класса. Единственное место где обьекст собственно и создается это статический метод getInstance() где он создается по необходимости, так называемая ленивая инициализация, и кешируется для дальнейшего использования. Собственно сам экран переключает между своими обьектами в классе MyLibgdxGame поэтому ScreenManager зависит от этого обьекта. Так же ScreenManager ответсвенен за создание, кеширование и переключение и самое важное освобождение ресурсов тех экранов которые уже были отображенны и больше не используются, поэтому используется ограничение доступа к реализации Screen.

Общая диаграмма классов приложения представлена на рисунке 2.4. На диаграмме изображено отношение между всеми классами приложения, взаимные вызовы, наследование и реализация интерфейсов. Вкратце опишем задачу каждого из класов:

- CharacterObject – Базовый класс для всех персонажей игры;

- Demoman, Scout –основные персонажи игры, содержит уникальные звуки и спрайты. Так и алгоритмы описывающие правила их взаимодействия;

- EmptyGameObject Обьект пустого поля игры;

- EndGameScreen Класс, реализующий отображение экрана после победы одной из команд;

- GameObject Базовый игровой класс;

- GameObjectManager Менеджер отвечающий за управления всеми игровыми обьектами;

- GameScreen Обьект ответсвенный за отображения игрового экрана;

- Langs В этом обьекте хранятся языковые настройки игры;

- MainMenuScreen Отображает главное игровое меню;

- Map Отвечает за отображение игровой карты;

- MyGameUi Интерфейс пользователя;

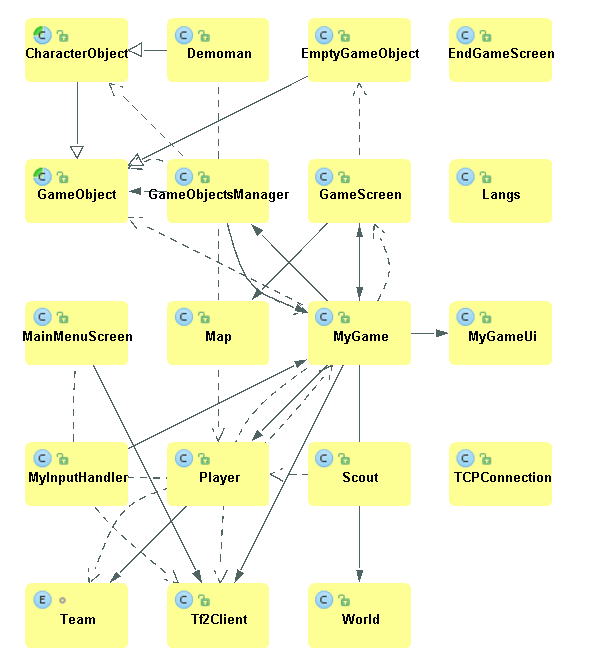
- Team Обьект отвественный за управление командами;

- MyInputHandler Обработка данных о нажатии на экран;

- Tf2Client Код клиента реализующий передачу указаний другим обьектам;

- Player Обькт игрока, содержащий данные о команде и его персонажах

- TCPConnection Обьект ответсвенный за сетевое взаимодействие и синхронизацию данных по сети.

Рисунок 2.4 Диаграмма классов приложения

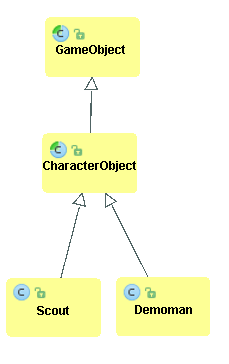


Рисунок 2.5. Диаграмма классов игровых персонажей.

На диаграмме 2.5, отображены [классы](https://ru.wikipedia.org/wiki/%D0%9A%D0%BB%D0%B0%D1%81%D1%81_%28%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5%29) системы, их [атрибуты](https://ru.wikipedia.org/wiki/%D0%9F%D0%BE%D0%BB%D0%B5_%D0%BA%D0%BB%D0%B0%D1%81%D1%81%D0%B0), [методы](https://ru.wikipedia.org/wiki/%D0%9C%D0%B5%D1%82%D0%BE%D0%B4_%28%D1%8F%D0%B7%D1%8B%D0%BA%D0%B8_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D1%8F%29) и [взаимосвязи](https://ru.wikipedia.org/wiki/%D0%9E%D0%B1%D1%8A%D0%B5%D0%BA%D1%82%D0%BD%D0%BE-%D0%BE%D1%80%D0%B8%D0%B5%D0%BD%D1%82%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%BD%D0%BE%D0%B5_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5) между ними. Статический вид диаграммы рассматривает логические взаимосвязи классов между собой. Аналитический вид диаграммы рассматривает общий вид и взаимосвязи классов, входящих в систему. Относительно концептуальной точки зрения — диаграмма классов игровых персонажей описывает модель предметной области, в ней присутствуют только классы прикладных объектов. С точки зрения реализации — диаграмма классов игровых персонажей отображены классы, используемые непосредственно в программном коде.

Обобщение (Generalization) показывает, что один из двух связанных классов например Scout (подтип) является частной формой другого (надтипа) CharacterObject, который называется обобщением первого. Обобщение также известно как наследование или «[is a](https://ru.wikipedia.org/w/index.php?title=Is-a&action=edit&redlink=1)» взаимосвязь между Scout/Demoman и GameObject.

Каждый из персонажей содержит уникальный спрайт для отображения на карте и уникальный набор звуков. Остальные свойства наследуются от базовых классов как например метод render ответственный за отображения объекта на экране определён в базовом классе GameObject рисунок 2.5, следовательно, персонажам нет необходимости знать низкоуровневые детали реализации вывода на экран.

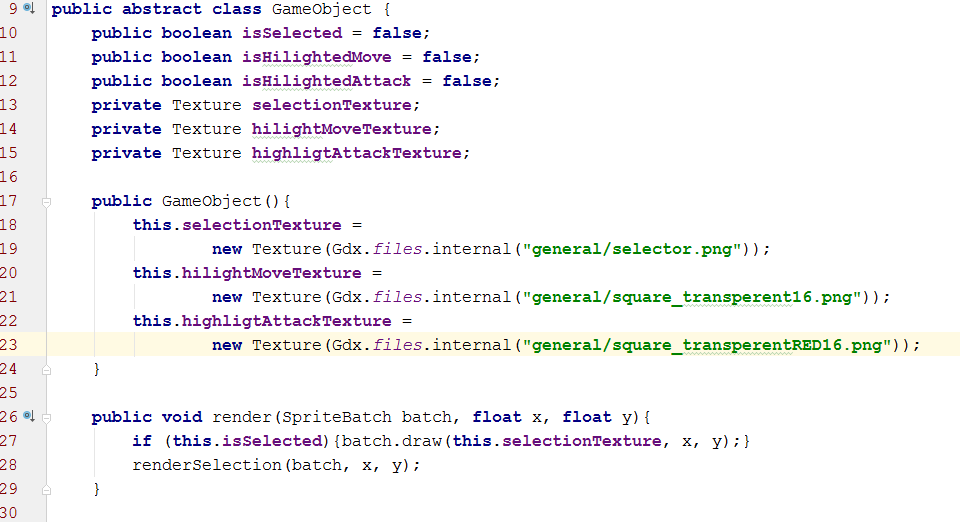


Рисунок 2.6. Определение базового класса игрового объекта

На рисунке 2.6 отображена реализация суперкласса GameObject — это класс, не имеющий суперкласса, и поэтому находится в основании дерева подклассов. от которого унаследованы все остальные классы обьектов в игре.Суперклассом или родительским классом назовем GameObject, на основе которого создаются другие классы. Классы, полученные на основе текущей реализации суперкласса, называются дочерними классами, производными классами или [подклассами](https://ru.wikipedia.org/wiki/%D0%9F%D0%BE%D0%B4%D0%BA%D0%BB%D0%B0%D1%81%D1%81_%28%D0%B8%D0%BD%D1%84%D0%BE%D1%80%D0%BC%D0%B0%D1%82%D0%B8%D0%BA%D0%B0%29).

Суперкласс GameObject позволяет создавать обобщенный интерфейс, между игровыми обьектами по всему приложению мобильной игры заключающий в себе настраиваемую функциональность за счет использования [виртуальных функций](https://ru.wikipedia.org/wiki/%D0%92%D0%B8%D1%80%D1%82%D1%83%D0%B0%D0%BB%D1%8C%D0%BD%D1%8B%D0%B9_%D0%BC%D0%B5%D1%82%D0%BE%D0%B4).

На рисунке 2.6 изображен пример функций render() которая позволяет отобразить на экране текущий спрайт, учитвыя что GameObject класс является базовым все его наследники могут реализовать этот метод и получить достук к графической системе.



Рисунок 2.7 Инициализация игровых менеджеров.

На рисунке 2.7 изображена диаграмма последовательности и очередность создания всех основным менеджеров в игре. Внутри точки входа main, осуществляется основной вызов метода createManager() который и производит всю работу по их созданию.

Каждый менеджер ответственен за управление массивом принадлежащих ему объектов и на более высоком уровне обладает набором необходимых атрибутов и методов для управления группой объектов. Методы и объекты вынесенные в каждый из менеджеров выбирались по принципу обобщения, т.е каждый из этих методов управляет группой подобных обьектов и поэтому не может быть реализован как метод каждого отдельного класса без нарушения принципа не повторения кода. Поэтому для упрощения последующего сопровождения программы все методы были вынесены из классов которые инициализируют обьекты в общие менеджеры, специально созданные для управления группой. Это позволяет упростить общую структуру приложения из за того что классы приложения будут выполнять четко определенные функциии уникальные для этого класса.

# Описание работы приложения

## Начало работы с сервером.

Для запуска сервера необходимо в командной строке набрать python server.py

После этого произойдет запуск сервера, который готов принимать соединения на 6789 порту по TCP протоколу. Результат выполнения команды приведен на рисунке 3.1.

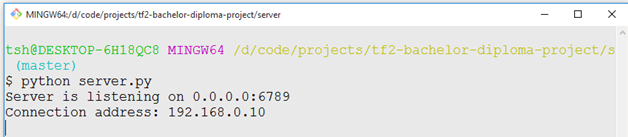


Рисунок 3.1. Запуск сервера

Когда к серверному приложению подключается мобильный клиент, на окно терминала будет выведено информационное сообщение об этом событии, где помимо информации о событии содержится уникальный идентификационный номер клиента, который ему присвоила система. Результат подключения клиента отображен на рисунке 3.2.

