**МИНИСТЕРСТВО ЦИФРОВОГО РАЗВИТИЯ, СВЯЗИ И МАССОВЫХ КОММУНИКАЦИЙ РОССИЙСКОЙ ФЕДЕРАЦИИ**

**ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ**

**УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ**

**«САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ТЕЛЕКОММУНИКАЦИЙ ИМ. ПРОФ. М.А. БОНЧ-БРУЕВИЧА»**

(СПбГУТ)

Факультет Инфокоммуникационных сетей и систем

Кафедра Защищенных систем связи

Дисциплина Разработка защищенных сетевых приложений

Отчет по курсовой работе

Разработка простой пошаговой игры

Студент:

Мелешкин О.С.

Преподаватель:

Шариков П.И.

Санкт-Петербург

2023

# Оглавление

[Оглавление 2](#_Toc153300426)

[Введение 3](#_Toc153300427)

[Основная часть 4](#_Toc153300428)

[1. Используемые библиотеки 4](#_Toc153300429)

[1.1. JavaFX 4](#_Toc153300430)

[1.1.1. Кнопки 5](#_Toc153300431)

[1.1.2. Текстовое поле 5](#_Toc153300432)

[1.1.3. Окна 6](#_Toc153300433)

[1.1.4. Фоновая музыка 7](#_Toc153300434)

[1.2. Random 8](#_Toc153300435)

[1.2.1. Random для генерации стоимости клеток 8](#_Toc153300436)

[1.2.2. Random для примитивного искусственного интеллекта 9](#_Toc153300437)

[1.3. Base64 11](#_Toc153300438)

[1.3.1. Принцип Base64 11](#_Toc153300439)

[1.3.2. Реализация Base64 в курсовой работе 12](#_Toc153300440)

[2. Реализация курсовой работы 13](#_Toc153300441)

[2.1. Создание основной механики. UML диаграмма прецедентов. 13](#_Toc153300442)

[2.2. Реализация программного кода 15](#_Toc153300443)

[2.2.1. Создание окон и сцен 15](#_Toc153300444)

[2.2.2. Создание элементов интерфейса 16](#_Toc153300445)

[2.2.3. Реализация сохранений 19](#_Toc153300446)

[2.2.4. UML диаграмма классов 21](#_Toc153300447)

[2.3. Демонстрация готового продукта 22](#_Toc153300448)

[Заключение 30](#_Toc153300449)

# Введение

Язык программирования Java является одним из самых популярных языков в мире и широко используется для разработки различных программных приложений, включая игры и стратегии. В данной курсовой работе мы будем рассматривать создание простой пошаговой стратегии на языке программирования Java.

Целью данной работы является ознакомление с основами разработки игр на языке Java и создание простой стратегии с использованием объектно-ориентированного подхода. Для достижения этой цели мы рассмотрим несколько ключевых аспектов разработки игр, таких как моделирование игрового мира, управление игровыми объектами, взаимодействие с игроком и логика игрового процесса.

В ходе работы мы будем использовать интегрированную среду разработки (IDE) для Java, которая предоставляет удобный набор инструментов для разработки и отладки программного обеспечения. Также мы будем использовать различные библиотек, которые позволят нам упростить разработку игры.

В первой части работы мы рассмотрим библиотеки, использованные при разработке простой пошаговой стратегии, что они делают, какие основные функции в них присутствуют и зачем они понадобились в моей курсовой работе.

Во второй части работы мы уже подробно рассмотрим сам процесс создания игры, а именно разработку идеи и основных механик, процесс кодирования и составления всего в единый готовый продукт.

В третьей части мы посмотрим на итоговый результат, как выглядит интерфейс программы, как в ней взаимодействуют различные аспекты и в целом как играть в эту пошаговую стратегию.

Таким образом, данная курсовая работа позволит ознакомиться с разработкой простой пошаговой стратегии на языке программирования Java и получить практические навыки в разработке игр.

# Основная часть

## Используемые библиотеки

В данной работе были использованы такие библиотеки как JavaFX, Base64, Random и log4j. JavaFX и log4j являются сторонними библиотеками для Java, которые я скачал с их официального сайта. Base64 и Random, в свою очередь, являются встроенными библиотеками языка Java. Далее я подробно расскажу о функционале каждой библиотеки, и, непосредственно, применении в курсовой работе.

### JavaFX

JavaFX является основой моей курсовой работы. Вокруг этой библиотеки завязана вся моя программа и весь мой код.

JavaFX – это библиотека позволяющая создать графический интерфейс для приложения, разработанного на Java. Она имеет огромный функционал, позволяющий реализовать невероятное множество идей. К таковому можно отнести следующие возможности:

#### Кнопки

Одна из главных вещей в графическом интерфейсе это кнопки. С помощью них в программе происходит абсолютное большинство действий. Кнопки есть в любой программе, начиная с простейшего текстового редактора, заканчивая ААА игрой современности. Всё что их объединяет это наличие в них кнопок. Как же реализовать кнопки на JavaFX?

Для начала необходимо создать объект типа «Button». Создаётся объект также, как и все остальные:  
private static Button next, load, newgame, change;  
Таким образом у нас создадутся объекты с типом кнопки под названием «next», «load», «newgame» и «change».

Далее необходимо, чтобы кнопка при нажатии что-либо делала. Для этого в библиотеке есть функция «setOnAction». С помощью неё можно назначить действие при нажатии определенной кнопки. Полная функция для назначения действия при нажатии кнопки «next» будет выглядеть так:

*next*.setOnAction(new EventHandler<ActionEvent>() {  
 @Override  
 public void handle(ActionEvent event) {  
   
 }  
});

Внутри функции «handle» непосредственно пишется, что должно произойти при нажатии кнопки.

Также возможны косметические изменения кнопок в программе. Можно изменить размер кнопки с помощью функции «setPrefSize» или изменить текст на кнопке с помощью «setText».

#### Текстовое поле

Текстовое поле также важно как и кнопки, и их также можно реализовать в JavaFX. По своей сути текстовое поле является кнопкой без функционала. Единственная цель такого поля – передать какой либо заданный текст или вывести новый. Создается и редактируется оно точно также как и кнопка.

#### Окна

Ещё одна незаменимая вещь при работе с графическим интерфейсом это окна. На них и завязана вся графика программы. В них находятся все элементы начиная от кнопок и заканчивая полем ввода данных. Именно поэтому остановимся на окнах поподробнее.

