Министерство образования Республики Беларусь

Учреждение образования

«Белорусский государственный университет

информатики и радиоэлектроники»

Кафедра информатики

Курсовая работа по теме «Игра жанра “Shoot `em up”»

Выполнил: Научный руководитель

студент группы 653502 Алексеев Юрий Игоревич

Галкин Илья Викторович

Минск, 2018

**Содержание**

[Введение 4](#_Toc514364214)

[1 Сравнение с аналогами 5](#_Toc514364215)

[**1.1 Crimsonland 5**](#_Toc514364216)

[**1.2 Nuclear Throne 6**](#_Toc514364217)

[**1.3 Hotline Miami 7**](#_Toc514364218)

[2 Анализ используемых технологий 8](#_Toc514364219)

[**2.1 Объектно-ориентированное программирование в разработке игр 8**](#_Toc514364220)

[**2.2 Использование паттернов проектирования в разработке игр 9**](#_Toc514364221)

[**2.2.1 Singleton 9**](#_Toc514364222)

[**2.2.2 Observer 10**](#_Toc514364223)

[**2.2.3 Factory method 11**](#_Toc514364224)

[**2.3 Использование сериализации в разработке игр 12**](#_Toc514364225)

[**2.3.1 Понятие сериализации 12**](#_Toc514364226)

[**2.3.2 Смысл сериализации 12**](#_Toc514364227)

[**2.3.3 Использование сериализации в проекте 13**](#_Toc514364228)

[3 Разработка программного средства 15](#_Toc514364229)

[**3.1 Схемы и диаграммы программного средства 15**](#_Toc514364230)

[**3.1.1 UML диаграмма классов проекта 15**](#_Toc514364231)

[**3.1.2 Блок-схема метода возрождения противников EnemySpawn класса GameManagement 16**](#_Toc514364232)

[**3.2 Описание структуры проекта. 17**](#_Toc514364233)

[**3.3 Описание классов и интерфейсов проекта. 18**](#_Toc514364234)

[**3.4 Описание работы программы 21**](#_Toc514364235)

[**3.4.1 Запуск игры 21**](#_Toc514364236)

[**3.4.2 Новая игра 22**](#_Toc514364237)

[**3.4.3 Общая механика игры 24**](#_Toc514364238)

[**3.4.4 Отдельные механики игры 25**](#_Toc514364239)

[**3.4.4.1 Механика зомби 25**](#_Toc514364240)

[**3.4.4.2 Оружие 27**](#_Toc514364241)

[**3.4.5 Конец игры и таблица очков 28**](#_Toc514364242)

[4 Тестирование программного средства 30](#_Toc514364243)

[**4.1 Описание концепций тестирования игрового программного средства 30**](#_Toc514364244)

[**4.2 Тестирование различных ситуаций программного взаимодействия 31**](#_Toc514364245)

[**4.2.1 Тестирование главного меню 31**](#_Toc514364246)

[**4.2.2 Тестирование перемещения и управления игрока 32**](#_Toc514364247)

[**4.2.3 Тестирование оружия 33**](#_Toc514364248)

[**4.2.4 Тестирование коллизии 34**](#_Toc514364249)

[**4.2.5 Тестирование сохранения игры 35**](#_Toc514364250)

[5 Руководство пользователя 37](#_Toc514364251)

[Заключение 38](#_Toc514364252)

[Список использованных источников 39](#_Toc514364253)

# **Введение**

В качестве темы курсового проекта была выбрана игра в жанре «Shoot `em up». Смысл игры заключается в том, чтобы защитить себя от орды противников и набрать как можно больше очков. Для защиты игроку предлагаются различные оружия. Оружия отличаются скорострельностью и поражающим уроном.

Действие игры всегда происходит на одной и то же карте. В самом начале игры игрок появляется в середине игровой карты и со всех сторон на него начинают идти враги. Количество врагов будет неограниченно, и они будут идти до тех пор, пока игра не закончится. По завершению игры, то количество очков, которые успел набрать игрок за игровую сессию, будут занесены в таблицу лидеров. Таким образом можно соревноваться с другими игроками по количеству набранных очков, либо же ставить новые личные рекорды.

Игра будет обладать простым и понятным интерфейсом с низким порогом вхождения. В плане использования системных ресурсов, игра будет требовать малую производительность компьютера. Управление производится при помощи клавиатуры и мыши.

Игры такого жанра хорошо развивают реакцию и тактическое мышление. С каждой новой попыткой игрок придумывает новые подходы к улучшению своего результата.

Целью курсовой работы является углубленное изучение языка Java, методов проектирования и разработки интерактивной 2D игры.

# **1 Сравнение с аналогами**

Данный жанр игр имеет огромное количество уникальных аналогов со своими особенностями. Рассмотрим некоторые из них.

## **1.1 Crimsonland**

****

Рисунок 1.1 Геймплей CrimsonLand

Игра CrimsonLand послужила прототипом для создания этого курсового проекта (рисунок 1.1). Игрок появляется в центре экрана с одним пистолетом и полным здоровьем. Через некоторое время начинают появляться различные враги со всех сторон экрана. За убийство врагов, игрок может получить новое оружие, очки опыта и бонусы. Игра заканчивается, когда у игрока заканчивается здоровье, которое теряется при попадании под атаки противников. Ведется таблица результатов, в которой записываются все рекорды игрока. Для игроков предоставляются несколько уровней сложности, которые влияют на количество врагов и шанс выпадения оружия.

