Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное
учреждение высшего образования
«ТОМСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ СИСТЕМ
УПРАВЛЕНИЯ И РАДИОЭЛЕКТРОНИКИ» (ТУСУР)

Кафедра автоматизации обработки информации (АОИ)

К ЗАЩИТЕ ДОПУСТИТЬ
Заведующий кафедрой АОИ
канд. экон. наук, доцент
_____А.А. Сидоров
«_____»_____2022 г.

Бакалаврская работа по направлению 09.03.04
«Программная инженерия»

# КОМПЬЮТЕРНАЯ ИГРА «2D–ПЛАТФОРМЕР» НА UNITY

Студент гр. 428-3
_____ Н.Е. Залогин
«_____» _____2022 г.

Руководитель
доцент кафедры АОИ, канд. техн. наук
_____ Т.О. Перемитина
«_____» _____2022 г.

Томск 2022

Министерство науки и высшего образования Российской Федерации

Федеральное государственное бюджетное образовательное учреждение высшего образования

«ТОМСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ СИСТЕМ УПРАВЛЕНИЯ И РАДИОЭЛЕКТРОНИКИ» (ТУСУР)

Кафедра автоматизации обработки информации (АОИ)

УТВЕРЖДАЮ
Заведующий кафедрой АОИ
канд. экон. наук, доцент
_____А.А. Сидоров
«_____»_____2022 г.

ЗАДАНИЕ
на выполнение бакалаврской работы

студенту Залогину Никите Евгеньевичу группы 428-3,
факультета систем управления (ФСУ)

1. Тема работы: «Компьютерная игра «2D – Платформер» на Unity»
утверждена приказом по вузу от _____ №_____

2. Срок сдачи работы на кафедру: _____

3. Содержание работы (перечень подлежащих разработке вопросов):
   1) провести анализ предметной области и обзор успешных аналогов;
   2) провести анализ сред разработки и выбрать наиболее подходящую;
   3) произвести проектирование архитектуры;
   4) разработать игру жанре 2D платформер.

4. Дата выдачи задания: _____

Руководитель:
доцент кафедры АОИ, канд. техн. наук _____ Т.О. Перемитина

«____» _____ 2022 г.

Задание принял к исполнению:
студент гр. 428-3 Н.Е. Залогин _____

«____» _____ 2022 г.

**РЕФЕРАТ**

Выпускная квалификационная работа, 63 страницы, 38 рисунков, 3 таблицы, 19 источников.

КОМПЬЮТЕРНАЯ ИГРА, 2D ПЛАТФОРМЕР, UNITY, ДЕСКТОП ПРИЛОЖЕНИЕ.

Объектом разработки является компьютерная игра в жанре 2D платформер.

Цель работы – создание компьютерной игры в жанре 2D платформер.

Выпускная квалификационная работа содержит:

– анализ предметной области;

– обзор и анализ аналогов;

– обзор и анализ сред разработки игр;

– проектирование архитектуры;

– разработку компьютерной игры в жанре 2D платформер.

В результате работы была создана компьютерная игра в жанре 2D платформер с основным функционалом аналогичных игр.

Выпускная квалификационная работа выполнена в текстовом редакторе Microsoft Word 2021.

Средства реализации компьютерной игры – среда разработки игр Unity, среда разработки Visual Studio 2022, язык программирования C#.

**ABSTRACT**

The final qualifying work contains 63 pages, 38 figures, 3 tables, 19 sources.

COMPUTER GAME, 2D PLATFORMER, UNITY, DESKTOP APPLICATION.

The object of development is a computer game in the 2D platformer genre.

The purpose of the work is to create a computer game in the 2D platformer genre.

– The final qualifying work contains:

– domain analysis;

– review and analysis of analogues;

– review and analysis of game development environments;

– architecture design;

– development of a computer game in the 2D platformer genre.

The result of the work is a created computer game in the 2D platformer genre with the main functionality of similar games.

The descriptive part of the final qualifying work was done in a text editor Microsoft Word 2021.

Computer game implementation tools - Unity game development environment, Visual Studio 2022 development environment, C# programming language.

**Оглавление**

# Введение

Видеоигры – это игры на различных электронных устройствах будь то персональные компьютеры (ПК), мобильные устройства или консоли (PlayStation, Xbox, Nintendo).

В период с 2016 по 2020 г. мировой рынок игровой индустрии вырос в 1,5 раза, среднегодовой темп роста составил 12%, а количество пользователей достигло 2,7 миллиарда человек, что на 30% больше показателей за 2016 год. В сравнении с другими сегментами индустрии развлечений производство и дистрибуция видеоигр уступает в темпах роста только OTT-телевидению (интернет телевиденье). Ежегодно выпускаются сотни новых продуктов: только на сервисе Steam в 2020 году было выпущено более 10 тыс. видеоигр, а количество игровых приложений в App Store к концу 2020 года составило 298 тысяч. Количество разработчиков и издателей на мировом рынке превышает 4 000 компаний [1].

На конец 2021 года, по сравнению с 2020 годом, мировой рынок видеоигр вырос на 1.4% и достиг 180 миллиардов долларов. Рост обеспечил мобильный сегмент, который вырос на 7.3%, в то время как консольный и ПК сегменты уменьшились на 6.6% и 0.8% соответственно [2]. Российский же сегмент увеличился за 2021 год на 9% и достиг 177 миллиардов рублей. Также как и на мировом рынке, наибольший рост показал сегмент мобильных видеоигр с ростом в 14%, рост ПК сегмента составил 4.2%, а консольного 8% [3].

Для данной работы была выбрана игра в жанре 2D – платформер. Данный жанр появился в начале 1980-х годов, но до сих пор сохраняет свою актуальность, а игры в этом жанре продолжают разрабатываться, как крупными студиями, так и индивидуальными разработчиками.

Целью данной работы является закрепление профессиональных умений и знаний в области индустриальной разработки ПО путем разработки компьютерной 2D игры в жанре платформер с помощью среды Unity.