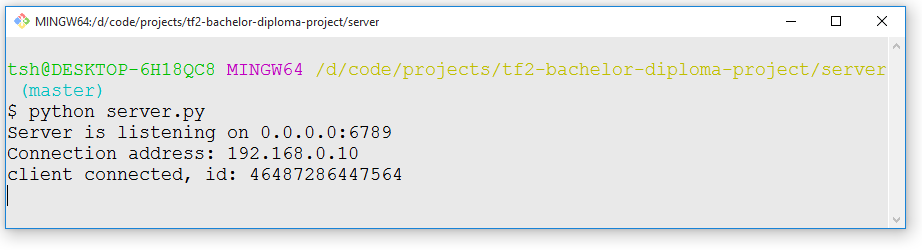


Рисунок 3.2 Подключение клиента к серверу.

Впоследствии этот уникальный номер используется как ключ в хэш таблице для уникальной идентификации клиента. Как значение по этому ключу хранится сам сетевой объект.

## Работа с клиентом

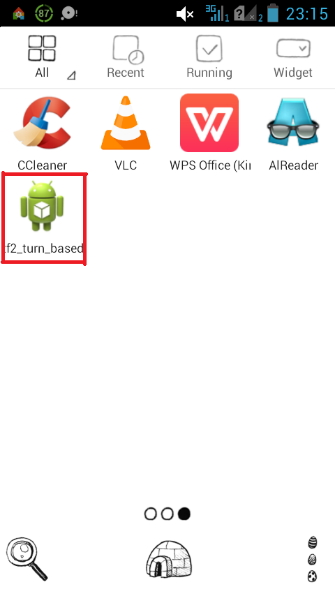
 После установки приложения на мобильное устройство среди установленных программ будет отображенна иконка установленного приложения.

Рисунок 3.3 Установленное приложение

Для запуска клиентского приложения на мобильном устройстве необходимо выбрать иконку приложения. Как изображено на рисунке 3.3. Еденичное нажатие на сенсорный экран над иконкой позволяет запустить мобильное приложения.

После выбора приложения, произойдет неявное тестирование графических и звуковых возможностей устройства, и в случае успеха появится окно главного меню как изображено на рисунке 3.4. Непосредственной тестирование все устройств производится набором менеджеров которые были созданы на этапе инициализации приложения. Все менеджеры обладают доступом к низкоуровневым функция операционной системы что позволяет с достаточной точностью определить возможности текущего приложения и поддержку текущего устройства для используемых библиотек. После успешного запуска приложения будет отображенно меню с экрана которого пользователю будет предложенно выбрать одно из возможных действий.

На экране главного меню игрок может совершать такие действия:

- начать игру выбрав пункт “Start game”;

- перейти в меню настроек выбрав пункт “Settings”;

- выйти из игры выбрав пункт “Quit”.



Рисунок 3.4 Экран главного меню.

После выбора пункта Start Game открывается главный экран игры изображенный на рисунке 3.5. С этого экрана осуществляется основное взаимодействие с игрой. Здесь же происходит выбор персонажа. Первый ход получает команда синих, это видно по цвету полосы по правой стороне, цвет которой изменяется в зависимости от того какая команда сейчас имеет право действия. По углам расположены персонажи которыми можно совершать действия, ведущие к победе в игре.

Для выбора персонажа необходимо один раз нажать на желаемый спрайт после чего вокруг этого персонажа появится маркер выделитель отображающий какой персонаж сейчас является активным. А так же осуществляется подсветка доступных действий персонажа на карте, как показано на рисунке 3.6

После выделения персонажа клетки, на которые он имеет право ходить выделяются цветом. Каждый персонаж имеет различную длину хода, и дальность атаки. Возможные клетки для атаки подсвечиваются более темным цветом относительно цвета шага. Персонажи того же цвета не могут быть атакованы даже если они находятся в зоне атаки.



Рисунок 3.5 Экран игры.



Рисунок 3.6 Возможные действия персонажа.

Если персонаж противоположного цвета находится в диапазоне атаки, то он может быть уничтожен. Для этого после выделения атакующего персонажа необходимо выбрать атакуемого персонажа, и если враг находится в пределах области действия он снимается с доски.

Персонаж не может перемещается в позицию, занятую другим персонажем независимо от его статуса, т.е. враждебен ли объект или принадлежит игроку. При попытке совершить такое действие, будет проигран характерный звук оповещающий о недопустимости такой операции. В нашем случае на рисунке 3.6 выделенный персонаж, отмечен желтыми полосами на границе клетки не может переместится в нижний левый угол из-за того, что там уже находится союзный персонаж.

Рисунок 3.7 Экран победы красной команды

После того как все персонажи противника были убраны с игрового поля, оставшийся игрок считается победителем, после чего игра меняет экран игрового поля на победный экран с логотипом победившей команды рисунок 3.7.

# Тестирование программного продукта, анализ опытной эксплуатации

## Основные понятия в тестировании

Тестирование программного обеспечения (Software Testing) – это проверка соответствия между реальным и ожидаемым поведением программы, осуществляемая на конечном наборе тестов, выбранном определенным образом [IEEE Guide to Software Engineering Body of Knowledge, SWEBOK, 2004].

В более широком смысле, тестирование – это одна из техник контроля качества, включающая в себя действия по планированию работ (Test Management), проектированию тестов (Test Design), выполнению тестирования (Test Execution) и анализу полученных результатов (Test Analysis).

Верификация (Verification) – это процесс оценки системы или её компонент с целью определения соответствия результатов текущего этапа разработки условиям, сформированным в начале этого этапа [IEEE].

Валидация (Validation) – это определение соответствия разрабатываемого ПО ожиданиям и потребностям пользователя, требованиям к системе [BS7925-1].

План Тестирования (Test Plan) – это документ, описывающий весь, объем работ по тестированию, начиная с описания объекта, стратегии, расписания, критериев начала и окончания тестирования, до необходимого в процессе работы оборудования, специальных знаний, а также оценки рисков с вариантами их разрешения.

Тестовый случай (Test Case) – это алгоритм, описывающий совокупность шагов, конкретных условий и параметров, необходимых для проверки реализации тестируемой функции или её части.

Тестовый набор (Test Suite) – это совокупность тест-кейсов, объединенных по некоторому общему признаку. Например, тест-кейсы для проверки функционала компоненты, используемой в разных формах приложения.

Баг / Дефект Репорт (BugReport) – это документ, описывающий ситуацию или последовательность действий, приведшую к некорректной работе объекта тестирования, с указанием причин и ожидаемого результата.

Объект тестирования – это отдельная логическая единица (форма, подсистема) со всеми элементами управления и диалогами.

## Инструментарий

Для автоматизации тестирования [11] рекомендуется использовать:

JUnit — [библиотека](https://ru.wikipedia.org/wiki/%D0%91%D0%B8%D0%B1%D0%BB%D0%B8%D0%BE%D1%82%D0%B5%D0%BA%D0%B0_%28%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5%29) для [модульного тестирования](https://ru.wikipedia.org/wiki/%D0%9C%D0%BE%D0%B4%D1%83%D0%BB%D1%8C%D0%BD%D0%BE%D0%B5_%D1%82%D0%B5%D1%81%D1%82%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5) программного обеспечения на языке [Java](https://ru.wikipedia.org/wiki/Java).

ZeNmap, XSpider, Metasploit - сканеры безопасности для исследования топологии сети, поиска уязвимостей в сетевой инфраструктуре приложения и их эксплуатации.

При ручном тестировании приложений рекомендуется использовать вспомогательные инструменты для исследования приложения:

Snagit - для снятия скриншотов и записи действий на экране.

Intercepter-NG, WinDump и др. – снифферы для перехвата и анализа сетевого траффика.

## Категории тестов

Рекомендуется последовательно разрабатывать отдельные тест-комплекты следующих категорий:

* для основного Init-теста в тестовом наборе добавить используемые во всех тестах глобальные переменные;
* для основного Init-теста в тестовом наборе добавить проверку корректности ресурсов и их наличия;
* для основного Init-теста в тестовом наборе добавить проверку того, что все обязательные элементы интерфейса главной формы приложения присутствуют и доступны;
* Init-тесты (шаблоны) для всех остальных тестов, разрабатываемые перед непосредственной реализацией основных проверок, должны содержать блок команд для корректного входа и выхода из приложения.

Функциональные регрессионные тесты для проверки корректности, решенных в системе bug/task-трекинга задач и отсутствия ранее имеющихся ошибок.

Обязательные действия в данной категории:

* добавить проверки важных задач task-ов (имеющих критичный функционал или влияющих на логику работы приложения);
* добавить проверки всех задач bug-ов, поставленных в системе bug/task-трекинга.

Функциональные позитивные тесты для проверки корректности завершения разрешенных (валидных) операций (например, проверка того, что после получения команды перемещения объект действительно изменил позицию), добавляются в соответствующие Init-тесты[12]. Обязательные действия в данной категории:

* добавить проверки наличия ключевых элементов интерфейса (все кнопки и надписи на своих местах);
* добавить проверки фактического выполнения операций: объекты меняют положение, логика приложения выполняется, звуковая система работает как полагается;
* добавить проверки наличия сообщений об успехе операций.

Функциональные негативные тесты для проверки корректной обработки неправильно введенных данных (например, проверка того, что при попытке совершения не разрешенного действия возникает сообщение об ошибке), добавляются в соответствующие Init-тесты.

Обязательные действия в данной категории:

* добавить проверки корректности обработки недопустимых или некорректных данных для операций: при таких данных программа не должна допускать их сохранения, редактирования или обновления игрового состояние не имея прав для его изменения;
* добавить проверки наличия сообщений о некорректных действиях пользователя или неверных исходных данных.

Функциональные end-to-end сценарии, имитирующие конкретную последовательность действий реального пользователя (например, проверка корректного выполнения последовательности действий по входу в систему, открытию нужной формы, выходу из системы), добавляются в соответствующие Init-тесты.

Обязательные действия в данной категории:

* добавить несколько различных вариантов завершенных и логичных действий пользователя при работе с приложением.

Нагрузочные тесты, для проверки возможности "нормальной" и корректной работы в системе определенного числа пользователей, выполняющих типичные действия (для имитации работы пользователей возможно использовать, например, уже имеющиеся CRUD-тесты или end-to-end сценарии) [13].

Количественные показатели измеряются для случаев:

* работы приложения без нагрузки (для определения сравнительных характеристик);
* работы приложения под равномерно увеличивающейся нагрузкой (с различным числом виртуальных пользователей или различным числом подключений к узлам приложения);
* работы приложения под пиковой нагрузкой (с различным числом виртуальных пользователей или различным числом подключений к узлам приложения).

Обязательные действия в данной категории:

* определить цели и объекты нагрузки;
* выбрать подходы и методики тестирования нагрузки;
* определить количественные и качественные показатели для нагрузочного тестирования (например, измерить среднее время доступа к приложению, данным и основным подсистемам приложения, степень загрузки процессоров, сетевых адаптеров, файла подкачки, памяти серверов приложений).

## Журнал тестирования и таблица проверок

Для проверки сервиса был создан журнал тестирования. Ниже приведена таблица проверки и оценки входных и выходных параметров [6].