Из особенностей игры можно выделить систему уровней персонажа, в которой игрок может повышать характеристики своего героя. Также в игре присутствуют различные бонусы, которые упрощают процесс игры. В последних версиях игры появились новые режимы игры, например, выживание, находясь только на одной точке.

Неплохим плюсом игры является то, что в игре присутствует режим для двух игроков [1].

## **1.2 Nuclear Throne**

Nuclear Throne также является игрой жанра «Shoot `em up» (рисунок 1.2). Каждую новую игру генерируется новая карта. Игроку предлагается 12 персонажей на выбор и у каждого персонажа есть свои особенности. Каждые несколько уровней появляются боссы и с некоторой вероятностью появляется главный босс игры. После устранения главного босса игра переходит на следующий уровень сложности. И так до тех пор, пока у игрока не закончится здоровье [2].



Рисунок 1.2 Геймплей Nuclear Throne

## **1.3 Hotline Miami**



Рисунок 1.5 Геймплей Hotline Miami

Hotline Miami является одним из самых известных представителей игр этого жанра (рисунок 1.5). Главная особенность игры заключается в проработанном сюжете и в разделении процесса игры на последовательные игровые уровни. Игроку предлагается несколько персонажей со своими особенностями и своей предысторией.

На каждой миссии игроку даются различные цели. В основном это устранить всех противников. Иногда требуется избавиться от сюжетного злодея. Игрок проигрывает при первом же получении урона, и игра предлагает начать игровой уровень заново [3].

# **2 Анализ используемых технологий**

## **2.1 Объектно-ориентированное программирование в разработке игр**

Современные игровые разработчики используют различные технологии для создания продукта, который будет актуален и востребован на игровом рынке. Для этого, зачастую, им приходится использовать все возможности современных компьютеров. Благодаря этому потребитель может получить максимально приближенную реальность к действительной. Иногда, чтобы получить желаемый результат, разработчику необходимо прибегнуть к методам, которые нарушают общепринятые нормы разработки. В современном мире это является распространённой проблемой, так как подобные методы могут повлиять на оптимизацию выпускаемого программного продукта. Чтобы избежать подобных проблем, необходимо понимать и разбираться в современных подходах разработки.

Объектно-ориентированное программирование является одним из подходов разработки программ. Данный подход позволяет организовать и создавать различные модели информационных объектов, что помогает разработчикам легче понимать логику разрабатываемого продукта. А при работе над крупным проектом это является особенно важным фактором.

В игровой индустрии объектно-ориентированное программирование используется повсеместно. Благодаря простой организации объектов, игровые модели получаются максимально понятными и целостными. Такой подход предоставляет возможность создавать множество однотипных объектов избегая повторения кода, как и создание уникального объекта со своими уникальными свойствами.

Три парадигмы ООП определяют всю разработку выпускаемой программы. Механизм наследования позволяет создавать новые типы объектов, основываясь на уже созданных моделях. Механизм инкапсуляции позволяет защитить важные данные от несанкционированного доступа. Механизм полиморфизма дает программам возможность не утруждать себя изучением вида объектов. Все это широко используется при разработке игр и во многом это удобнее структурного подхода программирования.

Среди минусов ООП можно выделить не самую быструю скорость выполнения кода и неэффективное распределение памяти по сравнению со структурным подходом программирования.

## **2.2 Использование паттернов проектирования в разработке игр**

Паттерн проектирования – это проверенный подход к решению определенной задачи, связанной с объектно-ориентированным программированием. Существует множество паттернов для разных ситуаций и многие из них не привязаны к конкретным языкам программирования. Но также существует паттерны для отдельных языков, которые дополняют эти языки новыми возможностями. В неумелых руках паттерн может принести немало проблем, а при правильном использовании такие подходы помогают быстро и правильно решать задачи реализации. Рассмотрим несколько из них.

### 2.2.1 Singleton

Основное назначение паттерна Singleton является гарантия того, что будет создан только лишь один экземпляр класса и к нему будет предоставлен глобальный доступ из любого места в программе. Различие синглтона и глобальной переменной лишь в том, что синглтон создает объект только при необходимости, когда глобальная переменная создается при первом же упоминании.

Основной смысл реализации синглтона заключается в создании в классе статической переменной, указывающей на единственный экземпляр этого класса. Этот объект можно получить, либо инициализировать только из специального метода, в котором прописана логика единственности объекта.

Хоть данный шаблон проектирования и является одним из самых простых, но это не исключает того факта, что он является одним из самых используемых паттернов. Потребность в хранении единственного объекта на класс не обходит и игровую разработку. Зачастую, даже объект игрока реализуется по паттерну Singleton, как это было сделано в данном игровом проекте.

### 2.2.2 Observer

Также является одним из самых популярных шаблонов проектирования. Основная идея заключается в том, что между объектами создается отношение «Один ко многим». Это означает, что существует такой объект-наблюдатель, который наблюдает за всеми объектами в системе. И как только один из этих объектов изменяется в процессе своего жизненного цикла, то наблюдатель сообщает всем заинтересованным частям системы об этом. Это позволяет изменять систему вслед за изменениями одного объекта. Такой подход позволяет избегать сильных связей между объектами и добавляет гибкость системе.