Для достижения поставленной цели необходимо решить следующие задачи:

– провести анализ предметной области с обзором аналогов;

– определить функциональные требования;

– выбрать среду разработки;

– спроектировать игру.

Таким образом была обозначена актуальность выбранной темы и поставлены задачи.

# 1 ТЕОРЕТИЧЕСКАЯ ЧАСТЬ

## 1.1 Анализ предметной области

Платформер – один из первых жанров компьютерных игр, появившийся в начале 1980-х годов, в которых основную часть игрового процесса составляют прыжки по платформам, лазанье по лестницам и сбор различных предметов, необходимых для прохождения игры. Многие представители подобного жанра характеризуются нереалистичностью, рисованной мультяшной или пиксельной графикой, однако современные платформеры могут иметь и достаточно реалистичное графическое исполнение.

Платформеры могут быть разделены на несколько видов:

– по глубине перемещения – в трёхмерных платформерах персонаж может перемещаться в трёх измерениях, в двухмерных платформерах только в двух, помимо этого в некоторых играх герой может углубляться и приближаться, переключаясь между несколькими линиями или вовсе менять ракурс камеры, для перемещения по двухмерной проекции трехмерного окружения;

– по виду графики – двухмерные или трёхмерные. Современные двухмерные платформеры, с перемещением только в двух измерениях, часто используют трёхмерную графику. Бывают также трёхмерные платформеры с перемещением в трёх измерениях с двухмерной графикой, обычно в изометрической перспективе, как, например, «Sonic 3D Blast»;

– по свободе – линейные и свободные. В линейных, персонаж должен пройти определённый путь с начала до конца уровня. Если это путь справа налево и слева направо, то игра называется сайд-скроллер (с горизонтальным движением экрана). В свободных платформерах персонаж может перемещаться по уровню без ограничений и для победы часто нужно посетить разные места в любой последовательности. Такие игры зачастую имеют жанр,

отличающийся механикой открытия новых путей на карте с помощью появляющихся в будущем способностей.

Одним из наиболее популярных поджанров платформеров является платформер-приключенческая игра. Такие игры сочетают в себе основные черты платформеров и элементы приключенческих или ролевых (РПГ) игр. Зачастую такие элементы включают в себя возможность свободного исследования карты с закрытыми областями, доступ к которым может быть получен при достижении определенного уровня, изучении необходимой способности или при нахождении нужного предмета. Наиболее известными из подобных игр являются «Metroid» и «Castlevania» из названий которых и появилось обозначение подобных игр – «Metroidvania» (метроидвания).

## 1.2 Обзор аналогов

### 1.2.1 Dead Cells

Dead Cells – это экшн-платформер в стиле rogue-lite, вдохновленный другим представителем данного жанра, под названием «Castlevania», разработанный французской студией Motion Twin. Сами разработчики называют жанр своей игры как «RogueVania» (игра с интенсивным геймплеем и только одной жизнью на попытку) и преподносят игру как «Soulslike» (игры выделяющиеся высоким уровнем сложности и упором на повествование истории, которая заложена в окружающей среде) [4].

На рисунке 1.1 изображена игра Dead Cells.


Рисунок 1.1 – Dead Cells

В ходе игры предстоит выбраться из лабиринта пройдя множество уровней, выполненных в духе двухмерного платформера и генерирующихся процедурным образом. По ним разбросаны враги и различные сокровища, в том числе оружие со случайно генерируемыми характеристиками. Подобно играм в жанре RogueLike, у персонажа Dead Cells только одна «жизнь» — если он погибнет, игрок будет вынужден начать игру с самого начала без собранного за предыдущую попытку снаряжения и без большей части накопленных ресурсов. Но некоторые единожды полученные способности, открывающие доступ к новым, ранее недоступным областям игры, переносятся и в последующие прохождения. Концепция игры сильно менялась в ходе разработки: первоначально Dead Cells задумывалась в жанре «tower defense» с обороной базы от полчищ зомби, но позже была полностью переработана под вдохновением от The Binding of Isaac. Постоянно растущее разнообразие вооружения по мере прохождения новых уровней, а также способностей и модификаторов с каждой новой попыткой открывает новые просторы для экспериментов. А множество различный видов существ, с которыми придется бороться, не дадут заскучать [4, 5].

Для небольшой студии с ограниченными ресурсами было бы затруднительно рисовать огромное количество пиксельных текстур для многочисленных видов вооружения и противников, в связи с этим главный и единственный художник компании придумал создавать все анимации используя самостоятельно разработанный конвертер, который бы переводил базовую анимированную 3D модель в пикселизированный вид путем рендера (получение изображения по его описанию) изначального изображения в малом размере без сглаживания. Используя данный метод, удалось во много раз сократить время разработки, по сравнению с традиционной отрисовкой каждого кадра вручную, а также упростить процесс корректирования анимаций [6].

Dead Cells получила чрезвычайно высокие оценки критики. Обозреватели отмечали привлекательную визуальную часть, сложный, но увлекательный игровой процесс и чрезвычайно высокое качество игры в целом.

### 1.2.2 Ori and the Blind Forest

Игра представляет собой двухмерный платформер. Игрок управляет персонажем по имени Ори и защищающим его духом по имени Сейн, который следует за Ори. С помощью Сейна можно атаковать врагов, в которых он выпускает заряды «духовного пламени». Кроме того, Сейн может сделать кратковременный мощный выброс энергии, поражающий всех врагов поблизости от Ори и разрушающий некоторые объекты. Сам Ори изначально умеет только прыгать, но в процессе игры он сможет научиться карабкаться по стенам, нырять под воду, парить в воздухе, совершать двойные-тройные прыжки и использовать энергию, чтобы выстреливать собой или отталкивать врагов и предметы.

Игровая карта, представляющая различные участки леса, загружается целиком, и игрок волен идти туда, куда ему хочется. Однако, не открыв

определённую для данного участка способность героя, игрок в любом случае не сможет туда попасть.

Изначально игрока выбрасывают в игровой мир без каких-либо пояснений о том, что нужно делать, и методом проб и ошибок необходимо искать решение самостоятельно. После того как игрок найдёт Сейна, он уже будет получать некоторые подсказки от него.