Для создания окна в первую очередь надо создать контейнер, класс Stage который запускает непосредственно окно на компьютере. Однако одного этого контейнера недостаточно. Нам также нужна сцена, которая является полем размещения всех элементов интерфейса. Есть множество различных типов сцен, но я расскажу о трёх, а именно о тех, которые я использовал в своей курсовой работе.

Первым рассмотрю VBox. Данный тип сцен позволяет расположить элементы интерфейса друг над другом. То есть будет вертикальный столбец на котором располагаются все кнопки\строки.

Второй тип сцен – это TitlePane. Его особенность заключается в том, что элементы располагаются как при письме: слева направо, сверху вниз. Таким образом кнопки\строки будут идти от левого края окна к правому, а когда окно кончится, то перейдут на следующую строку, и так далее, пока окно или элементы интерфейса не кончатся.

Стоит отметить, что к перечисленным выше сценам можно применить функции, которые регулируют размер пробелов между элементами интерфейса:  
*main\_menu*.setPadding(new Insets(10, 10, 10, 10));  
Данная функция сделает пробелы между элементами интерфейса во всех плоскостях размером 10 пикселей. Также можно применить функцию, ставящую ось центра интерфейса, то есть где будут располагаться кнопки и т.д.:  
*main\_menu*.setAlignment(Pos.*CENTER*);  
Данная функция ставит интерфейс в сцене «main\_menu» по центру. Для добавления элементов к сцене необходимо воспользоваться следующей функцией:  
*main\_menu*.getChildren().addAll(*newgame*, *load*, *musicButton*);  
Она добавляет в сцену «main\_menu» кнопки «newgame», «load», «musicButton». Добавить можно любое количество элементов, главное перечислить все через запятую в порядке, необходимом нам.

Третий тип – это GridPane. Эта сцена, как следует из названия, представляет собой сетку, на которой размещаются элементы интерфейса. Для размещения на определенной позиции чего-либо необходимо воспользоваться функцией:  
*game\_menu*.add(*next*, 1, 0);  
Она выставляет кнопку «next» в сцене «game\_menu» на 1-ой строке в 0-ом столбце.

#### Фоновая музыка

Неплохим дополнением к программе может быть фоновая музыка при запуске приложения. Далее мы поговори о том, как её добавить и как её зациклить, чтобы при окончании она начиналась заново.

Для запуска музыки, для начала, надо создать переменную типа «File» с указанием пути до необходимого файла, у которого будет соответствующий тип. После этого мы создаём переменную типа «Media» со следующими параметрами:  
*sound* = new Media(*musicFile*.toURI().toString());  
Таким образом мы создадим переменную «sound», в которой записан путь до файла с музыкой. Далее дело за малым, создаем объект типа «MediaPlayer», в параметрах которого мы указываем «sound». Когда все действия сделаны запускаем музыку с помощью команды:  
*mediaPlayer*.play();

После того, как мы запустили музыку, нам надо сделать так, чтобы она зациклилась, и не прекращалась в конце. Для этого в библиотеке JavaFX есть особый триггер под названием «setOnEndOfMedia». Как следует из названия, он активируется, когда проигрываемая музыка оканчивается. Далее, для повтора, надо запустить функцию на повторный проигрыш мелодии. Полностью функция с триггером выглядят так:  
*mediaPlayer*.setOnEndOfMedia(new Runnable() {  
 public void run() {  
 *mediaPlayer*.seek(Duration.*ZERO*);  
 }  
});  
С помощью данной команды как раз и получится повторять музыку раз за разом.

Подводя итоги по библиотеке JavaFX, можно сказать, что с её помощью можно сделать качественный интерфейс под практически любые нужды. В ней присутствуют множество функций под абсолютно разные задачи, в том числе и под создание простейших игр.

### Random

Библиотека Random позволяет генерировать случайные числа в заданном диапазоне. В моей курсовой работе данная библиотека выполняет две функции: случайная генерация стоимости полей в игре и случайные процессы в примитивном искусственном интеллекте. Далее я расскажу о функциях этой библиотеки и что я реализовывал в курсовой с помощью неё.

#### Random для генерации стоимости клеток

Первая функция данной библиотеки в моей работе это случайная генерация стоимости клеток для покупки. Есть три величины, которые случайным образом принимают целые значения от одного до девяти. Каждый раз при запуске новой игры каждой из 100 клеток присваивается три таких случайных значения. Ниже я покажу строчки кода, в которых это происходит:  
Random random = new Random();  
cost[0] = random.nextInt(9)+1;  
cost[1] = random.nextInt(9)+1;  
cost [2] = random.nextInt(9)+1;  
В данном случае мы выбираем случайное число из первых девяти позиций (от нуля до восьми) и прибавляем единицу, чтобы стоимость не была равна нулю. Стоит обратить внимание, что генерируются именно целые числа типа «Int».

#### Random для примитивного искусственного интеллекта

Вторая функция, реализованная в моём курсовом проекте это функция искусственного интеллекта. Далее я подробно опишу, как он работает в моей пошаговой стратегии.

У противника есть базовая вероятность по окончанию хода захватить соседнюю провинцию (эта вероятность равна 0,14). Каждый раз при нажатии кнопки «следующий ход» запускается генератор случайных чисел, который создаёт число в диапазоне от 0 до 100, причём число типа double. В случае, если вероятность больше, чем сгенерированное число, то противник захватывает регион в этом ходу.

Выбор региона происходит также случайным образом. Сначала проверяется, находится ли регион по соседству с уже имеющимся регионом противника. Если это условие выполняется, то регион записывается в специальный массив. После этого случайным образом выбирается целое число типа «int» не превышающее количество элементов этого специального массива. Выбранный таким образом регион меняет значение «fraction» на 2, тем самым становясь вражеским. При этом базовая вероятность на захват региона сбрасывается до 1.

В случае, если вероятность на захват меньше, чем случайное число типа «double», противник не захватывает ничего за ход, а базовая вероятность умножается на 1,4. Таким образом вероятность захватить регион будет расти с каждым ходом, пока поле не будет взято.