Данный шаблон проектирования используется, когда:

* Когда система состоит из множества классов, объекты которых должны находиться в согласованных состояниях.
* Когда общая схема взаимодействия объектов предполагает две стороны: одна рассылает сообщения и является главным, другая получает сообщения и реагирует на них. Отделение логики обеих сторон позволяет их рассматривать независимо и использовать отдельно друг от друга.
* Когда существует один объект, рассылающий сообщения, и множество подписчиков, которые получают сообщения. При этом точное число подписчиков заранее неизвестно и в процессе работы программы может изменяться.

Подобный шаблон хорошо вписывается в игровую концепцию. Объекты-менеджеры, которые управляют игрой должны постоянно оповещать все остальные объекты об изменениях в процессе игры. Таким образом, через можно сообщать всем игровым объектам, что игра переходит на новый уровень, либо игра была поставлена на паузу [4].

### 2.2.3 Factory method

Относится к порождающему типу паттернов. Данный шаблон проектирования предоставляет интерфейс для создания экземпляров некоторого класса. В момент создания наследники могут определить, какой класс инстанцировать. Иными словами, Фабрика делегирует создание объектов наследникам родительского класса. Это позволяет использовать в коде программы не специфические классы, а манипулировать абстрактными объектами на более высоком уровне.

Паттерн используется, когда:

* Класс не имеет информации о том, какой тип объекта он должен создать
* Класс передает ответственность по созданию объектов наследникам.
* Необходимо создать объект в зависимости от входящих данных.

В игровой разработке подобный подход можно использовать при создании объектов различных подвидов. Например, подобный интерфейс может отвечать за появление различных врагов на игровой карте, либо же используется при случайной генерации карты [5].

## **2.3 Использование сериализации в разработке игр**

### 2.3.1 Понятие сериализации

Сериализация - это процесс преобразования объекта в поток байтов для сохранения или передачи в память, базу данных или файл. Эта операция предназначена для того, чтобы сохранить состояния объекта для последующего воссоздания при необходимости. Обратный процесс восстановления объекта из байтов называется десериализацией [6].

2.3.2 Смысл сериализации

Зачастую при разработке программного средства разработчику требуется, чтобы результаты выполнения программы сохранялись для последующего использования, либо же требуется, чтобы была возможность обмена данными. Для этого используют технологию сериализации. Существует несколько альтернативных способов обмена данными, например, через собственный протокол, но такой подход является неэффективным, так как получателю нужно будет знать протокол для восстановления объекта. Сериализация является универсальным решением при передаче компонентов. В Java можно только сериализовать объекты, реализующие интерфейс Serializable, в котором нет методов, но он является флагом для системы, что такой объект можно сериализовать.

Из наиболее наглядных примеров можно выделить современные компьютерные игры. Большинство современных игр имеют достаточно долгую сюжетную линию, которая может растянуться на несколько часов прохождения. Для того, чтобы игроку не приходилось с каждым разом проходить игру сначала используется сериализация, которая сохраняет в файлы процесс прохождения игры.

### 2.3.3 Использование сериализации в проекте

В данном проекте сериализация используется для автоматического сохранения игры при ее закрытии и для возможности загрузить игру из того места, где игрок закрыл игру. Игра сохраняется в файл с именем игрока и, если игрок зайдет в игру в следующий раз под таким же именем, игра предложит возобновить игру. Также она используется для хранения таблицы рекордов. Игра сохраняет лучший результат игрока в таблицу при закрытии программы. Ниже приведен листинг кода программы, в котором реализован процесс сохранения игры в файл с именем игрока.

private void saveGame () {  
 try {  
 FileOutputStream fileOut = new FileOutputStream(playerName);  
 ObjectOutputStream out = new ObjectOutputStream(fileOut);  
 out.writeObject(status);  
  
 out.writeObject(currentObjects);  
 out.writeObject(player);  
 out.writeObject(score);  
 out.writeObject(player.getPos\_x());  
 out.writeObject(player.getPos\_y());  
 out.writeObject(player.getHp());  
  
 fileOut = new FileOutputStream("HighScore");  
 out = new ObjectOutputStream(fileOut);  
 out.writeObject(highScoreTable);

out.close();  
 fileOut.close();  
 }  
 catch(Exception e) {  
 System.err.println(e.getMessage());  
 }  
 }

Для сохранения объекта используется поток ObjectOutputStream. При его объявлении, в качестве аргумента передается поток записи байтов в файл FileOutputStream. Далее идет сохранение объектов в файл путем использования метода writeObject(), который сохраняет все принимаемые данные в файл. В приведенном коде сначала идет сохранение процесса игры игрока, а затем идет сохранение таблицы результатов.

Дессериализация происходит похожим образом. Игра загружает таблицу рекордов при заходе в игру, а процесс игрока, если игрок вышел заранее, проверяется при нажатии на кнопку начала игры. Для дессериализации используются те же потоки, что и для сериализации. Ниже приведет отрывок кода, загружающего из файла процесс игрока.