На рисунке 1.2 изображена игра Ori and the Blind Forest.

Рисунок 1.2 – Игра Ori and the Blind Forest

У Ори есть показатель здоровья и показатель энергии, представленные в виде ячеек. Изначально, Ори очень слаб, поэтому имеет всего три ячейки здоровья и одну ячейку энергии. Во время исследования мира, игрок будет находить дополнительные ячейки для этих показателей. Способности Ори можно будет улучшать находя очки умения.

Всего доступно три направления развития персонажа, которые допускается развивать одновременно. Одно из них отвечает за боевые навыки, второе — за совершенствование поиска тайников, третье — за

вспомогательные способности и навыки (дыхание под водой, уменьшение затрат ресурсов при выполнении каких-либо действий и т. д.).

Игрок может сохраниться в любой момент, что является также частью игрового процесса, так как контрольные точки расставлены на карте очень далеко друг от друга, и после гибели игрового персонажа придётся начинать с последней сохранённого момента. На сохранение тратится ячейка энергии, поэтому игрок в начале вынужден делать это избирательно. Ближе к концу игры, когда ячеек, как правило уже больше, сохранение уже не представляет большой проблемы.

Ori and the Blind Forest разрабатывалась 4 года студией Moon Studios. Программист Дэвид Кларк заявил, что «Ori and the Blind Forest» — это дань уважения таким классическим приключенческим играм как Rayman и Metroid.

При написании сюжета разработчики руководствовались такими произведениями как «Король Лев» и «Стальной гигант». Сама игра написана на игровом движке Unity. Действие происходит на одной большой карте без видимых подгрузок частей игрового мира, поэтому перед игрой необходимо выждать достаточно длительную загрузку [7].

Во многих обзорах по достоинству оценили музыку Ори, отмечая насколько она хороша и как динамически изменяется во время геймплея. Только главная деталь заключается в том, что никакого динамического или адаптивного музыкального сопровождения Ори нет. Нет заранее написанных слоев и ритмической основы, на которую они накладываются. Отчасти это обусловлено художественным решением команды и лично композитора игры Гарета Кокера. Для Ори было написано порядка 125 композиций. Такое количество и создает иллюзию интерактивного сопровождения, которая жестко продумана и завязана на триггеры, а сам композитор лично потратил около 500 часов в игре только на то, чтобы четко уложить музыку в линию повествования.

Далее, где-то в процессе, разработчики намеренно отказались от так называемой «боевой музыки», которая обычно появляется в других играх во

время перестрелок и как бы заставляет двигаться, подталкивает к нападению. Все дело в том, что в процессе создания игры, авторы поняли, что схватки могут происходить очень часто, и постоянная смена музыкального сопровождения выглядела бы ужасно. Поэтому было принято решение, что боевая музыка должна включаться только в те моменты, когда тебе действительно нужно по сюжету кого-то убить, а большую часть всех схваток вообще можно проигнорировать. То есть игра намеренно не стимулирует в игроке агрессию, а скорее наводит на мысль, что главное в ней это постоянное созерцание, исследование и движение [7, 8].

### 1.2.3 Rogue Legacy

Rogue Legacy – компьютерная игра в жанре платформера с элементами roguelike. Игрок исследует процедурно-сгенерированные локации, добывает золото, оружие и броню для своего персонажа, а также сражается с врагами. В Rogue Legacy присутствуют четыре локации – замок, лес, башня и подземелье. В каждой из них свой уровень сложности, свои враги и свой босс. Цель – победить четырёх боссов в разных локациях, после чего игрок получает возможность сразиться с пятым, финальным боссом [9, 10].

На рисунке 1.3 изображена игра Rogue Legacy.

Рисунок 1.3 – Игра Rogue Legacy

В случае гибели персонажа игрок должен выбрать одного из трёх новых персонажей – детей погибшего героя. В очередном прохождении игрок управляет уже этим наследником. У каждого из героев имеется свой класс, заклинание, которое он может использовать, а также свои особенности. Например, персонаж может страдать болезнью Альцгеймера, СДВГ, слабоумием и так далее [9].

После гибели персонажа игрок может использовать собранное им золото для улучшения характеристик будущих персонажей, а также потратить на покупку нового оружия, брони и рун, каждая из которых придаёт персонажу определённую способность. Чтобы получить доступ к определённой руне, игроку необходимо разблокировать её, для этого нужно найти в игре комнату с магическим сундуком и выполнить определённое задание, например, дойти до сундука без единого прыжка или уничтожить всех врагов в комнате. Если условие выполнено успешно, персонаж сможет открыть магический сундук и активировать найденную руну, позволяющую персонажу, например, совершать двойной прыжок. Помимо рун, по ходу игры можно обнаружить сундуки с чертежами. После их открытия игрок получает доступ к мечу или

броне, которые впоследствии сможет надеть на своего персонажа. Оружие и броня повышают атаку, защиту и ману игрока. Некоторые элементы брони и мечи могу дать игроку те или иные способности, такие как повышение шанса критического удара или восстановление очков здоровья после уничтожения врага [9].

Перед тем персонаж войдёт в замок, он может воспользоваться услугами одного из трёх персонажей, стоящих у входа в замок. Среди них кузнец, волшебница и архитектор. У кузнеца игрок может надевать на своего персонажа мечи, шлемы, нагрудники, поножи с перчатками и плащи, у волшебницы игрок может повесить на своего персонажа руны, придающие персонажу дополнительные способности, а архитектор может восстановить игровую карту с предыдущей попытки, забирая в качестве платы 40% золота, которого игрок добудет при следующем запуске уровня [9].

Rogue Legacy была разработана двумя братьями; вся студия Cellar Door Games состояла только из них самих. В прошлом братья занимались созданием браузерных игр, но Rogue Legacy была на тот момент крупнейшим проектом Cellar Door Games. Разработка игры заняла 18 месяцев, тогда как самый большой проект из предыдущих занял лишь 4 месяца. На Rogue Legacy было потрачено в общей сложности 14 878 долларов, затраты были покрыты спустя первый час после начала продажи игры. За первую неделю после выхода было продано свыше 100 тысяч копий игры [11].