Ниже будет приведет листинг код, отвечающий за сказанное мною выше:

if (chance > random.nextDouble(100)){  
 int k=0;  
 for (int i = 0; i<10; i++)  
 for (int j = 0; j<10; j++){  
 if (i!=9 && *field\_cost*[i][j].fraction==0 && *field\_cost*[1+i][j].fraction==2) {  
 boti[k]=i;  
 botj[k]=j;  
 k++;  
 }  
 else if (j!=9 && *field\_cost*[i][j].fraction==0 && *field\_cost*[i][1+j].fraction==2) {  
 boti[k]=i;  
 botj[k]=j;  
 k++;  
 }  
 else if (i!=0 && *field\_cost*[i][j].fraction==0 && *field\_cost*[i-1][j].fraction==2) {  
 boti[k]=i;  
 botj[k]=j;  
 k++;  
 }  
 else if (j!=0 && *field\_cost*[i][j].fraction==0 && *field\_cost*[i][j-1].fraction==2) {  
 boti[k]=i;  
 botj[k]=j;  
 k++;  
 }  
 }  
 int choose = new Random().nextInt(k);  
 *field\_cost*[boti[choose]][botj[choose]].fraction=2;  
 *field\_point*[boti[choose]][botj[choose]].setStyle("-fx-background-color: red;");  
 chance=1;  
}  
else  
 chance\*=1.4;

Именно таким образом библиотека Random реализовывается в моей курсовой работе и в моей пошаговой стратегии.

### Base64

Base64 – это библиотека, позволяющая шифровать текст на основе одноимённого кодирования. В моей работе данный «шифр» выполняет роль шифратора файлов сохранения и, соответственно, роль дешифратора этих же файлов. Ниже я опишу принцип работы системы Base64 и её реализацию в моей курсовой работе.

#### Принцип Base64

Base64 - это часто используемый и надежный метод представления произвольных двоичных данных в виде текста. Название происходит от того факта, что Base64 использует 64 возможных значения для представления двоичных данных. Это означает, что существует шесть битов для представления одного символа Base64.

Для примера зашифруем слово «one» этим типом кодировки. В двоичном представлении слово «one» представляет из себя следующий набор нулей и единиц. «о» - 01101111, «n» - 01101110, «е» - 01100101. Для кодировки с помощью Base64 мы сквозным потоком идём от первого бита, до последнего, при этом делая «паузу» после каждого шестого бита. Получаем четыре набора нулей и единиц: 011011, 110110, 111001, 100101. Получив их мы сравниваем с таблицей Base64 (в свободном доступе можно найти в сети Internet), где первому набору соответствует символ «b», второму «2», третьему «5», и четвёртому «l». Таким образом слово «one» преобразовалось в «b25l». Декодирование происходит в обратном порядке.

#### Реализация Base64 в курсовой работе

Как было сказано выше, в игре Base64 выполняет роль шифратора и дешифратора файлов сохранения. Необходимо это для того, чтобы игрок не получал преимущество путём редактирования файла сохранения, или, по крайней мере, делал это не самым простым методом.

В курсовой работе реализованы две функции из библиотеки Base64. Одна функция кодирует данные, а вторая декодирует. Первым делом рассмотрим метод для кодировки исходных данных:

Base64.*getEncoder*().encodeToString(Integer.*toString*(game.move).getBytes())

В данной реализации мы видим вызов библиотеки с помощью «Base64». Далее происходит вызов функции по кодированию «getEncoder». Далее происходит перевод из шестибитового представления символов в привычный нам строковый того, что мы закодируем. Кодируем мы число типа «int» переменной «game.move». Поскольку кодирование происходит только из типа данных «String», мы изначально переводим в этот тип наше число с помощью «Integer.toString». Дальше получаем байтовое представление с помощью «getBytes()». В конце концов полученные байты кодируются по описанному в предыдущем пункте методу.

Теперь поговорим о процессе декодирования. Он проходит немного проще, чем процесс кодирования. Участок кода из моей курсовой, в которой это реализовано выглядит следующим образом:  
String line = reader.readLine();  
line = new String(Base64.*getDecoder*().decode(line));  
game.move=Integer.*parseInt*(line);  
Сначала мы читаем из файла строку и записываем в переменную «line» типа «String». Дальше мы перезаписываем в эту переменную декодированный текст. Поскольку полученный в процессе декодирования текст является набором нулей и единиц, нам необходимо его также перевести в тип «String». Дальше, при необходимости, мы переводим строку в число «int» и присваиваем его соответствующей переменной, в моём случае «game.move». Таким образом повторяем со всеми строками в файле сохранения.

### Логгирование с помощью библиотеки log4j

Библиотека log4j предназначена для логгирования действий и прочих активностей во время работы программы. С её помощью можно легко отслеживать, на каком этапе произошли те или иные события.

В моей работе log4j используется для записи всех действий игрока, а именно нажатий кнопок. В случае каких либо ошибок, также происходит запись. Также, помимо этого, логгируется запуск и закрытие программы. Далее я собираюсь рассказать о том, каким образом происходит логгирование.

Первым делом надо создать объект с классом «Logger»:

private static final Logger *logger* = LogManager.*getLogger*("Strategy Game");

В данном случае он называется «Strategy Game». Далее остаётся только расставить логгер в соответствующих местах с соответствующим типом логгирования (ошибка, фатальная ошибка или просто информация). Делается это так:

*logger*.info("Program has been successfully launched");

*logger*.fatal("Program was closed\n");

*logger*.error("File has not been choosen");

Таким образом, подводя итоги по библиотеке log4j, можно сказать, что она станет хорошим дополнением к программе, с помощью которого разработчику будет проще находить ошибки, а пользователю понимать, что привело к ошибке.

## Реализация курсовой работы

Реализовалась данная курсовая работа примерно двадцать четыре часа, с учётом подготовки перед началом. Ниже я хочу показать весь процесс реализации курсовой работы, от обдумывания идеи и общей структуры проекта, до финального результата и демонстрации продукта. Поэтому данный пункт будет разделен на три раздела: создание основных механик пошаговой стратегии, реализация проекта и готовый продукт в работе.

### Создание основной механики. UML диаграмма прецедентов.

Прежде чем начать делать курсовой проект, а тем более игру в виде пошаговой стратегии, надо обдумать основные механики и в целом придумать цель игры.

Первым делом надо определить тип игры. Как следует из задания к курсовой работы – тип игры пошаговая стратегия. Однако у них есть множество подвидов. Лично я решил сделать экономическую пошаговую стратегию, в которой будет игрок и один противник. При всём этом, действие будет происходить на одной карте, а вражды с противником так таковой не будет.

Следующее, что я обдумал – это размер игрового поля. Очевидно, карта не может быть бесконечной, поэтому её размер также должен быть заранее определен. В моём случае размер я сделал десять на десять клеток, что в сумме даёт ровно 100 игровых клеток.