FileInputStream fileIn = new FileInputStream(playerName);  
ObjectInputStream in = new ObjectInputStream(fileIn);

…  
currentObjects =(ArrayList<GameObject>)in.readObject();  
player = (Player) in.readObject();  
score = (int) in.readObject();  
player.moveTo((double) in.readObject(), (double) in.readObject());  
for (GameObject e : currentObjects) {  
 e.moveTo(e.getPos\_x(), e.getPos\_y());  
}  
player.changeHP((int) in.readObject());

# **3 Разработка программного средства**

## **3.1 Схемы и диаграммы программного средства**

### 3.1.1 UML диаграмма классов проекта

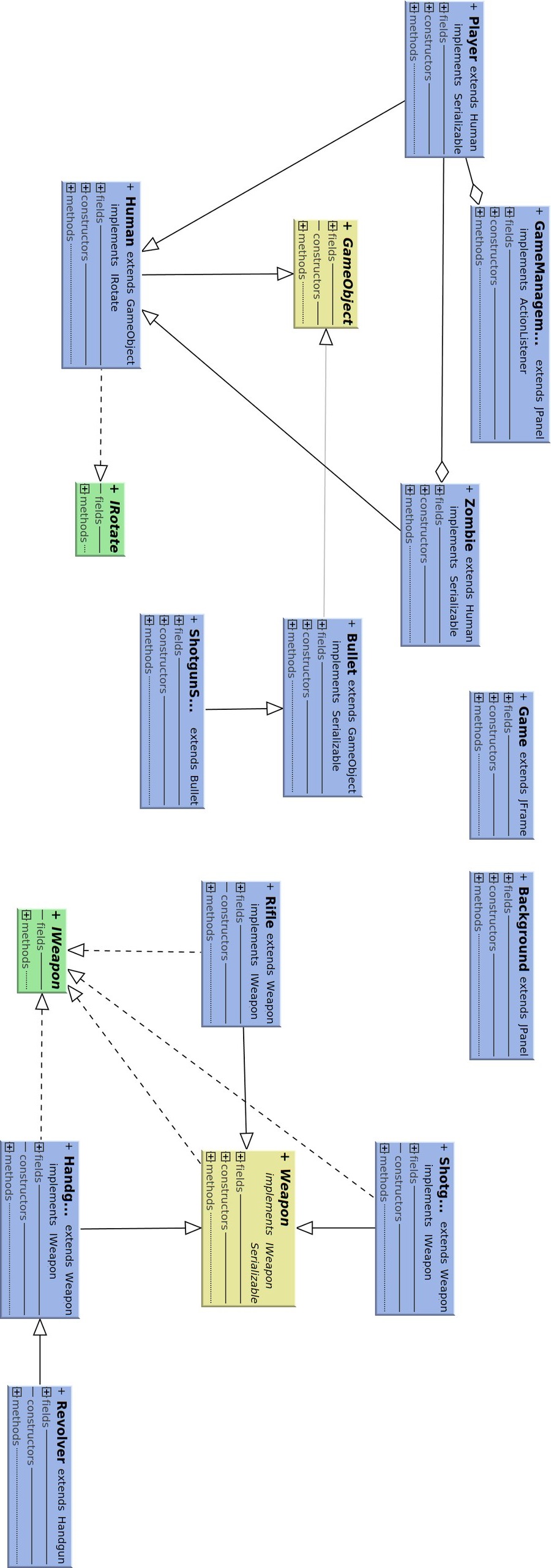


Рисунок 3.1 UML-диаграмма классов проекта

### 3.1.2 Блок-схема метода возрождения противников EnemySpawn класса GameManagement

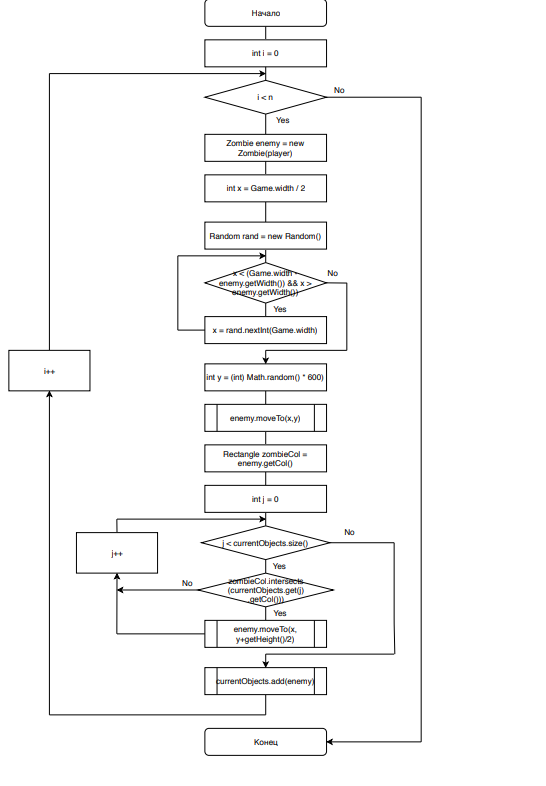


Рисунок 3.2 Блок-схема метода EnemySpawn

## **3.2 Описание структуры проекта**.

**. idea** – папка, содержащая настройки проекта.

**.out** – папка, в которой хранится скомпилированный проект.

**.scr** – папка, хранящая все ресурсы проекта.

**Zombie Rush.iml** – хранит информацию о модуле разработки. Содержит в себе пути к модулям, зависимости и другие настройки.