Источниками вдохновения для игры послужили такие игры, как Demon's Souls и Dark Souls. Братья хотели сделать двухмерную версию этих игр. Тедди Ли сравнивал процедурную генерацию замка в Rogue Legacy со схожей независимой игрой – Spelunky, отметив, что в собственной игре братья старались сделать её относительно щадящей по отношению к игроку и простой в освоении, позволяя игроку непрерывно совершенствовать своего персонажа, не начиная развитие каждый раз «с нуля». В связи с этим разработчиками было принято решение удалить из Rogue Legacy изначально задуманную систему накопления опыта, который давал бы игроку бонусы [12].

### 1.2.4 Cuphead

Cuphead – это shoot'em-up игра, основанная на непрерывных боях с боссами. Персонаж игрока Капхед проиграл в споре с дьяволом и пытается вернуть долг. Игра имеет вид разветвляющейся последовательности уровней. У Капхеда бесконечное число попыток и он не теряет оружие после гибели. У главного героя игры есть способность парирования различных объектов, закодированных розовым цветом. Успешные парирования заполняют специальный счётчик, который позволяет ему выполнять специальную способность.

На рисунке 1.4 изображена игра Cuphead.

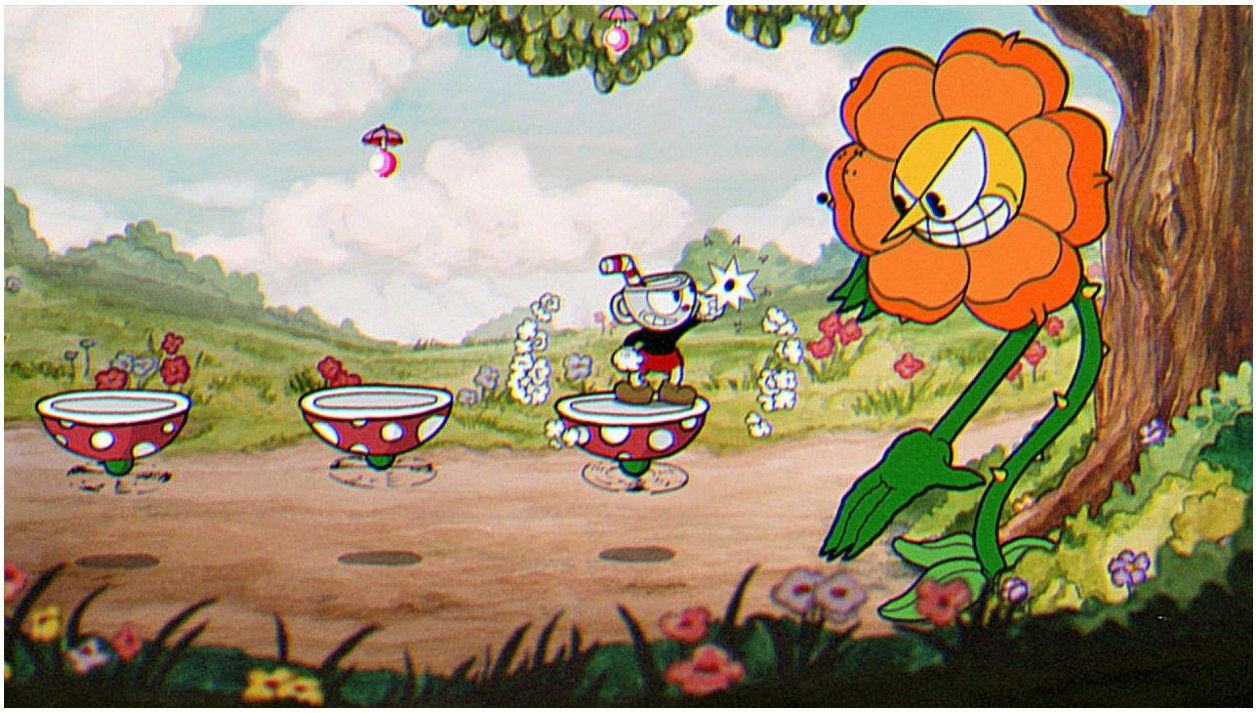

Рисунок 1.4 – Игра Cuphead

Cuphead – первая игра инди-студии StudioMDHR Entertainment, состоящая из двух братьев Молденхауэров. Игра была вдохновлена мультфильмами от Fleischer Studios, Disney, и мультипликаторами Абом Айверксом, Гримом Натвиком и Уиллардом Боуски.

Техника анимации, использованная в разработке Cuphead, аналогична мультипликации 1930-х годов [13]. Чад Молденхауэр, ранее работавший в графическом дизайне, рисовал анимацию и фоны вручную на бумаге, на что ушло 7 долгих лет. Он раскрашивал персонажей в Photoshop. Частота кадров является единственным отличием от первоначального процесса. Игровой процесс Cuphead имеет частоту кадров в 60 кадров в секунду, в отличие от 24 кадров в секунду мультипликации 30-х годов. Чад Молденхауэр рассказал, что видел свой процесс с присущими ему несовершенствами как реакцию на перфекционизм современного пиксельного искусства. Джаред Молденхауэр работал над другими аспектами игры, хотя они вместе обсуждали дизайн геймплея. Их студия наняла румынского разработчика, аниматора из Бруклина и джазового музыканта из Онтарио для проекта. Они стремились использовать процессы производства мультфильмов того временного периода, как если бы команда развивалась в ту эпоху [13].

Они планировали несколько различных уровней сложности и решили отказаться от типичного сюжета вида «дева в беде», выбрав тот, где Капхед постоянно создаёт проблемы самому себе. Разработчики планировали превзойти Мировой рекорд Гиннесса за количество битв с боссами в игре жанра run-and-gun, имея более 30 боссов, в то время как предыдущий рекорд был поставлен с 25.