У нас есть общая концепция игры и игровое поле. Однако ни одна пошаговая стратегия не обходится без ресурсов. В случае моей игры я решил добавить следующие четыре ресурса: крестьяне, дома, рис, вода.

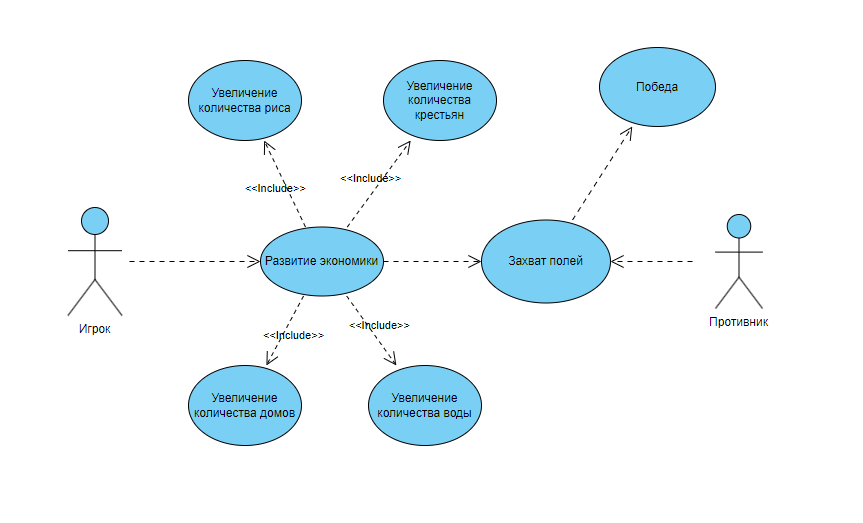
Помимо придумывания ресурсов, им надо придумать предназначение. Крестьяне – основной расходный материал, необходимый для любого действия. Дом – постройка которая автоматически воссоздаёт крестьян с каждым ходом. Для его постройки потребуется по единице каждого ресурса. Вода – второстепенный ресурс, который может быть использован для покупки поля на карте, либо для полива риса, от чего в следующем ходу рис можно будет собрать в трёхкратном размере. И последнее, рис – ресурс необходимый для покупки полей.

Для успешного начала игры необходимо придумать начальные условия, с которых игрок будет начинать. Для баланса я решил, что в начале будет по единице каждого ресурса, кроме крестьян. Их будет двое. К тому же начальными позициями на карте для нас и противника будут два противоположных угла. Таким образом шансы на победу будут одинаковыми.

Раз речь зашла о победе, надо также придумать условия выигрыша и проигрыша игрока. Я решил, что для победы игрок должен захватить более половины всех полей, а именно 51 штуку. Для поражения 51 поле должен захватить противник.

И последнее, что осталось решить, это как будет происходить покупка полей. Мной было принято решение, что их стоимость будет являться тремя случайными целыми числами от 1 до 9. Каждое из них это стоимость в крестьянах, воде и рисе соответственно.

В итоге мы составили общее представление о том, что будет из себя представлять будущий продукт и каким он примерно будет на выходе. Для большей наглядности нарисую UML диаграмму прецедентов:

Она показывает общую концепцию игры, к чему надо стремиться и что надо делать.

### Реализация программного кода

Когда мы придумали общую концепцию игры, пришло время её реализовывать. В этом разделе я расскажу о процессе разработки и подробно опишу некоторые реализации в проекте.

#### Создание окон и сцен

Первым делом необходимо было создать окна и сцены для них. По моей задумке, окон должно было быть четыре, а сцен для них шесть: окно главного меню, окно самой игры, окно покупки полей, окно карты и окно сохранения. Для каждого окна соответствующая сцена, однако для карты будет две сцены, а именно карта стоимости полей и карта контролируемых территорий. Ниже будет показан пример создания окна со сценой:  
*stagee* = new Stage();

*stagee*.setResizable(false);

*main\_menu* = new VBox();  
*main\_menu*.setPadding(new Insets(10, 10, 10, 10));  
*main\_menu*.setSpacing(10);  
*main\_menu*.setAlignment(Pos.*CENTER*);  
*main\_menu*.getChildren().addAll(*newgame*, *load*, *musicButton*);  
*scene* = new Scene(*main\_menu*, 800, 600);  
*stagee*.setTitle("Strategy Game");  
*stagee*.setScene(*scene*);  
*stagee*.show();  
В данном участке кода мы создаем окно «stageе» и запрещаем изменение его размера (для удобства работы над интерфейсом). Создаем «шаблон» «main\_menu» и настраиваем его (подробнее о функциях в разделе 1.1.3.). После чего создаём сцену «scene» в которую добавляем «main\_menu» и указываем размер окна. Присваиваем окну имя и сцену, после чего запускаем. Тоже самое делаем со всеми остальными окнами.

Таким образом у нас получается шесть окон с семью сценами, которые имеют свой набор кнопок табличек и др. элементов интерфейса.

#### Создание элементов интерфейса

Когда у нас есть окна, необходимо добавить к ним интерфейс в лице кнопок и текстовых полей. В моём случае есть 2 типа кнопок/полей. Одни предназначены для основных окон, таких как окно главного меню или игровое окно, а вторые предназначены для отображения игрового поля или меню покупок этих самых полей. Далее я расскажу о процессе создания обоих.

Я выделил для себя два типа кнопок интерфейса, а именно кнопки служебные и простые кнопки. Служебные кнопки имеют своё название. Среди них такие кнопки, как «next», «load», «newgame», «change». Остальные же кнопки находятся в массиве кнопок «buttons» и вызываются соответствующим порядковым номером. Пример объявления кнопок:  
private static Button *next*, *load*, *newgame*, *change*;  
private static final Button [] *buttons* = new Button[10];

В данном участке кода мы создали четыре указанных мною кнопки, а также массив из десяти кнопок.

Аналогичным образом я создал текстовые поля. Есть служебные, в моём случае одно, под названием «move» и такой же массив этих полей:  
private static Label *move*;  
private static final Label[] *lables* = new Label[10];

Далее я расскажу о том, как я присваивал этим элементам свой текст и другие различные свойства.

Чтобы назначить кнопке или полю некий текст надо прописать следующую «команду»:

*buttons*[3].setText("Купить поле");

*lables*[0].setText("Количество крестьян");

В данном случае мы присвоили кнопке «buttons[3]» и текстовому полю «lables[0]» текста в скобках. Помимо команды «set.Text» можно задавать текст сразу при объявлении, указав в параметрах соответствующие строки.