Таблица 4.1 – Таблица проверки и оценки входных и выходных параметров

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| № | Входной параметр | Выходной параметр | | Оценка качества | |
| 1 | Запуск сервиса | Открытие окна | | Отлично | |
| 2 | Вызов меню | Открытие окна меню | | Отлично | |
| 5 | Кнопка «Начать игру» | Переход экран игры | | Отлично | |
| 6 | Кнопка «Выход» | Завершения работы приложения | | Отлично | |
| 7 | Кнопка «Настройки» | Отображения экрана с настройками | | Отлично | |
| 8 | Нажатие на персонажа | Подсветка возможных действий | | Отлично | |
| 9 | Повторное нажатие на персонажа | Снятие любых выделений, снятие подсветки | | Отлично | |
| 10 | Нажатие на пустом поле | Перемещение персонажа в указанную позицию | | Отлично | |
| 11 | Завершение хода | Передача управления другому игроку | | Отлично | |
| 12 | Попытка совершить действие во время хода другого игрока | Сообщение об ошибке | | Отлично | |
| 13 | Атака союзника | | Сообщение об ошибке | | Отлично |
| 14 | Атака противника | | Персонаж противника снимается с поля | | Отлично |
| 15 | Все персонажи противника уничтожены | | Отображение победного экрана | | Отлично |

Как видно из табл. 4.1 приложение показало устойчивую работу по всем выходным параметрам. Оценить качество работы приложения можно на «Отлично».

Данное приложение может быть передано заказчикам для использования.

# Выводы

В рамках аттестационной работы бакалавра была разработана мобильная игра с клиент серверной архитектурой, в которой игрок управляет группой персонажей против команды управляемой другим игроком по сети.

Данное приложение осуществляет управление графической системой мобильных устройств для отображения графики и осуществляет коммуникацию с сервером по протоколу TCP. Сервер приложение осуществляет синхронизацию нескольких игровых клиентов.

В разработанном приложении имеется необходимая форма для совершение действий игровых персонажей, и система обратной связи с пользователем для его оповещения о действиях другого игрока. В случае недоступности сервера, система может работать в автономном режиме.

Приложение имеет простой и удобный интерфейс. В качестве мобильного устройства рекомендуется использовать устройство с версией android 4.0 или выше

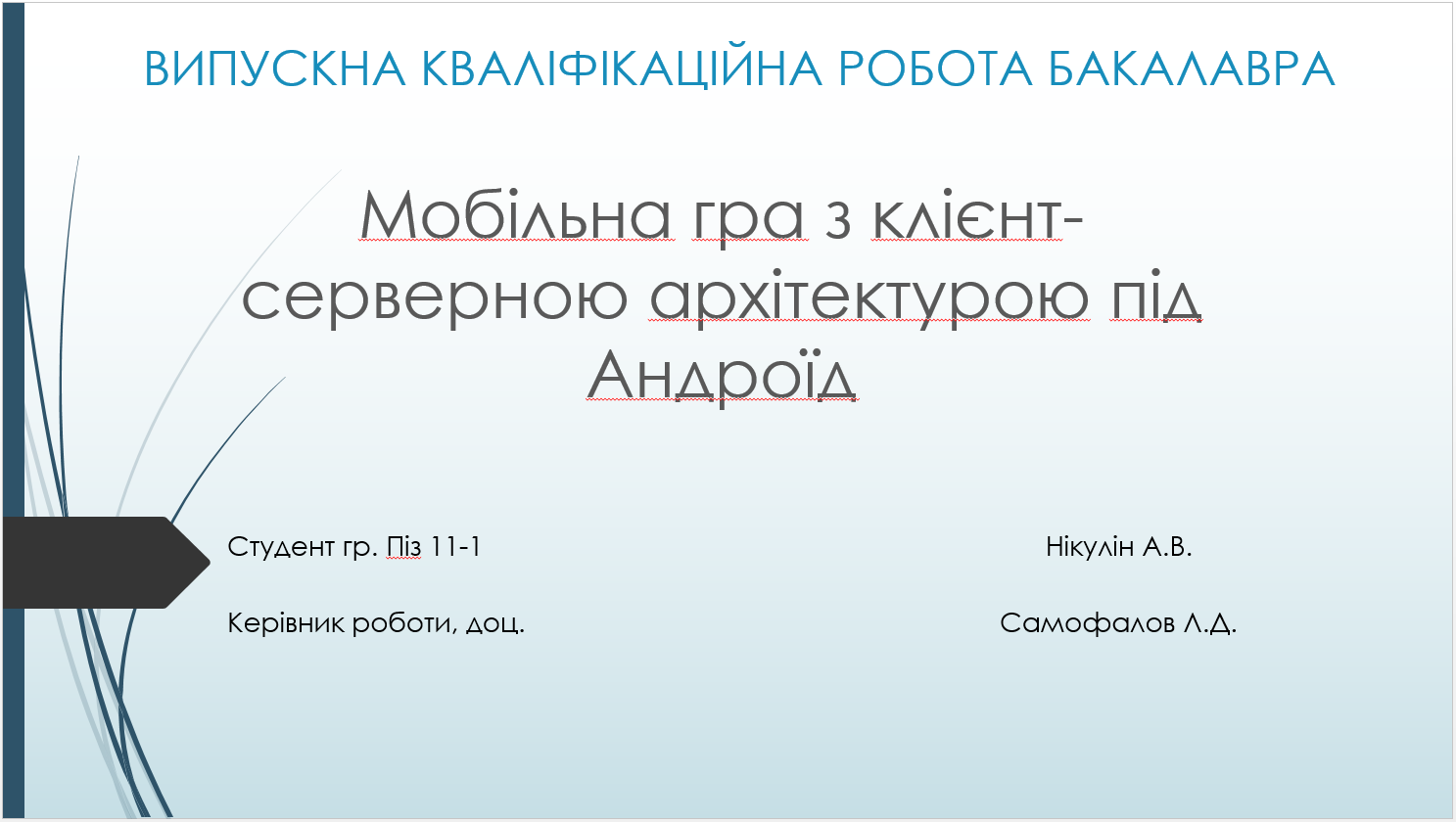
Программный продукт имеет простой и удобный интерфейс. Система предназначена для работы в различных устройствах под управлением ОС android.

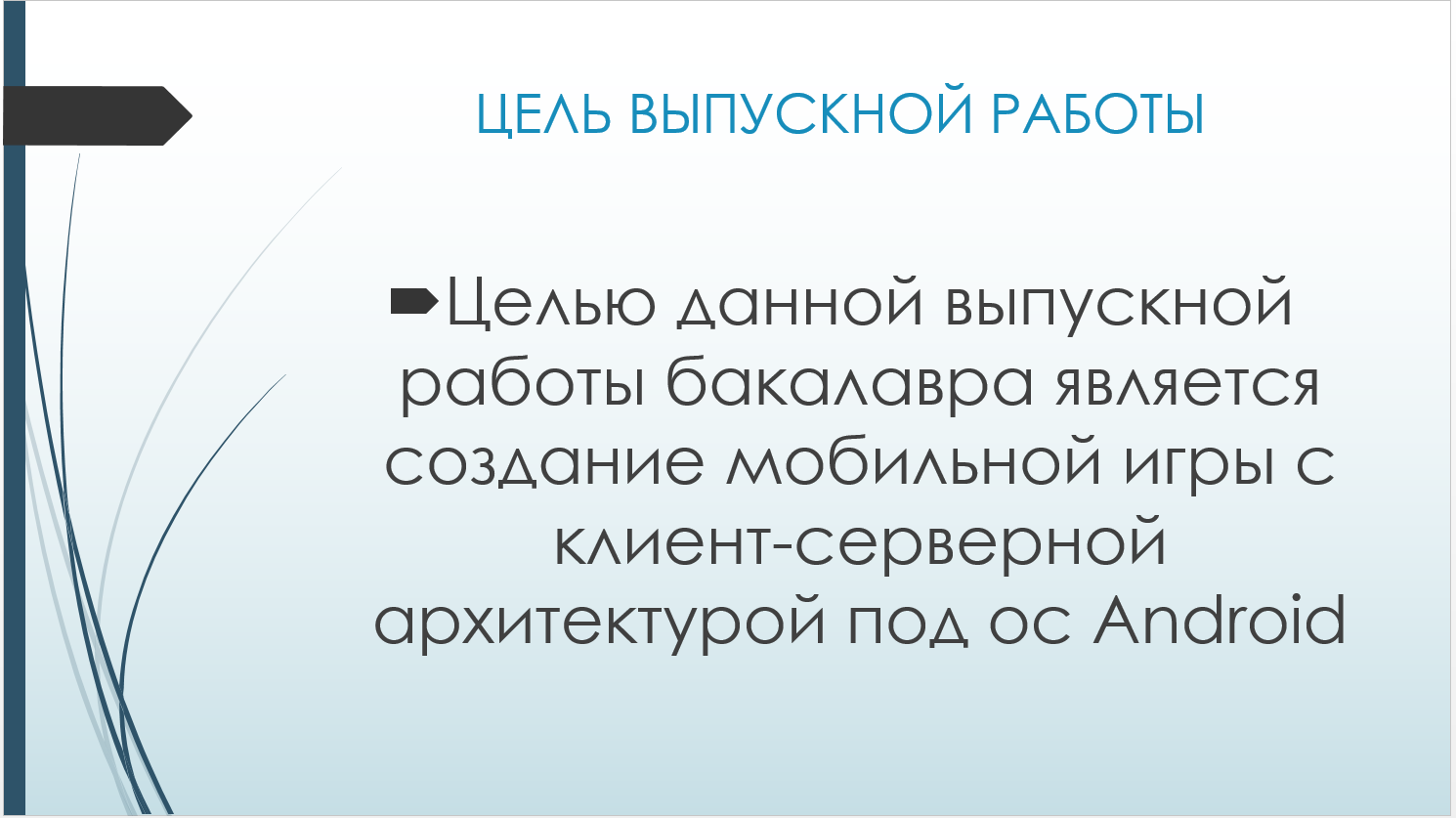
Тестирование разработанной системы показало её высокую работоспособность и устойчивость к ошибочным действиям пользователя.