## 1.3 Выбор среды разработки

### 1.3.1 Unity

Unity – это межплатформенная среда разработки трехмерных и двумерных компьютерных игр, разработанная американской компанией Unity Technologies. Она позволяет создавать приложения, работающие на более чем 25 различных платформах, включающих персональные компьютеры, игровые консоли, мобильные устройства, интернет-приложения и другие.

Основными преимуществами Unity являются наличие визуальной среды разработки с простым настраиваемым Drag&Drop интерфейсом, межплатформенной поддержкой и модульной системой компонентов. Поддерживает два сценарных языка: C#, JavaScript (модификация). А за расчёты физики отвечает физический движок PhysX от NVIDIA.

К недостаткам относят появление сложностей при работе с многокомпонентными схемами и затруднения при подключении внешних библиотек.

На Unity написаны тысячи игр, приложений, визуализации математических моделей, которые охватывают множество платформ и жанров. При этом Unity используется как крупными разработчиками, так и независимыми студиями.

Unity распространяется в основном бесплатно, с некоторыми незначительными ограничениями монетизации создаваемых игр. Помимо бесплатной, существуют четыре сборки - стандартная Unity, Unity iOS Pro, Android Pro и командная лицензия. Они отличаются стоимостью и функциональностью [14].

### 1.3.2 GameMaker: Studio

GameMaker: Studio – один из самых популярных игровых движков, позволяющий разрабатывать приложения под множество платформ. GameMaker: Studio является серьёзным развитием его предшественника — Game Maker и главным отличием является добавление кроссплатформенности. Благодаря ей, а также другим существенным доработкам, GameMaker: Studio стал мощным инструментом для профессиональной разработки.

Бесплатная версия (Standard) ограничена компиляцией под Windows. По сравнению с ней, Professional версия имеет множество преимуществ, включая управление ресурсами, компиляцию для macOS, Ubuntu и запуск на Android. Также, в профессиональной версии, можно покупать отдельные модули, расширяющие

функциональность программы. Версия Master Collection содержит все текущие модули и будущие дополнения [15].

### 1.3.3 Construct 3

Construct 3 – конструктор двумерных игр для Windows, разрабатываемый компанией Scirra позволяющий создавать 2D-игры различных жанров и сложности без навыков программирования. Игры, сделанные на нём, могут быть доступны на множестве платформ.

Интерфейс программы имеет визуальный (WYSIWYG) редактор, что дает возможность создать игру без навыков программирования. В редакторе имеются «события» и «действия», создающие логику игр.

Редактор позиционирует себя как «подходящий для людей с различным уровнем опыта программирования». Доступна функция создания прототипа игры, демоверсий, презентаций, обучающих программ [16].

Construct 3 – платный конструктор игр. Цена за персональный вариант (с возможностью ограниченного коммерческого использования) равна 129,99$ (6399 рублей в Steam), а за бизнес-версию (без ограничений коммерческого характера) — 429,99$ (20999 руб. в Steam). Также программа имеет базовую бесплатную версию, имеющую ограничение в 100 событий, 4 слоя, 2 эффекта, некоторые платформы для экспорта и невозможность использования в коммерческих целях. Имеется также специальная образовательная лицензия, которая стоит 159,99$ в год ($45,99 в квартал или $17,99 в месяц), для школ и прочих учебных заведений [16].

### 1.3.4 Формирование критериев сравнения

Сформируем сравнительную таблицу, по которой определим наиболее оптимальную среду разработки среди рассмотренных.

Основные критерии приведены в таблице 1.1. Шкала оценки приведена в таблице 1.2.

Таблица 1.1 – Критерии оценки сред

| Критерий | Степень важности |
|---|---|
| 1 Цена | 0,20 |
| 2 Сложность в освоении | 0,15 |
| 3 Кроссплатформенность | 0,10 |
| 4 Функционал для разработки 2D | 0,15 |
| 5 Количество подключаемых библиотек | 0,15 |
| 6 Количество готовых ассетов для разработки | 0,15 |
| 7 Количество официальных и неофициальных руководств | 0,10 |

Таблица 1.2 – Шкала оценки сред

| Баллы | Значение |
|---|---|
| 1 | Очень плохо |
| 2 | Плохо |
| 3 | Нормально |
| 4 | Хорошо |
| 5 | Отлично |

### 1.3.5 Сравнение и выбор среды разработки

Опираясь на информацию, находящуюся в свободном доступе на официальных сайтах различных сред разработки, был проведен анализ, с целью определения оптимальной среды. Результат анализа приведен в таблице 1.3.

Таблица 1.3 – Функциональное сравнение с аналогами

| Критерий | Unity | GameMaker : Studio | Construct 3 |
|---|---|---|---|
| 1 Цена | 5 | 4 | 2 |
| 2 Сложность в освоении | 4 | 4 | 5 |
| 3 Кроссплатформенность | 5 | 3 | 4 |
| 4 Функционал для разработки 2D | 5 | 5 | 5 |
| 5 Количество подключаемых библиотек | 5 | 3 | 3 |
| 6 Количество готовых ассетов для разработки | 4 | 4 | 3 |
| 7 Количество официальных и неофициальных руководств | 5 | 4 | 3 |
| Суммарная оценка | 4,7 | 3,9 | 3,5 |

По результатам проведенного анализа видно, что Unity имеет преимущество перед остальными рассматриваемыми средами разработки игр.

# 2 ПРОЕКТИРОВАНИЕ

## 2.1 Определение функциональных требований

У рассмотренных в подразделе 1.2 аналогов можно выделить некоторые черты, такие как:

– использование элементов метроидвании. Различные ограничения исследования карты в зависимости от имеющихся способностей, предметов или игровых достижений;

– случайная генерация уровня. Каждая новая попытка сопровождается новой, только что сгенерированной картой;

– сложность геймплея;

– интересная боевая система. Разнообразие врагов, боссов и вооружения, с помощью которых можно противостоять встречающимся на пути преградам;

– постоянное развитие персонажа. Открытие новых способностей, предметов и вооружения, их улучшение;

– единственная жизнь. В случае гибели игрок начинает уровень с самого начала с сохранением некоторого прогресса в виде боевого опыта, внутриигрового опыта, внутриигровой валюты или найденных предметов.