**Code** – папка, содержащая в себе весь код проекта.

**Resources** – папка, в которой хранятся все спрайты и все звуки игры.

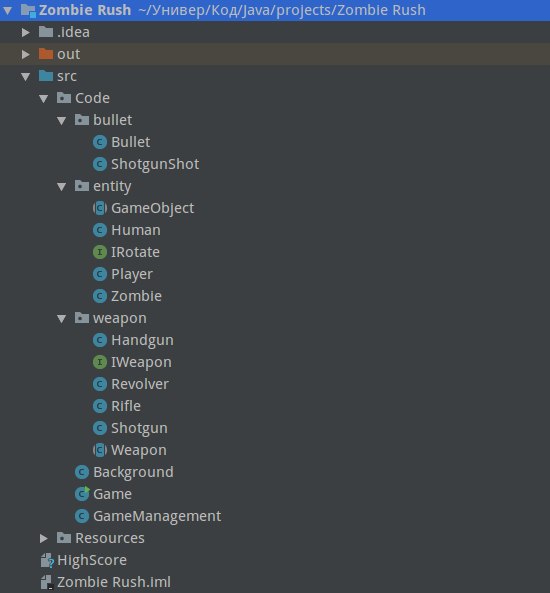


Рисунок 3.3 Структура проекта

## **3.3 Описание классов и интерфейсов проекта**.

**IRotate** – интерфейс, унаследованные классы которого гарантируют, что будут реализовывать методы поворота своих моделей.

**IWeapon** – интерфейс, унаследованные классы которого гарантируют, что будут реализовывать общие методы связанные со стрельбой.

**Human** – промежуточный класс, унаследованный от GameObject и реализующий IRotate. Хранит в себе общие методы и свойства игрока и противника.

**Player** – класс игрока, хранящий в себе методы управления и методы стрельбы, а также уникальные свойства объекта игрока. Унаследован от класса Human.

**Zombie** – класс, унаследованный от класса Human. Содержит искусственный интеллект и уникальные свойства зомби.

**Bullet** – класс пули, унаследованный от класса GameObject. Содержит свои уникальные свойства и собственный метод перемещения.

**ShotgunShot** – класс, унаследованный от Bullet. Описывает поведение выстрелов из дробовика.

**Weapon** – абстрактный класс, содержащий в себе стандартные методы и поля для всех типов оружия.

**Handgun, Revolver, Rifle, Shotgun** – классы оружия, унаследованные от Weapon и реализующие IWeapon. Хранят в себе свои уникальные свойства и свои методы стрельбы.

**GameManagement** – класс управляющий всей игрой. Обрабатывает все события, происходящие на игровой карте. Предоставляет главное меню и меню после завершения игры. Содержит методы сохранения и загрузки игровых сессий и управляет сохранением в таблице рекордов. Обрабатывает пересечения игрока с зомби и зомби с пулями. Перерисовывает все модели на карте.

**Game** – класс, запускающий игру. Также содержит в себе метод обработки любого звука игры.

**GameObject** – абстрактный класс, хранящий в себе общие методы и поля для всех объектов, взаимодействующих между собой на игровой карте.

Код класса GameObject:

public abstract class GameObject {  
 protected double pos\_x, pos\_y;  
 protected static int width = 100, height = 66;  
 public boolean visible = true;  
 protected int direction\_x, direction\_y;  
 protected Image image;  
  
 public void move() {  
 if(pos\_x + direction\_x - width/4 < 0 || pos\_x + direction\_x + width > Game.width || pos\_y + direction\_y -height/4 < 0 || pos\_y + direction\_y + height + 20 > Game.height)  
 return;  
 pos\_x += direction\_x;  
 pos\_y += direction\_y;  
 }  
 public void moveTo(double x, double y) {  
 this.pos\_x = x;  
 this.pos\_y = y;  
 }  
 public double getPos\_x() {  
 return pos\_x ;  
 }  
 public double getPos\_y() {  
 return pos\_y;  
 }  
 public int getWidth() {  
 return width;  
 }  
 public int getHeight() {  
 return height;  
 }  
 public boolean isVisible() {  
 return visible;  
 }  
 public void setVisible(boolean v) {  
 visible = v;  
 }  
 public Image getImage() {  
 if(null == image) {  
 String image\_location = "/Resources/" + this.getIcon();  
 ImageIcon ii = new ImageIcon(getClass().getResource(image\_location));  
 image = ii.getImage();  
 }  
 return image;  
 }  
 abstract public Rectangle getCol();  
 abstract public AffineTransform getAffineTransform();  
 abstract public String getIcon();  
}

Как видно из кода, данный класс является опорным для любого игрового объекта. Он хранит все необходимые методы для работы с визуальной моделью.

## **3.4 Описание работы программы**

### 3.4.1 Запуск игры

При первом запуске игры пользователю предлагается ввести имя, под которым он будет записываться в таблицу очков (рисунок 3.4). Далее это имя будет использоваться для сохранений и загрузки игры, а также для добавления в таблицу рекордов. Если игрок нажмет кнопку Cancel, то ему по умолчанию будет выдано имя Player.



Рисунок 3.4 Окно ввода имени

После ввода имени пользователь попадает в главное меню. Оттуда ему предоставляется возможность начать новую игру, открыть таблицу рекордов, либо выйти из игры (рисунок 3.5).

При первом заходе в главное меню, в папке с игрой создается новый файл с таблицей рекордов, если такового не было. В случае, если файл находился в папке с игрой до этого, то он загружается в игру.