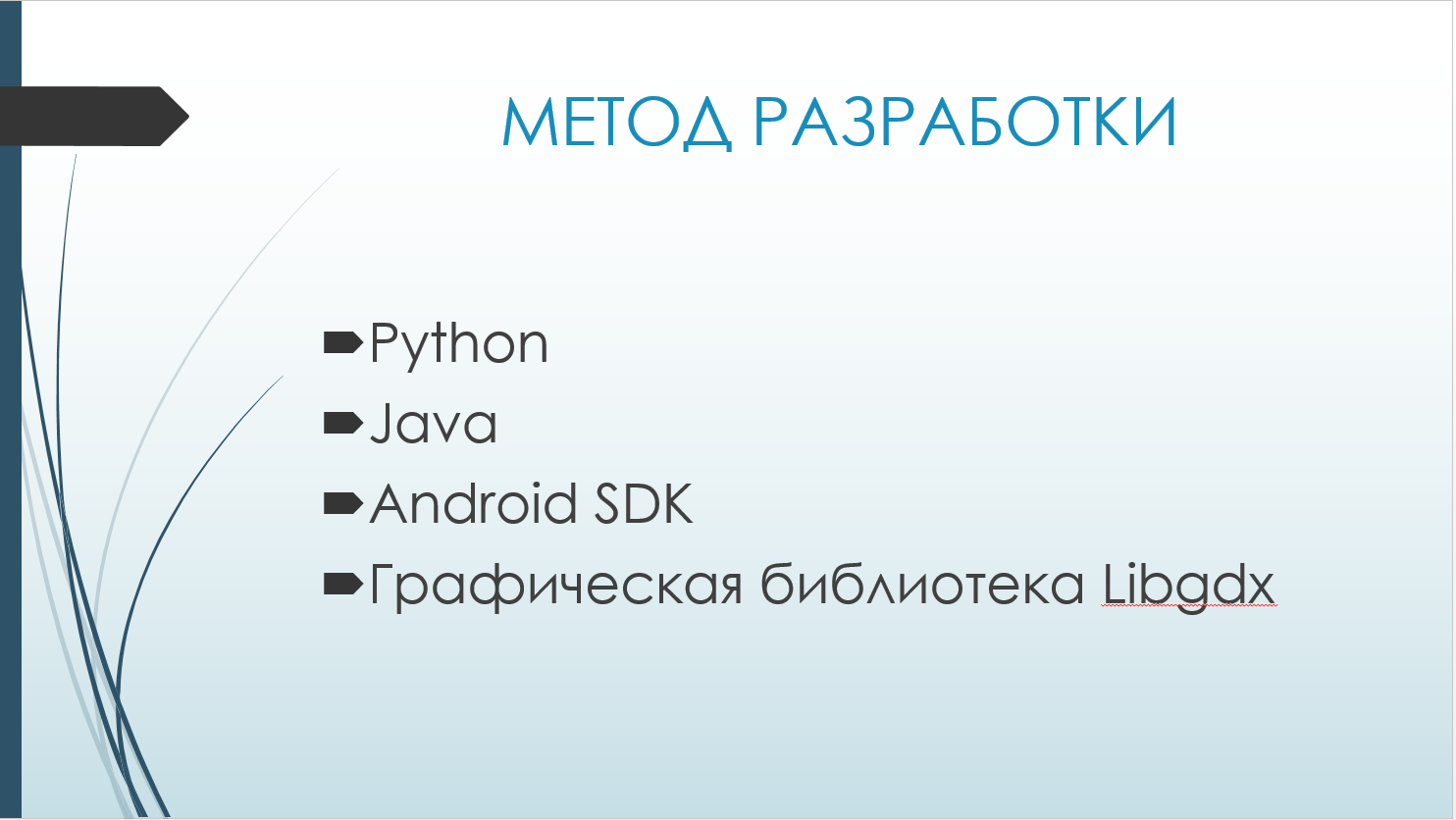
# Список источников

1. Веб-застосунок [Електронний ресурс] - Режим доступу: www/ URL: -<http://www.forbes.com/sites/tristanlouis/2013/08/10/how-much-do-average-apps-make/#68a7a8512cb3> - 25.03.2016 р.- Загл. з екрану.
2. Д.Моррис. Проектирование и архитектура игр [Текст] / Д.Моррис, - BHV-СПб, 2012. – 639с
3. А.Ламот. Проектирование и разработка игр. [Текст] / - А.Ламот САМС, 2010. – 700с
4. Р.Седжвик. Алгоритмы на С++. [Текст] / Р.Седжвик, - Adisson Wisley, 2015. – 1100с
5. С.Скиена. Алгоритмы. Дизайн и анализ. [Текст] / С.Скиена, - BHV-СПб, 2014. – 700с
6. К. Балакришаннаир. Изучаем LibGDX [Текст] / К. Балакришаннаир, - OReilly, 2016. – 380с
7. Ювал Босу. Основы Libgdx [Текст] / Ювал Босу, - Packt Publishing Ltd 2014. – 472с
8. Александро Родас. Программирование игр на python [Текст] / Александро Родас, - Packt Publishing Ltd 2015. – 520с
9. Дж. Харт. Основы геймдизайна [Текст] / Дж. Харт, - BHV-СПб, 2010. – 600с
10. А.Ламот. Архитектура сетевых игр. [Текст] / А.Ламот, - САМС, 2012. – 400с
11. Харитонова Елена. Поиск уязвимостей в программах с помощью анализаторов кода. [Электронный ресурс]; Режим доступу: [http://www.codenet.ru/progr/other/ code-analysers.php](http://www.codenet.ru/progr/other/%20code-analysers.php) - 15.04.2016 р.- Загл. с экрана.
12. Тестирование запросов/ответов с Windows PowerShell. [Електронний ресурс]; Режим доступу: http://msdn.microsoft.com/ru-ru/magazine/cc507642.aspx - 2.04.2016 р.- Загл. с экрана
13. Статическое и динамическое тестирование [Електронний ресурс]; Режим доступу: http://www.quizful.net/interview/qa/static-vs-dynamic-testing - 2.04.2016 р.- Загл. с экрана