Поскольку все из рассмотренных аналогов являются успешными представителями своего жанра, можно сделать вывод, что выделенные черты в правильной комбинации положительно влияют на игру.

Таким образом можно определить функциональные требования к создаваемой игре:

1) система перемещения по игровой карте, которая включает в себя:

а) бег;

б) прыжки;

в) множественные прыжки в воздухе;

г) перекаты;

2) боевая система для игрока, включающая в себя:

а) атаку;

б) блокирование ударов со стороны, в которую направлен персонаж;

в) неуязвимость во время перекатов;

г) неуязвимость на короткое время после получения урона;

3) враги ближнего и дальнего боя со своим искусственным интеллектом (далее ИИ), который включает в себя:

а) систему передвижения с возможностью преодоления препятствий с помощью прыжков;

б) патрулирование по маршрутным точкам;

в) поле зрения, в котором враг видит игрока;

4) боевая система для врагов, включающая в себя:

а) атаку;

б) оглушения при получении урона;

в) защиту от оглушения при получении определенного количества оглушений;

5) единственная жизнь. После гибели игрок начинает с самого начала;

6) открытие сундуков, разбросанных по игровой карте, и получение с них различных ресурсов;

7) диалог с неигровыми персонажами;

8) развитие главного персонажа путем улучшения его характеристик;

9) сохранение прогресса.

## 2.2 Логика основных механик

Цикл игры – это набор фундаментальных игровых механик, которые повторяются на протяжении всего игрового процесса и определяют базовый игровой опыт. По своей сути цикл игры является ее ядром, которое можно развивать и дополнять второстепенными механиками. А также менять с помощью добавления новых элементов по ходу внутриигрового развития для

внесения разнообразия и ощущения новизны в игровой процесс. Именно это затягивает пользователя и является причиной, почему он снова возвращается в игру.

Основными требованиями к циклу игры является его простота, понятность и однозначность. Он не должен быть перегружен множеством различных действий, которые могут запутать и, как следствие, отпугнуть игрока [17].

Цикл игры используется практически всеми разработчиками игр и выполняет следующие цели:

1) определяет возможности пользователя во время игры;

2) определяет примерный объем работ для создания проекта;

3) наглядно демонстрирует суть программного продукта и обеспечивает его единое видение;

4) фиксирует ключевые аспекты проекта.

Для создаваемого проекта также был разработан цикл игры, изображенный на рисунке 2.1.

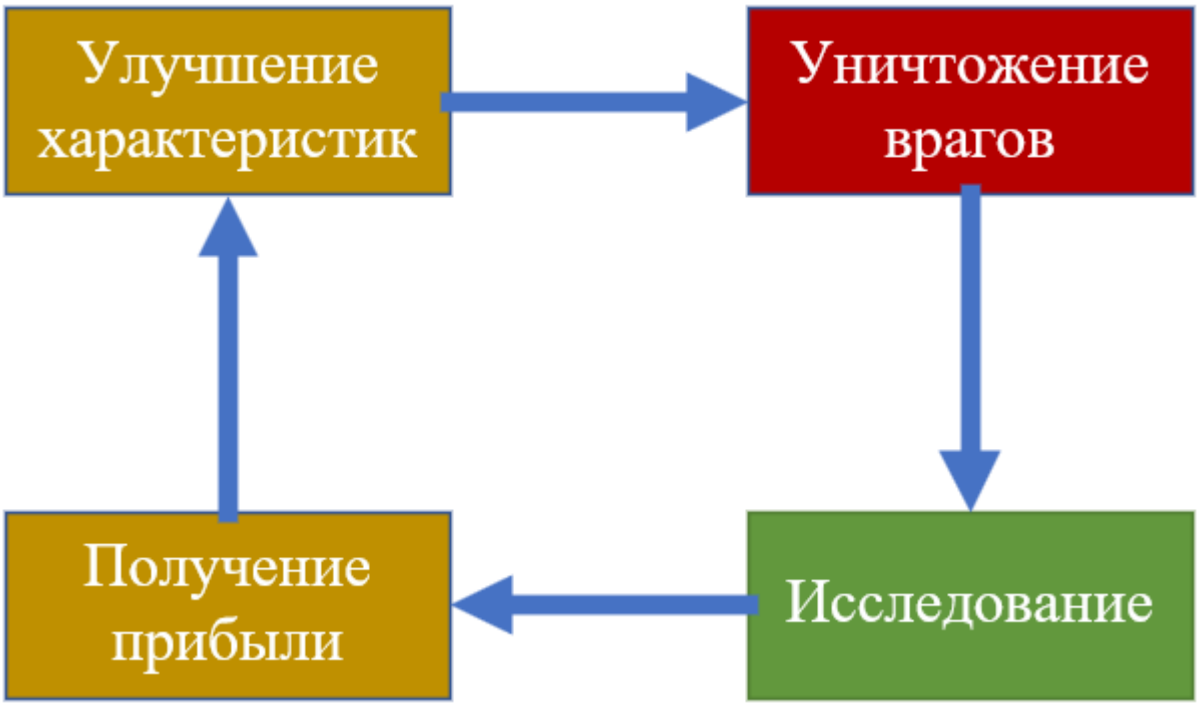


Рисунок 2.1 – Разработанный цикл игры

Помимо цикла игры были составлены диаграммы деятельности для основных и самых сложных с логической точки зрения механик, которые описывают, как они должны работать.

Одной из основных механик является система боя, имеющая две стороны – сторона игрока и сторона ИИ врага. В упрощенном виде логика боевой системы игрока представлена на рисунке 2.2.