Помимо присваивания текста, можно активировать и деактивировать кнопки. Деактивированная кнопка становится недоступна для прожатия. Это используется чтобы игрок не делал множество действий за один ход.

*buttons*[i].setDisable(true);

В указанном примере мы отключаем возможность нажатия на кнопку «buttons[i]»

Теперь поговорим о реализации кнопок и текстовых полей для окна карты и окна покупки зон. Кардинально ничего не изменилось, по сравнению с предыдущими абзацами, кроме того, что мы делаем двойной массив кнопок/полей. Поскольку наша карта представляет из себя матрицу 10х10 клеток, то и двойной массив будет иметь размерность 10х10. Генерация кнопок покупки полей будет выглядеть таким образом:

for (int i = 0; i<10; i++) {  
 for (int j = 0; j < 10; j++) {  
 field\_button[i][j] = new Button();  
 field\_button[i][j].setPrefSize(55, 55);  
 field\_button[i][j].setDisable(true);  
 *field\_2*.getChildren().add(field\_button[i][j]);  
 }  
}

В приведенном участке кода мы сгенерировали массив кнопок от «field\_button[0][0]» до «field\_button[9][9]». При этом каждой кнопке по умолчанию установили размер 55х55 пикселей, сделали неактивной и присвоили сцене «field\_2». Подобным образом мы генерируем и текстовые поля.

Когда мы создали объекты кнопок, необходимо задать им какое-то действие при нажатии. Сейчас я расскажу о некоторых кнопках. А именно о кнопке полива риса.

Листинг кода, отвечающего за нажатие кнопки «Поливка риса» выглядит следующим образом:

*buttons*[6].setOnAction(new EventHandler<ActionEvent>() {  
 @Override  
 public void handle(ActionEvent event) {  
 if (game.water.count > 0){  
 game.water.count-=1;  
 game.rise.ChangeWatered();  
 *buttons*[6].setText("Рис полит");  
 for (int i = 4; i<8; i++)  
 *buttons*[i].setDisable(true);  
 }  
 else{  
 *buttons*[6].setText("Недостаточно воды для полива");  
 }  
 }  
});

В случае нажатия на кнопку «buttons[6]» возникает событие, указанное в функции «handle». Происходит проверка количества воды у игрока. Если воды больше нуля, то списывается единичка воды, меняется статус риса и кнопка на «полит», а все кнопки, отвечающие за добычу ресурсов становятся неактивными. В случае если воды недостаточно, кнопка меняет текст на указанный в коде. Остальные кнопки имеют подобную структуру, но с другими методами и реализациями.

#### Реализация сохранений

Не менее важной вещью в создании игры является реализовывание сохранений. С помощью них можно сохранять и загружать уже начатую игру. Общая концепция сохранений в моём проекте достаточно простая: есть папка «Saves», в которой создаются файлы с расширением «.save». В них записан ход, количество ресурсов, стоимость каждой клетки поля, их принадлежность и возможность использовать кнопки развития. Далее я расскажу о конкретной реализации сохранения и загрузки игры.

Реализация сохранения можно разделить на две условные части: первая – это создание файла сохранения, а вторая – это запись в файл всех необходимых данных. Начнём с создания файла.

В окне, которое появляется после нажатия «сохранить игру» есть поле для ввода названия файла. Какое название мы введем – такой название и будет у файла сохранения. Реализовано это следующим образом:

try(FileWriter writer = new FileWriter("Saves\\"+ *save\_field*.getText() +".save"))

Мы открываем поток на запись с параметром указанным выше, где «Saves» - это папка в которой будет храниться файл, «save\_field.getText()» - команда, которая считывает текст, записанный игроком, а «.save» - это расширение файла. Таким образом у нас создастся файл с названием, которое придумал игрок.

Следующим этапом будет запись имеющихся данных, причём в кодировке Base64. Для этого мы создаём переменную типа «String» под названием «text». Далее прибавляем к этой переменной по очереди строку с определенными данными, заранее шифруя их по Base64 и не забывая после каждой строки переходить на следующую:

text=text+Base64.*getEncoder*().encodeToString(Integer.*toString*(game.move).getBytes())+"\n"

В данном участке мы к строке «text» добавили зашифрованное значение хода в игре и перенеслись на следующую строку. Все последующие данные записываются таким же образом.

Следующим этапом разработки станет реализация загрузки игры. По своей сути, мы открываем файл сохранения на чтение и в обратном порядке достаём оттуда все записанные данные, попутно декодируя их по Base64.

При загрузке сохранений реализовывается такой класс как «FileChooser». С его помощью можно удобно выбирать файл на своём компьютере.

FileChooser fileChooser = new FileChooser();  
fileChooser.getExtensionFilters().addAll(  
 new FileChooser.ExtensionFilter("SAVE", "\*.save"),  
 new FileChooser.ExtensionFilter("ALL FILES", "\*.\*"));  
fileChooser.setTitle("Open Resource File");  
File file = fileChooser.showOpenDialog(*stagee*);

В приведенном выше участке кода реализован метод, позволяющий выбрать файл. По умолчанию выводятся только файлы типа «.save». Выбрав соответствующий файл, его путь переносится в переменную file типа «File».

if (file == null) {  
 *alert\_no\_file*.showAndWait();

В случае если файл не выбран, у нас выходит диалоговое окно с ошибкой, где говорится о том, что файл не выбран.

Если же файл выбран, мы открываем его на чтение и присваиваем первую строку переменной «line» типа «String»:

BufferedReader reader = new BufferedReader(new FileReader(file.getAbsolutePath()));  
String line = reader.readLine();