# Приложение А Слайды презентации







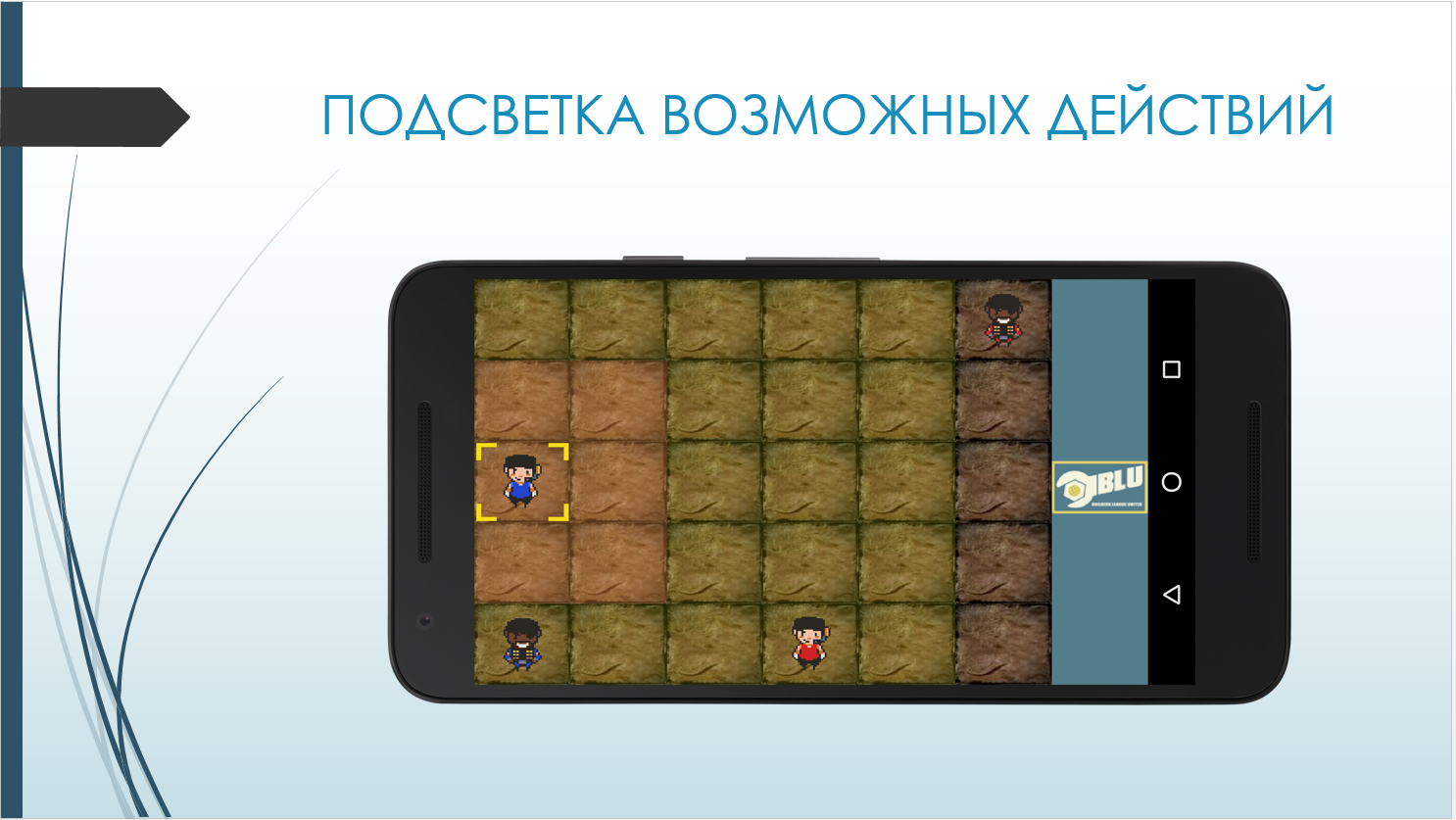




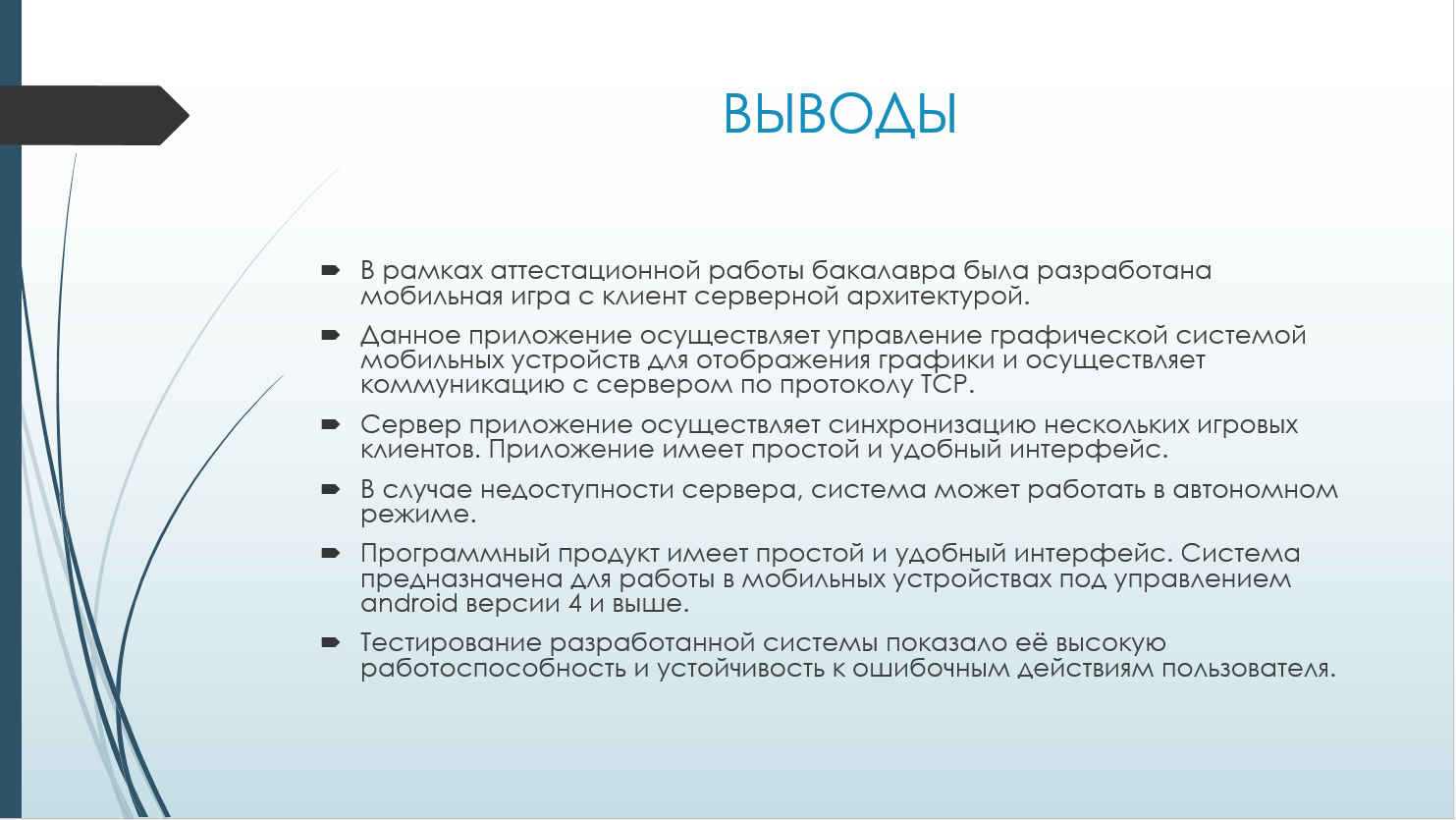












# ПРИЛОЖЕНИЕ Б ПРИМЕРЫ КОДОВ ПРОГРАММЫ

MyGame.java

**package** io.gameclient.game;  
  
**import** com.badlogic.gdx.graphics.g2d.SpriteBatch;  
  
**import** java.util.Random;  
  
**enum** Team { ***RED***, ***BLUE*** }  
  
**public class** MyGame {  
 Player **redPlayer**;  
 Player **bluePlayer**;  
 Player **currentPlayer**;  
 GameObjectsManager **gameObjectsManager**;  
 MyGameUi **ui**;  
 Tf2Client **game**;  
 GameScreen **gameScreen**;  
  
 **public** MyGame(Tf2Client gam, GameScreen gs){  
 **this**.**game** = gam;  
 **this**.**gameScreen** = gs;  
 **this**.**redPlayer** = **new** Player(Team.***RED***, **false**);  
 **this**.**bluePlayer** = **new** Player(Team.***BLUE***, **true**);  
 **this**.**currentPlayer** = **this**.**bluePlayer**;  
 **this**.**gameObjectsManager** = **new** GameObjectsManager(**this**);  
 **this**.**ui** = **new** MyGameUi();  
 }  
  
 **public void** endTurn(){  
 **gameObjectsManager**.unselectObjects();  
 **gameObjectsManager**.undoHighlightPattern();  
 *// switch players* **if** (**this**.**currentPlayer** == **this**.**redPlayer**){  
 **this**.**currentPlayer** = **this**.**bluePlayer**;  
 } **else** {  
 **this**.**currentPlayer** = **this**.**redPlayer**;  
 }  
 **this**.**currentPlayer**.**canPerformAction** = **true**;  
 **this**.**ui**.setActiveTeam(**currentPlayer**.**team**);  
 System.***out***.println(**currentPlayer**.**team** + **" "** + **currentPlayer**.**canPerformAction**);  
 *// Check win condition* **int** liveBlue = **this**.**gameObjectsManager**.getLiveTeamCharacterCount(Team.***BLUE***);  
 **int** liveRed = **this**.**gameObjectsManager**.getLiveTeamCharacterCount(Team.***RED***);  
 **if** (liveBlue == 0 ){  
 **this**.**game**.setScreen(**new** EndGameScreen(Team.***RED***, **this**.**game**));  
 dispose();  
 **gameScreen**.dispose();  
 }  
 **if** (liveRed == 0 ){  
 **this**.**game**.setScreen(**new** EndGameScreen(Team.***BLUE***, **this**.**game**));  
 dispose();  
 **gameScreen**.dispose();  
 }  
  
 }  
  
  
 **public void** performAction(GameObject selectedObject, GameObject clickedObject){  
 *// Chars only* **if** (!(selectedObject **instanceof** CharacterObject)){  
 **return**;  
 }  
 **if** (!canPerformAction((CharacterObject)selectedObject)){  
 **return**;  
 }  
 */\* Parse actions \*/  
 // move* **if** ((clickedObject **instanceof** EmptyGameObject)){  
 **if** (canMove((CharacterObject)selectedObject, (EmptyGameObject) clickedObject)) {  
 **this**.**gameObjectsManager**.moveObject(selectedObject, clickedObject);  
 ((CharacterObject) selectedObject).sayMove();  
 endTurn();  
 }  
 }  
 *// attack* **else if** ((clickedObject **instanceof** CharacterObject) && !(selectedObject == clickedObject)){  
 **if** (canAttack((CharacterObject)selectedObject, (CharacterObject)clickedObject)) {  
 **this**.**gameObjectsManager**.attackTarget(selectedObject, (CharacterObject) clickedObject);  
 ((CharacterObject) selectedObject).sayKill();  
 endTurn();  
 }  
 }  
 }  
  
  
 **public boolean** canPerformAction(CharacterObject character){  
 **return** character.**player** == **currentPlayer** && **currentPlayer**.**canPerformAction**;  
 }  
  
  
 **public boolean** canMove(CharacterObject movable, EmptyGameObject destination){  
 **float** distance = **this**.**gameObjectsManager**.getDistanceBetweenObjects(movable, destination);  
 **return** movable.canMove(distance);  
 };  
  
  
 **public boolean** canAttack(CharacterObject attacker, CharacterObject target){  
 **float** distance = **this**.**gameObjectsManager**.getDistanceBetweenObjects(attacker, target);  
 **return** attacker.canAttack(target, distance);  
 }  
  
  
 **public void** inputReceived(**float** worldX, **float** worldY){  
 **this**.**gameObjectsManager**.inputReceived(worldX, worldY);  
 }  
  
  
 **public void** render(SpriteBatch batch){  
 **this**.**gameObjectsManager**.render(batch);  
 **this**.**ui**.render(batch);  
 }  
  
 **public void** dispose(){  
 **this**.**ui**.dispose();  
 }  
}

GameObject.Java

**package** io.gameclient.game;  
  
**import** com.badlogic.gdx.Gdx;  
**import** com.badlogic.gdx.graphics.Color;  
**import** com.badlogic.gdx.graphics.Texture;  
**import** com.badlogic.gdx.graphics.g2d.SpriteBatch;  
  
  
**public abstract class** GameObject {  
 **public boolean isSelected** = **false**;  
 **public boolean isHilightedMove** = **false**;  
 **public boolean isHilightedAttack** = **false**;  
 **private** Texture **selectionTexture**;  
 **private** Texture **hilightMoveTexture**;  
 **private** Texture **highligtAttackTexture**;  
  
 **public** GameObject(){  
 **this**.**selectionTexture** = **new** Texture(Gdx.*files*.internal(**"general/selector.png"**));  
 **this**.**hilightMoveTexture** = **new** Texture(Gdx.*files*.internal(**"general/square\_transperent16.png"**));  
 **this**.**highligtAttackTexture** = **new** Texture(Gdx.*files*.internal(**"general/square\_transperentRED16.png"**));  
 }  
  
 **public void** render(SpriteBatch batch, **float** x, **float** y){  
 **if** (**this**.**isSelected**){batch.draw(**this**.**selectionTexture**, x, y);}  
 renderSelection(batch, x, y);  
 }  
  
 **public void** renderSelection(SpriteBatch batch, **float** x, **float** y) {  
 **if** (**this**.**isHilightedMove**) {batch.draw(**this**.**hilightMoveTexture**, x, y);}  
 **if** (**this**.**isHilightedAttack**) {batch.draw(**this**.**highligtAttackTexture**, x, y);}  
  
 };  
  
 **public void** setSelected(){  
 **this**.**isSelected** = **true**;  
 }  
  
 **public void** setUnselected(){  
 **this**.**isSelected** = **false**;  
 }  
  
 **public void** toggleSelected(){  
 **this**.**isSelected** = !**this**.**isSelected**;  
 }  
  
  
 **public void** setHilightedMoveTrue(){**this**.**isHilightedMove** = **true**;}  
 **public void** setHilightedMoveFalse(){**this**.**isHilightedMove** = **false**;}  
  
 **public void** setHighlightedAttackTrue(){**this**.**isHilightedAttack** = **true**;}  
 **public void** setHighlightedAttackFalse(){**this**.**isHilightedAttack** = **false**;}  
  
 **public void** dispose(){  
 **this**.**selectionTexture**.dispose();  
 **this**.**hilightMoveTexture**.dispose();  
 **this**.**highligtAttackTexture**.dispose();  
 };  
}

**package** io.gameclient.game;  
  
**import** com.badlogic.gdx.Gdx;  
**import** com.badlogic.gdx.Screen;  
**import** com.badlogic.gdx.audio.Music;  
**import** com.badlogic.gdx.graphics.Color;  
**import** com.badlogic.gdx.graphics.GL20;  
**import** com.badlogic.gdx.graphics.OrthographicCamera;  
**import** com.badlogic.gdx.graphics.Pixmap;  
**import** com.badlogic.gdx.graphics.Texture;  
**import** com.badlogic.gdx.graphics.g2d.BitmapFont;  
**import** com.badlogic.gdx.graphics.g2d.Gdx2DPixmap;  
**import** com.badlogic.gdx.graphics.g2d.TextureAtlas;  
**import** com.badlogic.gdx.scenes.scene2d.Actor;  
**import** com.badlogic.gdx.scenes.scene2d.Stage;  
**import** com.badlogic.gdx.scenes.scene2d.ui.Image;  
**import** com.badlogic.gdx.scenes.scene2d.ui.Skin;  
**import** com.badlogic.gdx.scenes.scene2d.ui.Table;  
**import** com.badlogic.gdx.scenes.scene2d.ui.TextButton;  
**import** com.badlogic.gdx.scenes.scene2d.utils.ChangeListener;  
**import** com.badlogic.gdx.utils.viewport.ScreenViewport;  
**import** com.badlogic.gdx.utils.viewport.StretchViewport;  
  