Рисунок 3.5 Окно главного меню

### 3.4.2 Новая игра

При нажатии кнопки «New Game», в случае, если игрок уже раньше играл под таким именем и вышел из игры до того, как проиграл, игра предложит продолжить ту игру с места, откуда он вышел (рисунок 3.6). Игра загрузит его местоположение, его очки здоровья, состояние оружия и счет.



Рисунок 3.6 Запрос на продолжение игры

В случае, если игрок играет впервые, либо же начинает новую игру, то модель игрока появляется в центре игрового поля и вместе с ним со всех сторон появляются зомби (рисунок 3.7).



Рисунок 3.7 Появление игрока на карте

Во время игры сверху игрового окна можно заметить зеленую панель отображающую имя игрока, взятое оружие, текущее здоровье, количество патронов, счет и, в случае паузы, индикатор, показывающий, что игра в данный момент стоит на паузе (рисунок 3.8).



Рисунок 3.8 Верхняя информационная панель

### 3.4.3 Общая механика игры

Весь процесс игры устроен на работе таймера, который каждый определенный промежуток времени вызывает методы, которые задают игру. Ниже приведен листинг кода программы, в котором реализована обработка каждого тика таймера.

public void actionPerformed(ActionEvent event) {  
 for (int i = 0; i < currentObjects.size();i++){  
 GameObject e = currentObjects.get(i);  
 if (e.isVisible()) {  
 e.move();  
 } else {  
 currentObjects.remove(i);  
 }  
 }  
 for (int i = 0;i < player.bullets.size();i++){  
 Bullet b = player.bullets.get(i);  
 if (b.isVisible()) {  
 b.move();  
 } else {  
 player.bullets.remove(i);  
 }  
 }  
 ShotCD();  
 player.move();  
 Colshapes();  
 NewWave();  
 UpdateInterface();  
 repaint();  
 }

Весь процесс тика таймера представляет собой:

1. Проверка на видимость зомби. Если зомби видимый, то вызывает метод перемещения зомби. В случае, если зомби невидимый, то удаляет его из программы.
2. Также и для пуль.
3. Вызывает метод ShotCD, который высчитывает задержку между выстрелами оружия.
4. Вызывает метод перемещения у игрока.
5. Вызывается метод, проверяющий пересечения объектов.
6. Вызывается метод, проверяющий достаточно ли врагов на поле боя.
7. Вызывается метод, обновляющий верхнюю панель.
8. Последним вызывается метод, перерисовывающий на игровом поле все объекты

### 3.4.4 Отдельные механики игры

В игре реализовано множество различных механик для каждого объекта, рассмотрим самые интересные из них.

##### 3.4.4.1 Механика зомби

Зомби появляются в игре таким образом, что они не могут появится друг на друге и не могут появиться в центре. В противном случае, в самом начале игры, зомби могли бы появляться на игроке и сразу же забирали у него начальное здоровье.

Ниже приведет листинг программного кода, реализующего появление противников на игровой карте.

private void EnemySpawn(int n) {  
 for (int i = 0; i < n; i++)   
 {  
 Zombie enemy = new Zombie(player);  
 int x = Game.width / 2;  
 Random rand = new Random();  
 while (x < (Game.width - enemy.getWidth()) && x > enemy.getWidth())   
 {  
 x = rand.nextInt(Game.width);  
 }  
 int y = (int) (Math.random() \* 600);  
 enemy.moveTo(x, y);  
 Rectangle zombieCol = enemy.getCol();  
 for (int j = 0; j < currentObjects.size();j++)  
 {  
 if(zombieCol.intersects(currentObjects.get(j).getCol()))  
 enemy.moveTo(x, y + getHeight() / 2);  
 }  
 currentObjects.add(enemy);  
 }  
 }

Можно заметить, что позиция появления зомби выбирается случайно в определенных границах. Затем идет проверка всех зомби на соприкосновение с другими зомби. В случае, если зомби соприкасается с другим зомби, то тогда идет смещение позиции первого зомби.

Зомби запрограммированы так, что они всегда будут идти за игроком, до тех пор, пока его не достигнут. При соприкосновении игрока с зомби, зомби пропадает, но тем самым отнимает 1 единицу здоровья у игрока. Ниже приведен листинг кода, который направляет зомби в сторону игрока.

public void move() {  
 double diffX = player.getPos\_x()- pos\_x;  
 double diffY = player.getPos\_y() - pos\_y;  
 double angle = (float)Math.atan2(diffY, diffX);  
 pos\_x += 1.5 \* Math.cos(angle);  
 pos\_y += 1.5 \* Math.sin(angle);  
 watch(player.getPos\_x(),player.getPos\_y());  
 }

##### 3.4.4.2 Оружие

В игре игроку предлагается 4 вида оружия: пистолет, револьвер, дробовик и винтовка. Все оружия отличаются между собой скорострельностью, поражающим уроном, звуковыми эффектами, количеством патронов и дальностью стрельбы (рисунок 3.9). Оружия переключаются на специальные кнопки.



Рисунок 3.9 Сравнение стрельбы винтовки и дробовика

Каждое оружие обладает своими плюсами и своими минусами. Например, у дробовика сильная мощь, однако, через каждые 2 выстрела нужно его перезаряжать. При столкновении пуль оружия с врагами срабатывает скрипт, который добавляет одно очко в счет игрока и убирает врага с игровой карты.