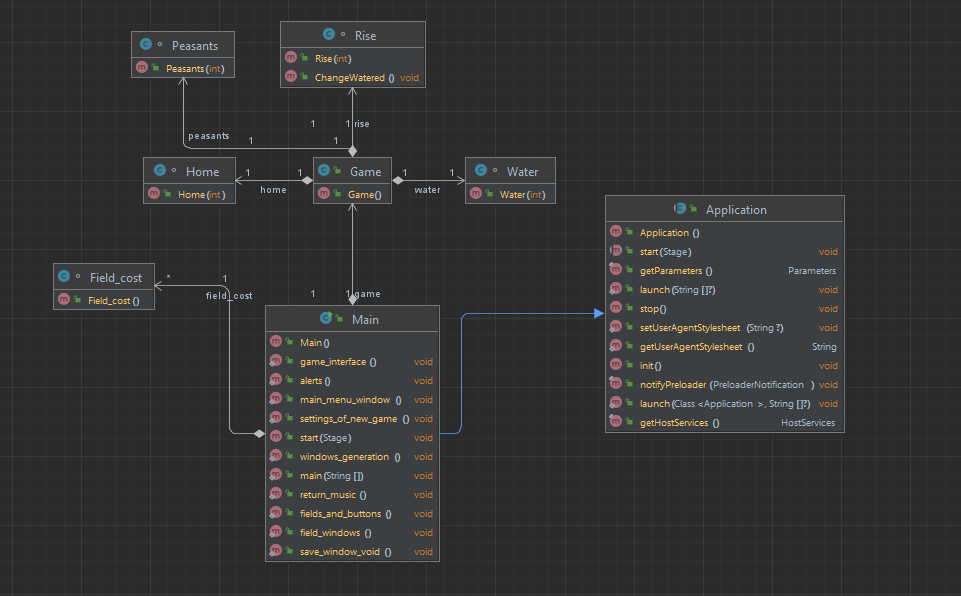
После этого мы декодируем строку с помощью Base64 и полученное значение присваиваем соответствующей переменной, после чего идём дальше по строкам:

line = new String(Base64.*getDecoder*().decode(line));  
game.move=Integer.*parseInt*(line);  
line = reader.readLine();

Загрузив всё с файла сохранения, программа запускает окно игрового меню со всеми соответствующими параметрами. Также стоит отметить, что если в файле будут символы, не подходящие под необходимые нам переменные, откроется диалоговое окно, сообщающее о том, что в файле содержатся запрещенные символы.

Таким образом и реализованы сохранения в моей игре. С помощью них можно в любой момент сохранить весь полученный прогресс и продолжить на том же моменте без всяких проблем.

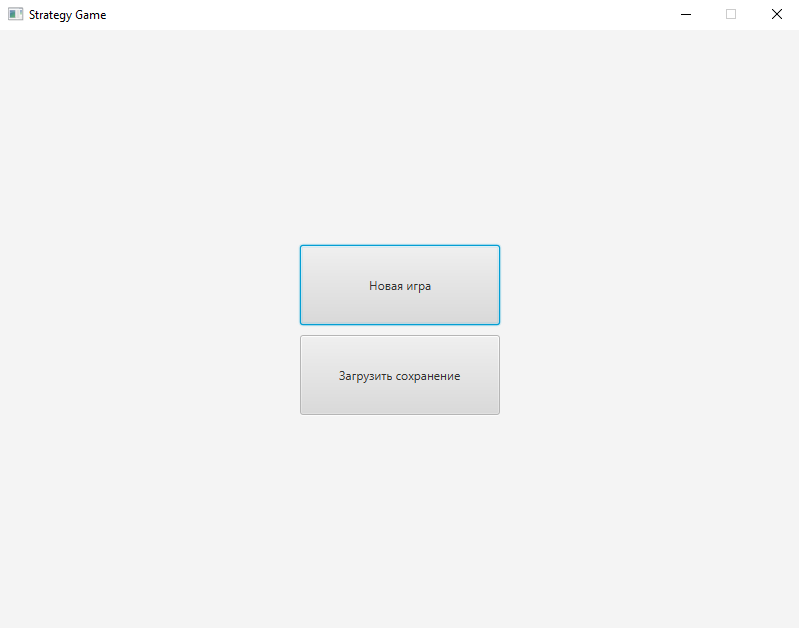
#### UML диаграмма классов

Подводя итоги по разработке пошаговой стратегии, хотелось бы также привести UML диаграмму классов. С её помощью можно наглядно увидеть зависимости внутри самого приложения.

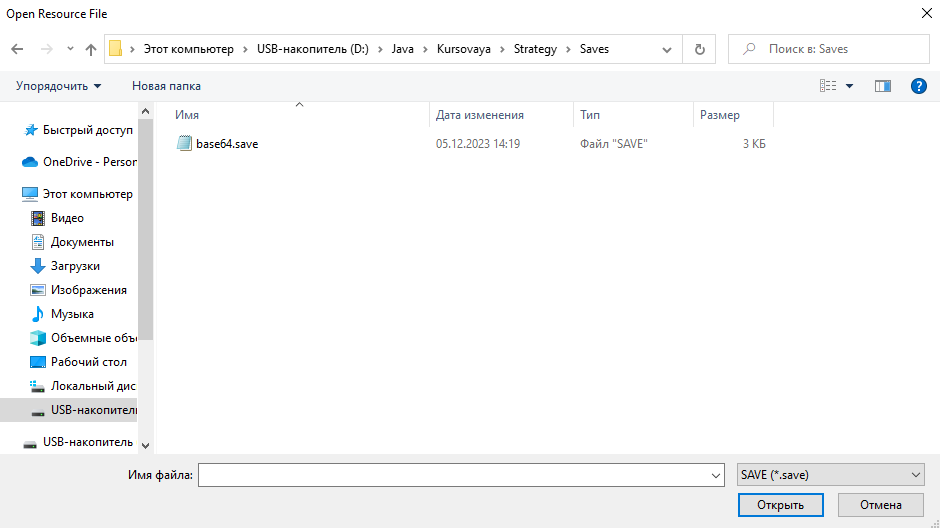
Как видно из диаграммы, классы ресурсов относятся к классу игры и активно в нём используются. В свою очередь класс игры относится к главному классу, где и реализован основной геймплей. Класс отвечающий за поля, а именно поля на карте и поля в меню покупок, сразу реализован в главном классе. В конце концов, главный класс реализовывается при запуске приложения с помощью надкласса «Application». Это особенность библиотеки JavaFX.