**public class** MainMenuScreen **implements** Screen {  
 **float WIDTH** = 800;  
 **float HEIGHT** = 480;  
  
 **final** Tf2Client **game**;  
 OrthographicCamera **camera**;  
  
 Music **backgroundMusic**;  
 Texture **backgroundTexture**;  
 Texture **logo**;  
  
 **private** Stage **stage**;  
 **private** Table **table**;  
 Skin **skin**;  
  
 Actor **root**;  
  
 **public** MainMenuScreen(**final** Tf2Client gam) {  
 **try** {  
 TCPConnection tcp = **new** TCPConnection();  
 } **catch** (Exception e){  
 {System.***out***.println(**"Connection error: "**+e);}  
 }  
  
 **game** = gam;  
  
 **camera** = **new** OrthographicCamera();  
 **camera**.setToOrtho(**false**, **WIDTH**, **HEIGHT**);  
  
 **backgroundMusic** = Gdx.*audio*.newMusic(Gdx.*files*.internal(**"audio/background/rocket\_jump\_waltz.mp3"**));  
 **backgroundMusic**.setVolume(0.5f);  
 **backgroundMusic**.setLooping(**true**);  
 **backgroundMusic**.play();  
  
 **this**.**backgroundTexture** = **new** Texture(Gdx.*files*.internal(**"ui/2fort\_800px.png"**));  
 **this**.**logo** = **new** Texture(**"ui/tf2\_logo.png"**);  
  
 **stage** = **new** Stage(**new** StretchViewport(640, 480));  
 Gdx.*input*.setInputProcessor(**stage**);  
  
 Pixmap pixmap = **new** Pixmap(1, 1, Pixmap.Format.***RGBA8888***);  
 pixmap.setColor(Gdx2DPixmap.***GDX2D\_FORMAT\_ALPHA***);  
 pixmap.fill();  
 **skin** = **new** Skin();  
 **skin**.add(**"white"**, **new** Texture(pixmap));  
 BitmapFont bFont = **new** BitmapFont();  
 BitmapFont.BitmapFontData bfData = bFont.getData();  
 bfData.setScale(2);  
 **skin**.add(**"default"**, bFont);  
  
 TextButton.TextButtonStyle textButtonStyle = **new** TextButton.TextButtonStyle();  
 textButtonStyle.**up** = **skin**.newDrawable(**"white"**);  
 textButtonStyle.**over** = **skin**.newDrawable(**"white"**, Color.***LIGHT\_GRAY***);  
 textButtonStyle.**font** = **skin**.getFont(**"default"**);  
 **skin**.add(**"default"**, textButtonStyle);  
  
 Table table = **new** Table();  
 table.setFillParent(**true**);  
 **stage**.addActor(table);  
 *// MENU* String startGameText = **""**;  
 String settingsText = **"Settings"**;  
 String quitGameText = **""**;  
  
 **switch** (**game**.**prefs**.getInteger(**"lang"**)){  
 **case** Langs.***EN***:  
 startGameText = **"Start Game"**;  
 quitGameText = **"Quit"**;  
 **break**;  
 **case** Langs.***FR***:  
 startGameText = **"Démarrer le jeu"**;  
 quitGameText = **"Quitter"**;  
 **break**;  
 **case** Langs.***DE***:  
 startGameText = **"Spiel beginnen"**;  
 quitGameText = **"Verlassen"**;  
 **break**;  
 }  
 **final** TextButton button = **new** TextButton(startGameText, **skin**);  
 **final** TextButton settingsButton = **new** TextButton(settingsText, **skin**);  
 **final** TextButton exitButton = **new** TextButton(quitGameText, **skin**);  
 table.add(button);  
 table.row();  
 table.add(settingsButton);  
 table.row();  
 table.add(exitButton).padTop(10);  
 table.left().pad(20);  
 table.setBounds(75, 30, Gdx.*graphics*.getWidth(), Gdx.*graphics*.getHeight());  
  
 button.addListener(**new** ChangeListener() {  
 **public void** changed(ChangeListener.ChangeEvent event, Actor actor) {  
 **game**.setScreen(**new** GameScreen(**game**));  
 dispose();  
 }  
 });  
  
 exitButton.addListener(**new** ChangeListener() {  
 **public void** changed(ChangeListener.ChangeEvent event, Actor actor) {  
 Gdx.*app*.exit();  
 }  
 });  
 }  
  
 @Override  
 **public void** render(**float** delta) {  
 **stage**.getBatch().begin();  
 **stage**.getBatch().draw(**this**.**backgroundTexture**, 0, 0);  
 **stage**.getBatch().draw(**this**.**logo**, 0, 0);  
 **stage**.getBatch().end();  
  
 **stage**.act(Math.*min*(Gdx.*graphics*.getDeltaTime(), 1 / 30f));  
 **stage**.draw();  
 }  
  
 @Override  
 **public void** resize(**int** width, **int** height) {  
 **stage**.getViewport().update(width, height, **true**);  
 }  
  
 @Override  
 **public void** show() {  
 }  
  
 @Override  
 **public void** hide() {  
 }  
  
 @Override  
 **public void** pause() {  
 }  
  
 @Override  
 **public void** resume() {  
 }  
  
 @Override  
 **public void** dispose() {  
 **stage**.dispose();  
 **backgroundMusic**.dispose();  
 **logo**.dispose();  
 **backgroundTexture**.dispose();  
 }  
}

**package** io.gameclient.game;  
  
  
**import** com.badlogic.gdx.graphics.g2d.SpriteBatch;  
**import** com.badlogic.gdx.math.Vector2;  
  
**public class** GameObjectsManager {  
 **public final** Integer **TILE\_WIDTH** = 100;  
 **public final** Integer **TILE\_HEIGHT** = 100;  
  
 **private** GameObject **currentlySelectedObject** = **null**;  
 **private** MyGame **game**;  
  
 GameObject[][] **gameObjectsMap**;  
  
 **public** GameObjectsManager(MyGame game){  
 **this**.**game** = game;  
 **this**.**gameObjectsMap** = **new** GameObject[][]{  
 {**new** Demoman(game.**bluePlayer**), **new** EmptyGameObject(), **new** EmptyGameObject(), **new** EmptyGameObject(), **new** Scout(game.**redPlayer**)},  
 {**new** EmptyGameObject(), **new** EmptyGameObject(), **new** EmptyGameObject(), **new** EmptyGameObject(), **new** EmptyGameObject()},  
 {**new** EmptyGameObject(), **new** EmptyGameObject(), **new** EmptyGameObject(), **new** EmptyGameObject(), **new** EmptyGameObject()},  
 {**new** EmptyGameObject(), **new** EmptyGameObject(), **new** EmptyGameObject(), **new** EmptyGameObject(), **new** EmptyGameObject(),},  
 {**new** EmptyGameObject(), **new** EmptyGameObject(), **new** EmptyGameObject(), **new** EmptyGameObject(), **new** EmptyGameObject()},  
 {**new** Scout(game.**bluePlayer**), **new** EmptyGameObject(), **new** EmptyGameObject(), **new** EmptyGameObject(), **new** Demoman(game.**redPlayer**)}  
 };  
 }  
  
 **public void** render(SpriteBatch batch){  
 **for** (**int** i = 0; i < **gameObjectsMap**.**length**; i++){  
 **for** (**int** j = 0; j < **gameObjectsMap**[i].**length**; j++){  
 GameObject go = **gameObjectsMap**[i][j];  
 go.render(batch, i \* **TILE\_WIDTH**, j \* **TILE\_HEIGHT**);  
 }  
 }  
 }  
  
 **public void** inputReceived(**float** worldX, **float** worldY){  
 **int** row = (**int**)Math.*floor*(worldX / **TILE\_WIDTH**);  
 **int** column = (**int**)Math.*floor*(worldY / **TILE\_HEIGHT**);  
 GameObject clickedObject = **this**.**gameObjectsMap**[row][column];  
 *// set selection* **if** (**currentlySelectedObject** == **null** || **currentlySelectedObject instanceof** EmptyGameObject){  
 clickedObject.setSelected();  
 *// highlight action pattern* **if** (clickedObject **instanceof** CharacterObject){ highlightPattern((CharacterObject) clickedObject);}  
 **currentlySelectedObject** = clickedObject;  
 }  
 *// unset selection* **else if** (**currentlySelectedObject** == clickedObject){  
 clickedObject.toggleSelected();  
 **if** (clickedObject **instanceof** CharacterObject){ undoHighlightPattern();}  
 **currentlySelectedObject** = **null**;  
 }  
  
 **this**.**game**.performAction(**currentlySelectedObject**, clickedObject);  
 }  
  
 **public void** moveObject(GameObject movedObject, **int** destinationRow, **int** destinationColumn){  
 **int**[] movedObjectPosition = getObjectPosition(movedObject);  
 **gameObjectsMap**[destinationRow][destinationColumn] = movedObject;  
 **gameObjectsMap**[movedObjectPosition[0]][movedObjectPosition[1]] = **new** EmptyGameObject();  
 }  
  
 **public void** moveObject(GameObject movedObject, GameObject destinationObject){  
 **int**[] movedObjectPosition = getObjectPosition(movedObject);  
 **int**[] destinationPosition = getObjectPosition(destinationObject);  
 **gameObjectsMap**[destinationPosition[0]][destinationPosition[1]] = movedObject;  
 **gameObjectsMap**[movedObjectPosition[0]][movedObjectPosition[1]] = **new** EmptyGameObject();  
 }  
  
 **public void** attackTarget(GameObject attacker, CharacterObject target){  
 removeCharacter(target);  
 }  
  
 **private void** removeCharacter(CharacterObject characterObject){  
 **int**[] position = getObjectPosition(characterObject);  
 **gameObjectsMap**[position[0]][position[1]] = **new** EmptyGameObject();  
  
 }  
  
 **public int**[] getObjectPosition(GameObject object){  
 **int**[] position = **new int**[2];  
 **for** (**int** i = 0; i < **gameObjectsMap**.**length**; i++){  
 **for** (**int** j = 0; j < **gameObjectsMap**[i].**length**; j++){  
 **if** (**gameObjectsMap**[i][j] == object){  
 position[0] = i;  
 position[1] = j;  
 **return** position;  
 }  
 }  
 }  
 **throw new** IllegalArgumentException(**"Object not found in GameObjectArray"**);  
 }  
  