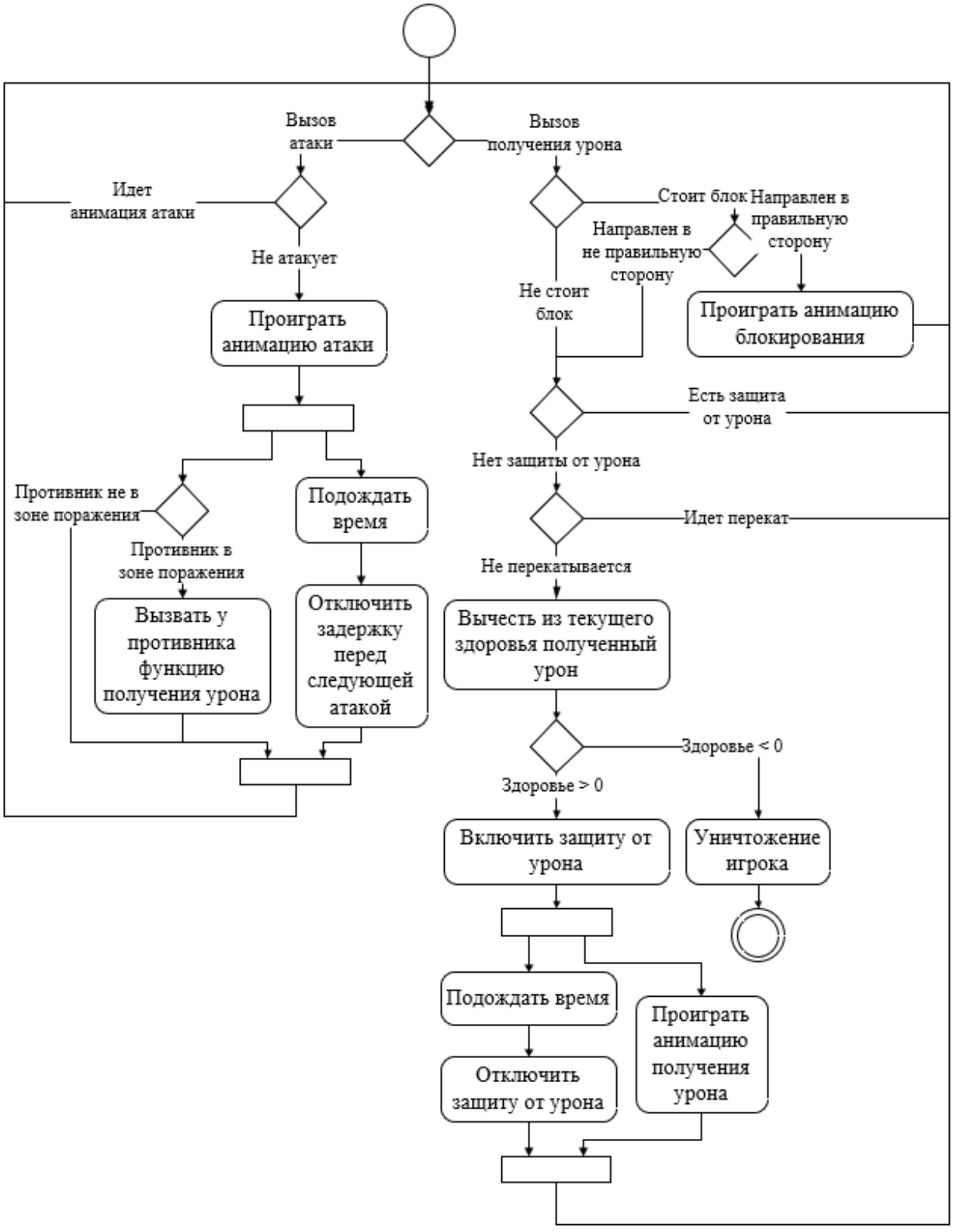


Рисунок 2.2 – Боевая система игрока

В общем виде боевая система противников ближнего и дальнего боя представлена на рисунке 2.3.