Когда у игрока заканчиваются патроны, он теряет возможность стрелять. Для возвращения патронов оружие нужно перезарядить. Для этого нужно нажать специальную клавишу. Перезарядка, в среднем, длится 2 секунды и в это время игрок также не может стрелять.

### 3.4.5 Конец игры и таблица очков

Когда у игрока заканчиваются очки здоровья игра показывает количество очков, которые игрок успел набрать за игровую сессию (рисунок 3.10). Очки добавляются в таблицу рекордов только в том случае, если предыдущий результат игрока был хуже, чем теперешний.



Рисунок 3.10 Конец игры

После нажатия специальной клавиши, игрок переходит обратно в главное меню, где оттуда может начать новую игровую сессию. Существует одна особенность игры, которая позволяет начинать несколько раз игру с точки сохранения. Таким образом, если у игрока была сохранена игра на определенном моменте, он может несколько раз начинать игру именно с этого момента. Либо же игрок может отказаться от сохранения и начать игру сначала.

Из главного меню пользователь может открыть таблицу рекордов (рисунок 3.11). В ней расположены все максимальные рекорды игроков в порядке убывания.

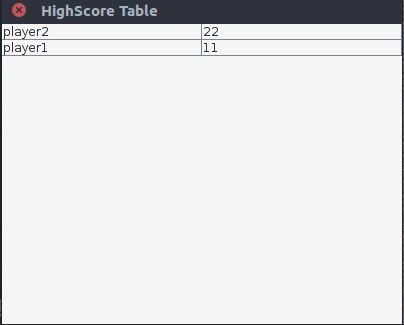


Рисунок 3.11 Окно таблицы рекордов

# **4 Тестирование программного средства**

## **4.1 Описание концепций тестирования игрового программного средства**

Одним из критериев успеха игрового программного продукта является фактор того, что выпускаемый продукт был максимально протестирован. Необходимо предусмотреть все варианты использования программы и предугадать все возможные ситуации в процессе эксплуатации продукта.

Игровое программное средство должно полностью соответствовать всем базовым канонам игровой индустрии. Банально, программа должна обработать все возможные ситуации и ни в коем случае не должна вызывать аварийное завершение. Также программа должна быть защищена от возможности нарушить стандартный игровой процесс. Такие ситуации отпугивают пользователей и это чревато тем, что программный продукт не будет использоваться в дальнейшем.

Тестирование игрового продукта проходит в несколько этапов:

1. Системное тестирование – тестирование всех программы в целом на работоспособность.
2. Модульное тестирование - тестирование всех модулей программы по-отдельности.
3. Тестирование на выявление всех неполадок и неординарных ситуаций.
4. Тестирование на совместимость с другими устройствами.
5. Нагрузочное тестирование – тестирование с целью выявить пик максимальной нагрузки на программу.
6. Стресс-тестирование – тестирование с целью выявления, что происходит с программой при переходе в нерабочее состояние
7. Тестирование локализации – проверка перевода программного средства.

В процессе разработки программы были протестированы все модули данного проекта. Каждый модуль тестировался как отдельно, так и в совокупности со всей программой. Была обеспечена защита от неправильного использования и аварийных завершений программы. Каких-либо проблем с производительностью не было замечено.

## **4.2 Тестирование различных ситуаций программного взаимодействия**

### Тестирование главного меню

* **Тест 1**

**Тестовая ситуация:**

Пользователь начинает новую игру.

**Исходный набор данных:**

Нажата кнопка начала игры с учетом того, что у пользователя нет файла сохранения.

**Ожидаемый результат:**

Игрок появиться в центре игрового поля.

**Фактический результат:**

Игрок появиться в центре игрового поля.

* **Тест 2**

**Тестовая ситуация:**

Пользователь начинает новую игру.

**Исходный набор данных:**

Нажата кнопка начала игры с учетом того, что у пользователя есть файл сохранения.

**Ожидаемый результат:**

Игроку предложат продолжить игру с того момента, где он закрыл программу.

**Фактический результат:**

Игроку предложили продолжить игру с того момента, где он закрыл программу.

* **Тест 3**

**Тестовая ситуация:**

Пользователь открывает таблицу рекордов.

**Исходный набор данных:**

Нажата кнопка открытия таблицы рекордов.

**Ожидаемый результат:**

Откроется таблица рекордов.

**Фактический результат:**

Была открыта таблица рекордов.

* **Тест 4**

**Тестовая ситуация:**

Пользователь закрывает программу через кнопку в меню.

**Исходный набор данных:**

Нажата кнопка закрытия игры. Игрок сыграл как минимум одну игру.

**Ожидаемый результат:**

Программа будет закрыта и в папке с игрой появиться файл с таблицей рекордов.

**Фактический результат:**

Программа закрылась и в папке с игрой появился файл с таблицей рекордов.

### Тестирование перемещения и управления игрока

* **Тест 1**

**Тестовая ситуация:**

Игрок находится в центре игрового поля.

**Исходный набор данных:**

Нажата кнопка перемещения вправо и кнопка перемещения вверх.

**Ожидаемый результат:**

Модель игрока переместиться в направлении правого верхнего угла.

**Фактический результат:**

Модель игрока переместилась в направлении правого верхнего угла.

* **Тест 2**

**Тестовая ситуация:**

Игрок находится рядом с левым краем игрового поля.

**Исходный набор данных:**

Нажата кнопка перемещения влево.