 **public** Vector2 getObjectPositionVector(GameObject object){  
 Vector2 position = **new** Vector2(0.0f, 0.0f);  
 **for** (**int** i = 0; i < **gameObjectsMap**.**length**; i++){  
 **for** (**int** j = 0; j < **gameObjectsMap**[i].**length**; j++){  
 **if** (**gameObjectsMap**[i][j] == object){  
 position.**x** = i;  
 position.**y** = j;  
 **return** position;  
 }  
 }  
 }  
 **throw new** IllegalArgumentException(**"Object not found in GameObjectArray"**);  
 }  
  
 **public int** getLiveTeamCharacterCount(Team team){  
 **int** count = 0;  
 **for** (**int** i = 0; i < **gameObjectsMap**.**length**; i++){  
 **for** (**int** j = 0; j < **gameObjectsMap**[i].**length**; j++){  
 **if** (**gameObjectsMap**[i][j] **instanceof** CharacterObject){  
 CharacterObject chr = (CharacterObject)**gameObjectsMap**[i][j];  
 **if** (chr.**player**.**team** == team) {count += 1;}  
 }  
 }  
 }  
 **return** count;  
 }  
  
 **public void** highlightPattern(CharacterObject origin) {  
 **for** (**int** i = 0; i < **gameObjectsMap**.**length**; i++) {  
 **for** (**int** j = 0; j < **gameObjectsMap**[i].**length**; j++) {  
 **float** distance = getDistanceBetweenObjects(origin, **gameObjectsMap**[i][j]);  
 System.***out***.println(distance + **" "** + origin.getMaxMoveDistance() + **" "** + origin.getMaxAttackDistance());  
 **if** (distance < origin.getMaxMoveDistance()) {  
 **gameObjectsMap**[i][j].setHilightedMoveTrue();  
 }  
 **if** (distance < origin.getMaxAttackDistance()){  
 **gameObjectsMap**[i][j].setHighlightedAttackTrue();  
 }  
 }  
 }  
 };  
  
 **public void** undoHighlightPattern(){  
 **for** (**int** i = 0; i < **gameObjectsMap**.**length**; i++){  
 **for** (**int** j = 0; j < **gameObjectsMap**[i].**length**; j++){  
 **gameObjectsMap**[i][j].setHilightedMoveFalse();  
 **gameObjectsMap**[i][j].setHighlightedAttackFalse();  
 }  
 }  
 }  
  
 **public float** getDistanceBetweenObjects(GameObject origin, GameObject dest){  
 Vector2 originPosition = getObjectPositionVector(origin);  
 Vector2 destPosition = getObjectPositionVector(dest);  
 **return** originPosition.dst(destPosition);  
 }  
  
 **public void** dispose(){  
 **for** (**int** i = 0; i < **gameObjectsMap**.**length**; i++){  
 **for** (**int** j = 0; j < **gameObjectsMap**[i].**length**; j++){  
 **gameObjectsMap**[i][j].dispose();  
 }  
 }  
 }  
  
 **public void** unselectObjects(){  
 **this**.**currentlySelectedObject**.setUnselected();  
 **this**.**currentlySelectedObject** = **null**;  
 }  
}

**package** io.gameclient.game;  
  
  
**import** com.badlogic.gdx.Gdx;  
**import** com.badlogic.gdx.audio.Music;  
**import** com.badlogic.gdx.audio.Sound;  
**import** com.badlogic.gdx.graphics.Texture;  
**import** com.badlogic.gdx.graphics.g2d.SpriteBatch;  
  
**import** java.io.File;  
**import** java.util.ArrayList;  
**import** java.util.List;  
  
**public class** Scout **extends** CharacterObject {  
 Texture **texture**;  
 **public int**[][] **movePattern** = {  
 {1, 1, 1, 1},  
 {1, 1, 1, 1},  
 {1, 1, 1, 1},  
 {1, 1, 1, 1}  
 };  
 List<Sound> **noSounds** = **new** ArrayList<Sound>();  
 List<Sound> **moveSounds** = **new** ArrayList<Sound>();  
 List<Sound> **killSounds** = **new** ArrayList<Sound>();  
  
 **public** Scout(Player player){  
 **super**(player);  
 **if** (player.**team** == Team.***BLUE***) {**this**.**texture** = **new** Texture(Gdx.*files*.internal(**"characters/blue\_scout\_alfa.png"**));}  
 **else** {**this**.**texture** = **new** Texture(Gdx.*files*.internal(**"characters/red\_scout\_alfa.png"**));}  
  
 **noSounds**.add(Gdx.*audio*.newSound(Gdx.*files*.internal(**"audio/scout/no/Scout\_jeers11.wav"**)));  
 **noSounds**.add(Gdx.*audio*.newSound(Gdx.*files*.internal(**"audio/scout/no/Scout\_jeers12.wav"**)));  
 **noSounds**.add(Gdx.*audio*.newSound(Gdx.*files*.internal(**"audio/scout/no/Scout\_no01.wav"**)));  
 **noSounds**.add(Gdx.*audio*.newSound(Gdx.*files*.internal(**"audio/scout/no/Scout\_no03.wav"**)));  
  
 **moveSounds**.add(Gdx.*audio*.newSound(Gdx.*files*.internal(**"audio/scout/move/Scout\_go01.wav"**)));  
 **moveSounds**.add(Gdx.*audio*.newSound(Gdx.*files*.internal(**"audio/scout/move/Scout\_battlecry03.wav"**)));  
 **moveSounds**.add(Gdx.*audio*.newSound(Gdx.*files*.internal(**"audio/scout/move/Scout\_battlecry04.wav"**)));  
 **moveSounds**.add(Gdx.*audio*.newSound(Gdx.*files*.internal(**"audio/scout/move/Scout\_battlecry05.wav"**)));  
  
 **killSounds**.add(Gdx.*audio*.newSound(Gdx.*files*.internal(**"audio/scout/kill/Scout\_domination19.wav"**)));  
 **killSounds**.add(Gdx.*audio*.newSound(Gdx.*files*.internal(**"audio/scout/kill/Scout\_dominationpyr04.wav"**)));  
 **killSounds**.add(Gdx.*audio*.newSound(Gdx.*files*.internal(**"audio/scout/kill/Scout\_dominationsct02.wav"**)));  
 }  
  
 **public float** getMaxAttackDistance(){**return** 1.6f;}  
 **public float** getMaxMoveDistance(){**return** 4.5f;}  
  
 **public void** render(SpriteBatch batch, **float** x, **float** y){  
 **super**.render(batch, x, y);  
 batch.draw(**this**.**texture**, x, y);  
 }  
  
 **public void** sayNo(){  
 **this**.getRandomSound(**this**.**noSounds**).play();  
 }  
  
 **public void** sayMove(){  
 **this**.getRandomSound(**this**.**moveSounds**).play();  
 }  
  
 **public void** sayKill(){  
 **this**.getRandomSound(**this**.killSounds).play();  
 }  
  
 **public void** dispose(){  
 **for** (Sound sound: **this**.noSounds){  
 sound.dispose();  
 }  
 **for** (Sound sound: **this**.moveSounds){  
 sound.dispose();  
 }  
 **for** (Sound sound: **this**.killSounds){  
 sound.dispose();  
 }  
 }  
}

**package** io.gameclient.game;  
  
**import** com.badlogic.gdx.Gdx;  
**import** com.badlogic.gdx.audio.Sound;  
**import** com.badlogic.gdx.graphics.Texture;  
**import** com.badlogic.gdx.graphics.g2d.SpriteBatch;  
  
**import** java.util.ArrayList;  
**import** java.util.List;  
  
**public class** Demoman **extends** CharacterObject{  
 Texture **texture**;  
 Sound **noSound**;  
  
 List<Sound> **noSounds** = **new** ArrayList<Sound>();  
 List<Sound> **moveSounds** = **new** ArrayList<Sound>();  
 List<Sound> **killSounds** = **new** ArrayList<Sound>();  
  
 **public** Demoman(Player player) {  
 **super**(player);  
 **if** (player.team == Team.BLUE) {  
 **this**.texture = **new** Texture(Gdx.files.internal(**"characters/blue\_demo\_alfa.png"**));  
 } **else** {  
 **this**.texture = **new** Texture(Gdx.files.internal(**"characters/red\_demo\_alfa.png"**));  
 }  
  
 noSounds.add(Gdx.audio.newSound(Gdx.files.internal(**"audio/demo/no/Demoman\_no01.wav"**)));  
 noSounds.add(Gdx.audio.newSound(Gdx.files.internal(**"audio/demo/no/Demoman\_no02.wav"**)));  
  
 moveSounds.add(Gdx.audio.newSound(Gdx.files.internal(**"audio/demo/move/en\_freedom.wav"**)));  
 moveSounds.add(Gdx.audio.newSound(Gdx.files.internal(**"audio/demo/move/en\_get\_em\_boyos.wav"**)));  
  
 killSounds.add(Gdx.audio.newSound(Gdx.files.internal(**"audio/demo/kill/en\_could\_be\_one.wav"**)));  
 killSounds.add(Gdx.audio.newSound(Gdx.files.internal(**"audio/demo/kill/en\_gonna\_kill\_you.wav"**)));  
 killSounds.add(Gdx.audio.newSound(Gdx.files.internal(**"audio/demo/kill/en\_reign\_of\_blood.wav"**)));  
 killSounds.add(Gdx.audio.newSound(Gdx.files.internal(**"audio/demo/kill/en\_wee\_lasses.wav"**)));  
  
 *// "NO" sound* **switch** (prefs.getInteger(**"lang"**)) {  
 **case** Langs.EN:  
 **this**.noSound = Gdx.audio.newSound(Gdx.files.internal(**"audio/demoman/Demoman\_no01.wav"**));  
 **break**;  
 **case** Langs.FR:  
 **this**.noSound = Gdx.audio.newSound(Gdx.files.internal(**"audio/demoman/Demoman\_no01\_fr.wav"**));  
 **break**;  
 **case** Langs.DE:  
 **this**.noSound = Gdx.audio.newSound(Gdx.files.internal(**"audio/demoman/Demoman\_no01\_de.wav"**));  
 **break**;  
 }  
 }  
  
 **public float** getMaxAttackDistance(){**return** 3.0f;}  
 **public float** getMaxMoveDistance(){**return** 3.0f;}  
  
  
 **public void** render(SpriteBatch batch, **float** x, **float** y){  
 **super**.render(batch, x, y);  
 batch.draw(**this**.texture, x, y);  
 }  
  
 **public void** sayNo(){  
 **this**.getRandomSound(**this**.noSounds).play();  
 }  
  
 **public void** sayMove(){  
 **this**.getRandomSound(**this**.moveSounds).play();  
 }  
  
 **public void** sayKill(){  
 **this**.getRandomSound(**this**.killSounds).play();  
 }  
  
 **public void** dispose(){  
 **this**.noSound.dispose();  
 }  
  
}

**package** io.gameclient.game;  
  
**import** com.badlogic.gdx.Gdx;  
**import** com.badlogic.gdx.Screen;  
**import** com.badlogic.gdx.audio.Music;  
**import** com.badlogic.gdx.graphics.OrthographicCamera;  
**import** com.badlogic.gdx.graphics.Texture;  
**import** com.badlogic.gdx.scenes.scene2d.Stage;  
**import** com.badlogic.gdx.utils.viewport.StretchViewport;  
  