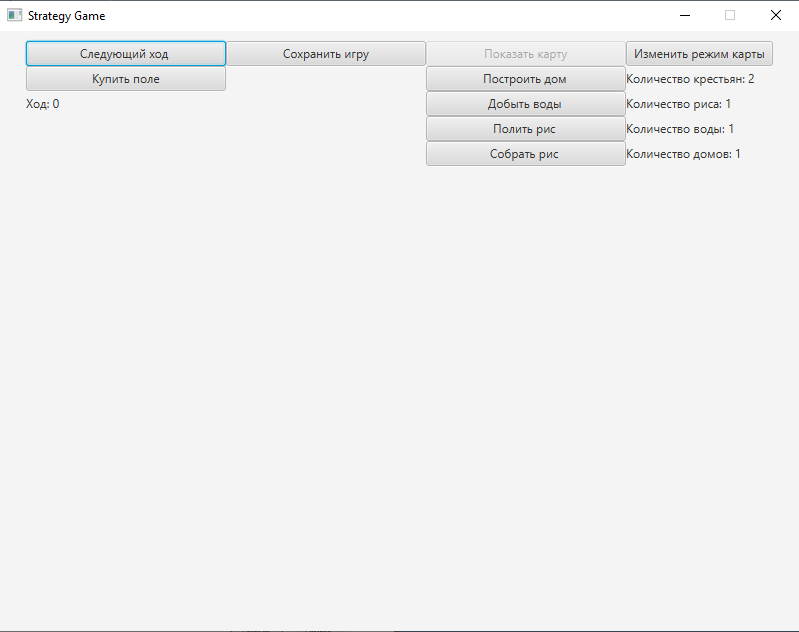
### Демонстрация готового продукта

Теперь, когда приложение полностью разработано, его можно показать во всей красе. В данном пункте я планирую показать в действии каждый аспект работы своей пошаговой стратегии.

Начнём с самого начала, с главного меню. Оно выглядит следующим образом:

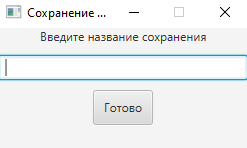
В нём присутствуют три кнопки: «Новая игра» и «Загрузить сохранение». При нажатии на «загрузить сохранение» откроется диалоговое окно, где необходимо выбрать файл сохранения:

При нажатии на «новая игра» у нас запустится игровое меню, а окно главного меню закроется. Плюсом к этому откроется карта в режиме захваченных территорий. Тоже самое произойдет и при выборе соответствующего сохранения. Единственное отличие будет в прогрессе игры. Меню игры выглядит следующим образом:

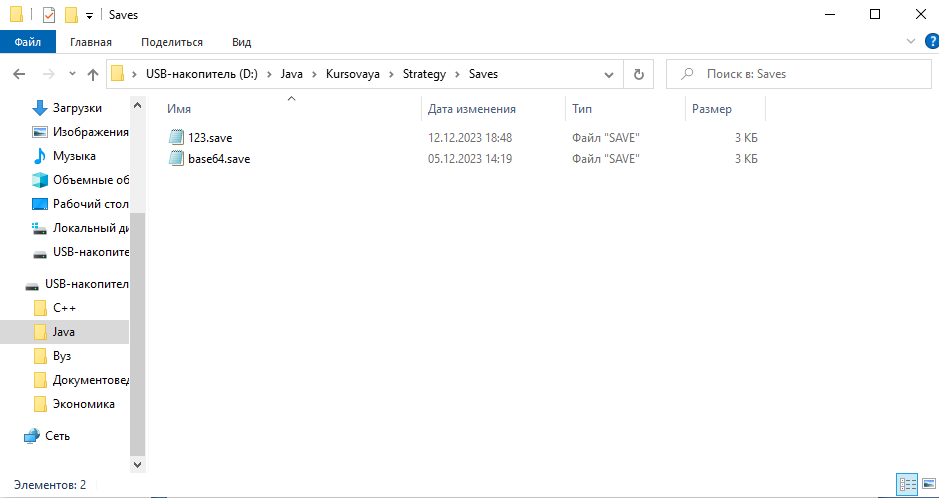
На нём мы видим множество кнопок. При нажатии на «следующий ход» будет пропущен ровно один ход, а счётчик этого показателя прибавится на единицу:



При нажатии на «сохранить игру» откроется окно с полем, куда надо ввести название сохранения:



Если ввести туда «123» и нажать «готово», то в папке «Saves» появится сохранение «123.save»:



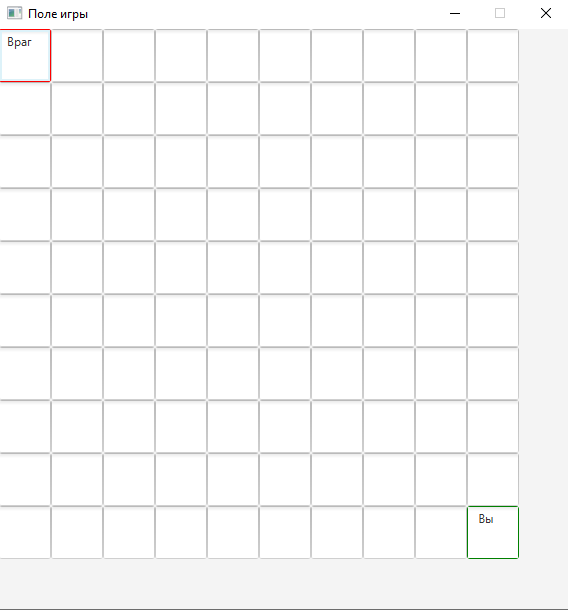
Нажатие на кнопку «построить дом» увеличивает количество домов на один, а количество остальных ресурсов уменьшает на единицу.

Кнопка «добыть воды» увеличивает количество воды на единицу взамен одного крестьянина.

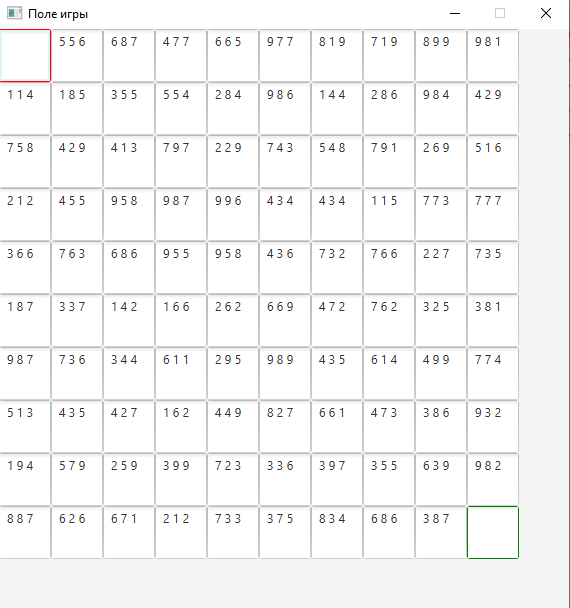
При нажатии на «полить рис», рис становится политым и при его сборе количество добычи увеличится в три раза.

Нажатие на «собрать рис» увеличит показатель риса на один или на три (в зависимости от статуса поливки) и уменьшит показатель крестьян также на единицу.