**Ожидаемый результат:**

Модель игрока не сможет переместиться влево.

**Фактический результат:**

Модель игрока не переместилась влево.

### Тестирование оружия

* **Тест 1**

**Тестовая ситуация:**

Игрок хочет выстрелить в направлении мыши.

**Исходный набор данных:**

Нажата кнопка выстрела.

**Ожидаемый результат:**

Игрок выстрелит в направлении мыши.

**Фактический результат:**

Игрок выстрелил в направлении мыши.

* **Тест 2**

**Тестовая ситуация:**

Игрок хочет переключить оружие на другое, а затем обратно на старое.

**Исходный набор данных:**

Нажата кнопка переключения оружия вперед, а затем кнопка переключения оружия назад.

**Ожидаемый результат:**

Будет переключено оружие игрока на новое, а затем на старое.

**Фактический результат:**

Оружие игрока переключилось на новое, а затем на старое.

* **Тест 3**

**Тестовая ситуация:**

Игрок хочет выстрелить.

**Исходный набор данных:**

Нажата кнопка выстрела. У оружия нет патронов, либо оружие перезаряжается.

**Ожидаемый результат:**

Оружие не выстрелит.

**Фактический результат:**

Оружие не выстрелило.

### Тестирование коллизии

* **Тест 1**

**Тестовая ситуация:**

Игрок потерял все очки здоровья после столкновения с зомби.

**Исходный набор данных:**

Здоровья игрока равняется 0.

**Ожидаемый результат:**

Игра остановиться и покажет на весь экран набранные очки игрока.

**Фактический результат:**

Игра остановилась и показала набранные очки игрока.

* **Тест 2**

**Тестовая ситуация:**

Зомби догоняет игрока.

**Исходный набор данных:**

Коллизионный прямоугольник зомби пересекается с прямоугольником игрока.

**Ожидаемый результат:**

Зомби исчезает. У игрока отнимается 1 очко здоровья.

**Фактический результат:**

Зомби исчез. У игрока отнялось 1 очко здоровья.

* **Тест 3**

**Тестовая ситуация:**

Игрок стреляет в зомби.

**Исходный набор данных:**

Пуля игрока пересекает коллизионный прямоугольник зомби.

**Ожидаемый результат:**

Зомби исчезает. Игроку прибавляется 1 очко к счету.

**Фактический результат:**

Зомби исчез. У игрока прибавилось 1 очко к счету.

### Тестирование сохранения игры

* **Тест 1**

**Тестовая ситуация:**

Игрок находится в игровой сессии.

**Исходный набор данных:**

Игрок закрывает игру не через главное меню.

**Ожидаемый результат:**

Игра сохранит результат игрока в папке с игрой.

**Фактический результат:**

Игра сохранила результат игрока в папке с игрой.

* **Тест 2**

**Тестовая ситуация:**

Игроку после игровой сессии показывает количество набранных очков.

**Исходный набор данных:**

Игрок закрывает игру через крестик.

**Ожидаемый результат:**

Игра не сохранит игровую сессию.

**Фактический результат:**

Игра не сохранила игровую сессию.

# **5 Руководство пользователя**

Управление моделью игрока в игровой сессии производится путем нажатия клавиш WASD. Стрельба производиться по нажатию клавиши «Space». Выбор направления стрельбы определяется положением курсора на игровом поле. Нажатие клавиши «Escape» поставит игровую сессию на паузу.

Игра заканчивается, когда у игрока заканчивается здоровье. После окончания игры игроку покажет сколько очков он успел набрать за игровую сессию и если это рекорд, то он будет добавлен в таблицу очков.

# **Заключение**

По результатам курсового проекта было создано игровое приложение в жанре «Shot `em up». Во время разработки программы были учтены все требования, предъявляемые к игровому приложению такого рода.

Приложение было написано с использованием принципов объектно-ориентированного программирования, что делает код проекта максимально понятным и легко модифицируемым.

В результате разработки проекта были изучены основы визуального программирования на Java, закреплены общие знания по данному языку программирования, были изучены основные методы тестирования программных средств и были изучены шаблоны проектирования и их использование в игровой разработке.

Данный курсовой проект является лишь первой версией разрабатываемой игры. В дальнейшем планируется добавление нескольких новых модулей и новых механик в проект. Игры подобного жанра предоставляют удобную площадку для реализаций интересных идей.

# **Список использованных источников**

[1] Сайт Crimsonland [Электронный ресурс]. – Электронные данные. – Режим доступа: http://crimsonland.com

[2] Сайт Nuclear Throne [Электронный ресурс]. – Электронные данные. – Режим доступа: http://nuclearthrone.com/

[3] Сайт Hotline Miami [Электронный ресурс]. – Электронные данные. – Режим доступа: http://hotlinemiami.com/

[4] Сайт [Metanit](https://sohabr.net/posts/all) [Электронный ресурс]. – Электронные данные. – Режим доступа: <https://metanit.com/sharp/patterns/3.2.php>

[5] Сайт Metanit [Электронный ресурс]. – Электронные данные. – Режим доступа: <https://metanit.com/sharp/patterns/2.1.php>

[6] Сайт Microsoft Docs [Электронный ресурс]. – Электронные данные. – Режим доступа: <https://docs.microsoft.com/ru-ru/dotnet/csharp/programming-guide/concepts/serialization/>