*/\*\*  
 \* Created by tsh on 20-Jan-16.  
 \*/***public class** EndGameScreen **implements** Screen {  
 Texture **victoryLogo**;  
 **private** Stage **stage**;  
 Tf2Client **game**;  
 Integer **counter**;  
 Music backgroundMusic;  
  
 **public** EndGameScreen(Team victoriousTeam, Tf2Client gam){  
 **this**.**game** = gam;  
 **stage** = **new** Stage(**new** StretchViewport(854, 480));  
 **if** (victoriousTeam == Team.***RED***){  
 **victoryLogo** = **new** Texture(Gdx.*files*.internal(**"ui/red\_win.png"**));  
 } **else** {  
 **victoryLogo** = **new** Texture(Gdx.*files*.internal(**"ui/blue\_win.png"**));  
 }  
 **counter** = 0;  
  
 backgroundMusic = Gdx.*audio*.newMusic(Gdx.*files*.internal(**"audio/background/the-art-of-war.mp3"**));  
 backgroundMusic.setVolume(0.5f);  
 backgroundMusic.setLooping(**true**);  
 backgroundMusic.play();  
 }  
  
 @Override  
 **public void** show() {  
  
 }  
  
 @Override  
 **public void** render(**float** delta) {  
 **stage**.getBatch().begin();  
 **stage**.getBatch().draw(**this**.**victoryLogo**, 0, 0);  
 **stage**.getBatch().end();  
  
 **stage**.act(Math.*min*(Gdx.*graphics*.getDeltaTime(), 1 / 30f));  
 **stage**.draw();  
 **counter** += 1;  
 System.***out***.println(**counter**);  
 **if** (Gdx.*input*.isTouched() && **counter** > 60) {  
 **this**.**game**.setScreen(**new** MainMenuScreen(**this**.**game**));  
 dispose();  
 }  
  
 }  
  
 @Override  
 **public void** resize(**int** width, **int** height) {  
  
 }  
  
 @Override  
 **public void** pause() {  
  
 }  
  
 @Override  
 **public void** resume() {  
  
 }  
  
 @Override  
 **public void** hide() {  
  
 }  
  
 @Override  
 **public void** dispose() {  
 **this**.**stage**.dispose();  
 **this**.**victoryLogo**.dispose();  
 backgroundMusic.dispose();  
 }  
}

**package** io.gameclient.game;  
  
  
**import** com.badlogic.gdx.graphics.g2d.SpriteBatch;  
**import** com.badlogic.gdx.math.Vector2;  
  
**public class** GameObjectsManager {  
 **public final** Integer **TILE\_WIDTH** = 100;  
 **public final** Integer **TILE\_HEIGHT** = 100;  
  
 **private** GameObject **currentlySelectedObject** = **null**;  
 **private** MyGame **game**;  
  
 GameObject[][] **gameObjectsMap**;  
  
 **public** GameObjectsManager(MyGame game){  
 **this**.**game** = game;  
 **this**.**gameObjectsMap** = **new** GameObject[][]{  
 {**new** Demoman(game.**bluePlayer**), **new** EmptyGameObject(), **new** EmptyGameObject(), **new** EmptyGameObject(), **new** Scout(game.**redPlayer**)},  
 {**new** EmptyGameObject(), **new** EmptyGameObject(), **new** EmptyGameObject(), **new** EmptyGameObject(), **new** EmptyGameObject()},  
 {**new** EmptyGameObject(), **new** EmptyGameObject(), **new** EmptyGameObject(), **new** EmptyGameObject(), **new** EmptyGameObject()},  
 {**new** EmptyGameObject(), **new** EmptyGameObject(), **new** EmptyGameObject(), **new** EmptyGameObject(), **new** EmptyGameObject(),},  
 {**new** EmptyGameObject(), **new** EmptyGameObject(), **new** EmptyGameObject(), **new** EmptyGameObject(), **new** EmptyGameObject()},  
 {**new** Scout(game.**bluePlayer**), **new** EmptyGameObject(), **new** EmptyGameObject(), **new** EmptyGameObject(), **new** Demoman(game.**redPlayer**)}  
 };  
 }  
  
 **public void** render(SpriteBatch batch){  
 **for** (**int** i = 0; i < **gameObjectsMap**.**length**; i++){  
 **for** (**int** j = 0; j < **gameObjectsMap**[i].**length**; j++){  
 GameObject go = **gameObjectsMap**[i][j];  
 go.render(batch, i \* **TILE\_WIDTH**, j \* **TILE\_HEIGHT**);  
 }  
 }  
 }  
  
 **public void** inputReceived(**float** worldX, **float** worldY){  
 **int** row = (**int**)Math.*floor*(worldX / **TILE\_WIDTH**);  
 **int** column = (**int**)Math.*floor*(worldY / **TILE\_HEIGHT**);  
 GameObject clickedObject = **this**.**gameObjectsMap**[row][column];  
 *// set selection* **if** (**currentlySelectedObject** == **null** || **currentlySelectedObject instanceof** EmptyGameObject){  
 clickedObject.setSelected();  
 *// highlight action pattern* **if** (clickedObject **instanceof** CharacterObject){ highlightPattern((CharacterObject) clickedObject);}  
 **currentlySelectedObject** = clickedObject;  
 }  
 *// unset selection* **else if** (**currentlySelectedObject** == clickedObject){  
 clickedObject.toggleSelected();  
 **if** (clickedObject **instanceof** CharacterObject){ undoHighlightPattern();}  
 **currentlySelectedObject** = **null**;  
 }  
  
 **this**.**game**.performAction(**currentlySelectedObject**, clickedObject);  
 }  
  
 **public void** moveObject(GameObject movedObject, **int** destinationRow, **int** destinationColumn){  
 **int**[] movedObjectPosition = getObjectPosition(movedObject);  
 **gameObjectsMap**[destinationRow][destinationColumn] = movedObject;  
 **gameObjectsMap**[movedObjectPosition[0]][movedObjectPosition[1]] = **new** EmptyGameObject();  
 }  
  
 **public void** moveObject(GameObject movedObject, GameObject destinationObject){  
 **int**[] movedObjectPosition = getObjectPosition(movedObject);  
 **int**[] destinationPosition = getObjectPosition(destinationObject);  
 **gameObjectsMap**[destinationPosition[0]][destinationPosition[1]] = movedObject;  
 **gameObjectsMap**[movedObjectPosition[0]][movedObjectPosition[1]] = **new** EmptyGameObject();  
 }  
  
 **public void** attackTarget(GameObject attacker, CharacterObject target){  
 removeCharacter(target);  
 }  
  
 **private void** removeCharacter(CharacterObject characterObject){  
 **int**[] position = getObjectPosition(characterObject);  
 **gameObjectsMap**[position[0]][position[1]] = **new** EmptyGameObject();  
  
 }  
  
 **public int**[] getObjectPosition(GameObject object){  
 **int**[] position = **new int**[2];  
 **for** (**int** i = 0; i < **gameObjectsMap**.**length**; i++){  
 **for** (**int** j = 0; j < **gameObjectsMap**[i].**length**; j++){  
 **if** (**gameObjectsMap**[i][j] == object){  
 position[0] = i;  
 position[1] = j;  
 **return** position;  
 }  
 }  
 }  
 **throw new** IllegalArgumentException(**"Object not found in GameObjectArray"**);  
 }  
  
 **public** Vector2 getObjectPositionVector(GameObject object){  
 Vector2 position = **new** Vector2(0.0f, 0.0f);  
 **for** (**int** i = 0; i < **gameObjectsMap**.**length**; i++){  
 **for** (**int** j = 0; j < **gameObjectsMap**[i].**length**; j++){  
 **if** (**gameObjectsMap**[i][j] == object){  
 position.**x** = i;  
 position.**y** = j;  
 **return** position;  
 }  
 }  
 }  
 **throw new** IllegalArgumentException(**"Object not found in GameObjectArray"**);  
 }  
  
 **public int** getLiveTeamCharacterCount(Team team){  
 **int** count = 0;  
 **for** (**int** i = 0; i < **gameObjectsMap**.**length**; i++){  
 **for** (**int** j = 0; j < **gameObjectsMap**[i].**length**; j++){  
 **if** (**gameObjectsMap**[i][j] **instanceof** CharacterObject){  
 CharacterObject chr = (CharacterObject)**gameObjectsMap**[i][j];  
 **if** (chr.**player**.**team** == team) {count += 1;}  
 }  
 }  
 }  
 **return** count;  
 }  
  
 **public void** highlightPattern(CharacterObject origin) {  
 **for** (**int** i = 0; i < **gameObjectsMap**.**length**; i++) {  
 **for** (**int** j = 0; j < **gameObjectsMap**[i].**length**; j++) {  
 **float** distance = getDistanceBetweenObjects(origin, **gameObjectsMap**[i][j]);  
 System.***out***.println(distance + **" "** + origin.getMaxMoveDistance() + **" "** + origin.getMaxAttackDistance());  
 **if** (distance < origin.getMaxMoveDistance()) {  
 **gameObjectsMap**[i][j].setHilightedMoveTrue();  
 }  
 **if** (distance < origin.getMaxAttackDistance()){  
 **gameObjectsMap**[i][j].setHighlightedAttackTrue();  
 }  
 }  
 }  
 };  
  
 **public void** undoHighlightPattern(){  
 **for** (**int** i = 0; i < **gameObjectsMap**.**length**; i++){  
 **for** (**int** j = 0; j < **gameObjectsMap**[i].**length**; j++){  
 **gameObjectsMap**[i][j].setHilightedMoveFalse();  
 **gameObjectsMap**[i][j].setHighlightedAttackFalse();  
 }  
 }  
 }  
  
 **public float** getDistanceBetweenObjects(GameObject origin, GameObject dest){  
 Vector2 originPosition = getObjectPositionVector(origin);  
 Vector2 destPosition = getObjectPositionVector(dest);  
 **return** originPosition.dst(destPosition);  
 }  
  
 **public void** dispose(){  
 **for** (**int** i = 0; i < **gameObjectsMap**.**length**; i++){  
 **for** (**int** j = 0; j < **gameObjectsMap**[i].**length**; j++){  
 **gameObjectsMap**[i][j].dispose();  
 }  
 }  
 }  
  
 **public void** unselectObjects(){  
 **this**.**currentlySelectedObject**.setUnselected();  
 **this**.**currentlySelectedObject** = **null**;  
 }  
}

# Приложение В DVD с электронными материалами к проекту