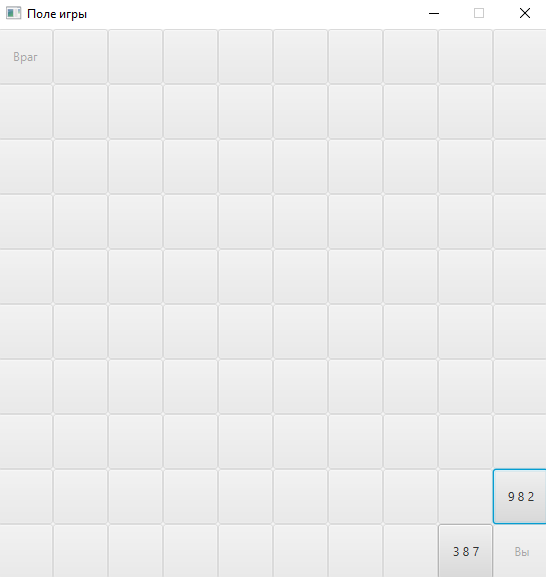
Дальше рассмотрим карту. Если она закрыта, то для показа используется кнопка «показать карту»:



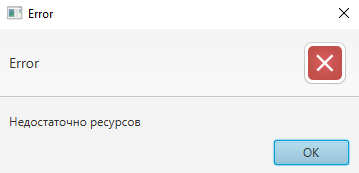
Как мы видим, у нас есть поле 10х10. В левом верхнем углу – враг. В правом нижнем – игрок. При нажатии на «изменить режим карты» наша карта поменяется, и вместо отображения принадлежности полей будет отображена стоимость незахваченных полей:



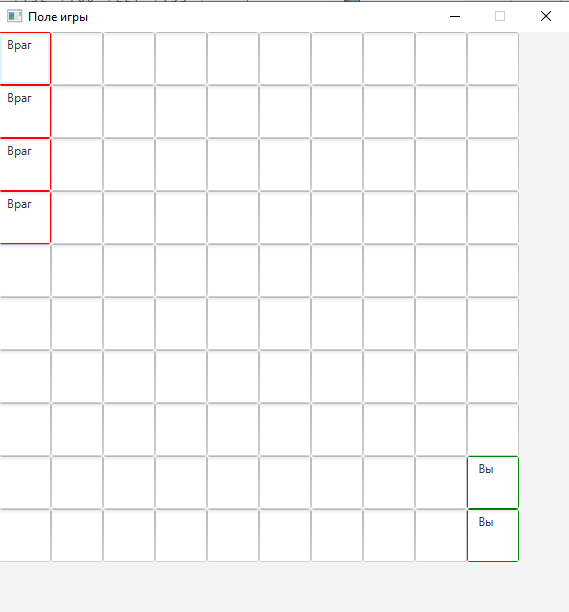
Как и было сказано, теперь на карте отображена стоимость ячеек. Дальше нажмем на «купить поле» в левой части игрового меню. У нас откроется интерактивная карта, где можно приобрести любое поле, прилегающее к уже имеющимся территориям:



Можно попробовать купить поле, однако за неимением необходимого количества ресурсов, у нас ничего не выйдет и вылезет окно ошибки, сообщающее об этом:



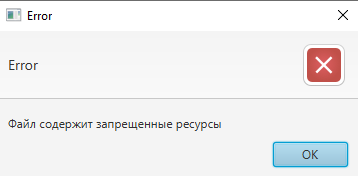
Однако если ресурсов достаточно, то купленное поле окрасится в зеленый и будет являться собственностью игрока:



Также, как мы видим, противник не стоит на месте и тоже захватывает провинции. В целом, показывать в готовом продукте больше нечего, осталось показать лишь пару окон об ошибке. Ошибка, при отсутствии выбранного файла:



Ошибка чтения файла сохранения, когда в файле содержатся запрещенные символы:



Это всё, что я хотел бы показать в данном разделе. Тонкости геймплея и игровой логики можно будет опробовать самим, скачав данную игру по данной ссылке на GitHub: https://github.com/Pelmen149/kursovaya

# Заключение

В рамках данной курсовой работы мы изучили и реализовали простую пошаговую стратегию на языке программирования Java. Задачей было ознакомиться с основами разработки игр на этом языке, а также применить полученные знания для создания игрового мира, управления игровыми объектами и реализации логики игрового процесса.

В процессе работы мы успешно применили принципы объектно-ориентированного программирования, разделив игровой мир на компоненты и создав классы, отвечающие за каждый из них. Мы определили структуру игрового мира, параметры объектов и реализовали их взаимодействие. Это позволило создать цельную игровую среду, где игрок может взаимодействовать с различными объектами и следить за изменениями в игровом состоянии.

Для разработки логики игрового процесса мы использовали условные операторы, циклы и алгоритмы принятия решений. Мы реализовали управление игроком и врагами.

Мы также изучили и применили инструменты для обработки ввода с клавиатуры, а также вывода информации на экран игрока. Это позволило сделать игру более удобной и понятной для пользователя.

В процессе работы над курсовой работой мы смогли углубить свои знания и навыки в разработке игр на языке Java. Мы познакомились с основными принципами разработки игр, научились применять объектно-ориентированный подход и использовать ключевые инструменты и библиотеки для создания игрового приложения.

В будущем, можно продолжить разработку игры, добавив новые функциональные возможности и улучшив графический интерфейс. Также, можно провести дополнительные исследования в области оптимизации и улучшения производительности игры.

Таким образом, выполнение данной курсовой работы позволило углубить наши знания в разработке игр на языке Java и приобрести опыт в создании простой пошаговой стратегии. Эта работа стала отличной отправной точкой для дальнейшего развития в области игровой разработки и программирования в целом.

# Список литературы

1. JavaFX Documentation Project : официальный сайт. – Последнее обновление 2022-06-26 22:06:47 UTC. – URL: <https://fxdocs.github.io/docs/html5/> (Дата обращения 09.12.2023)
2. Васильев, А.Н. Самоучитель Java c примерами и программами 3-е издание : учеб. пособие / А.Н. Васильев ; - СПб.: Наука и Техника, 2016. – 368 с. (Дата обращения 09.12.2023)
3. Введение в кодирование Base64 : сайт. – 2022. – URL: [https: https://skine.ru/articles/27784/](https://fxdocs.github.io/docs/html5/) (Дата обращения 12.12.2023)