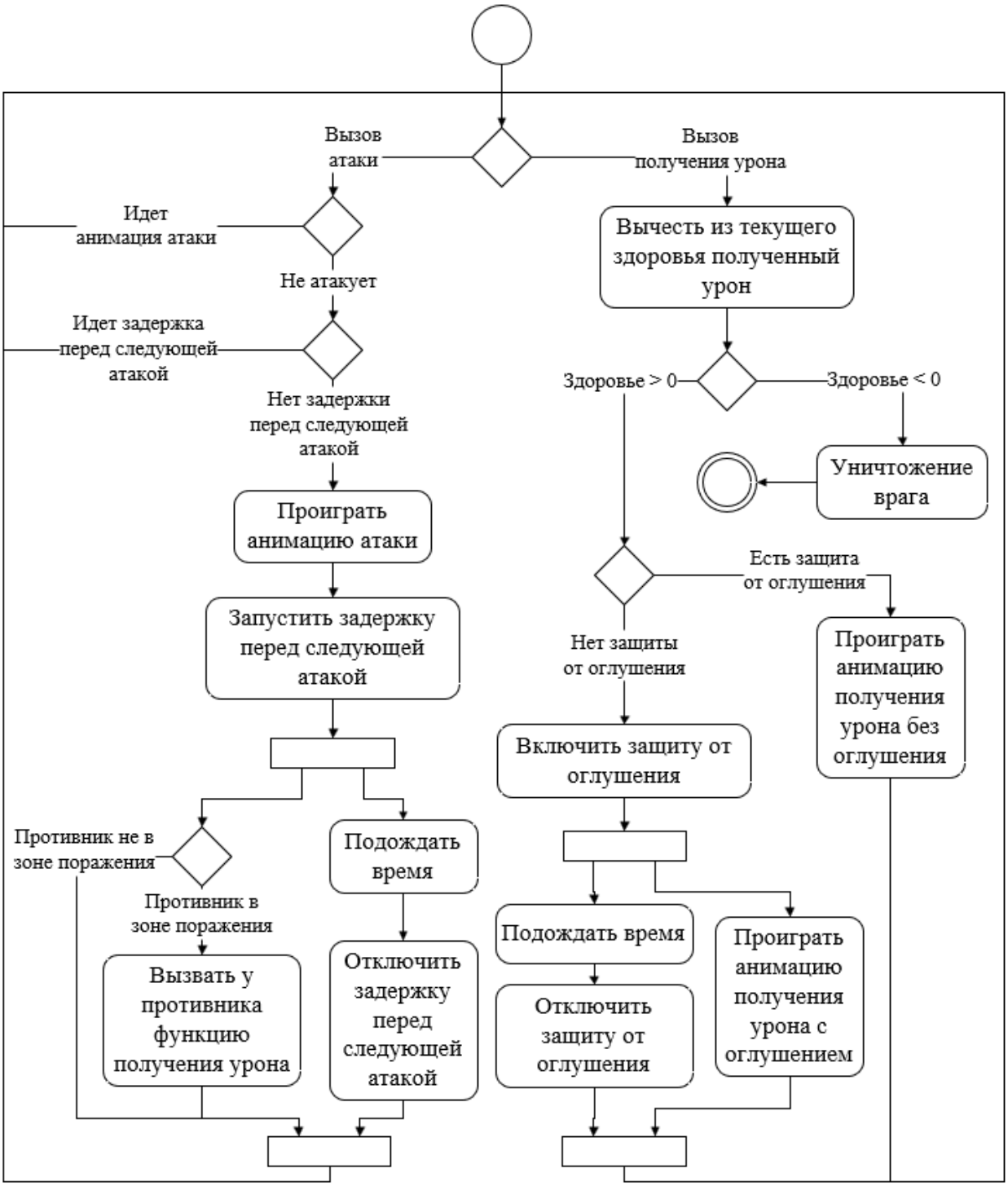


Рисунок 2.3 – Общий вид боевой системы противников

Помимо боевой системы у противников также должен присутствовать ИИ, который будет управлять ими. В общем виде ИИ противников ближнего и дальнего боя представлен на рисунке 2.4.

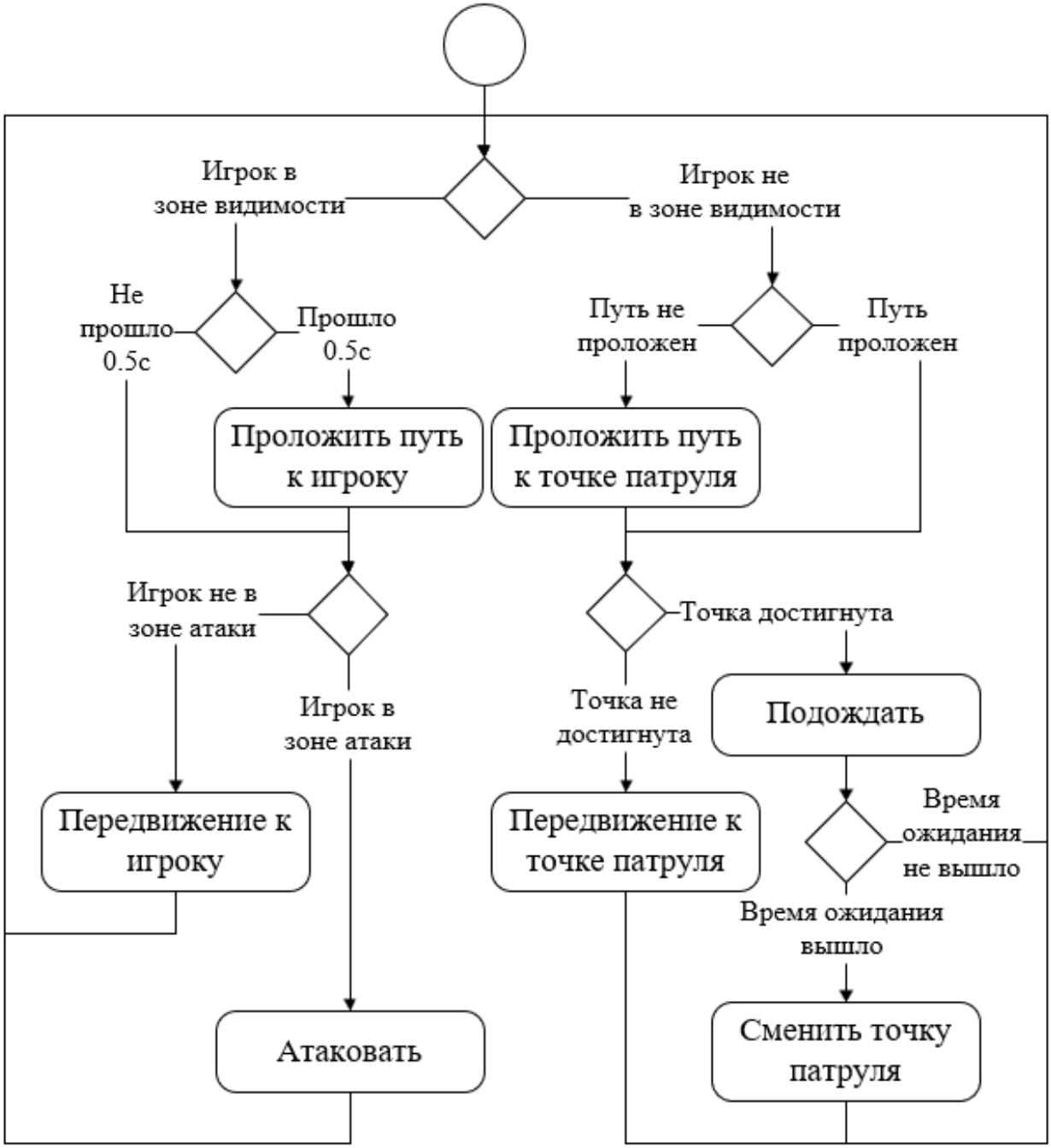


Рисунок 2.4 – Общий вид ИИ противников

## 2.3 Архитектура классов

Для проектирования создаваемой игры использовалась схема классов UML. Разработанная схема включает в себя около 35 различных классов, которые были распределены на логические группы. Некоторые из созданных классов описаны далее.

Для управления камерой и задним фоном с эффектом параллакса были разработаны классы, изображенные на рисунке 2.5.

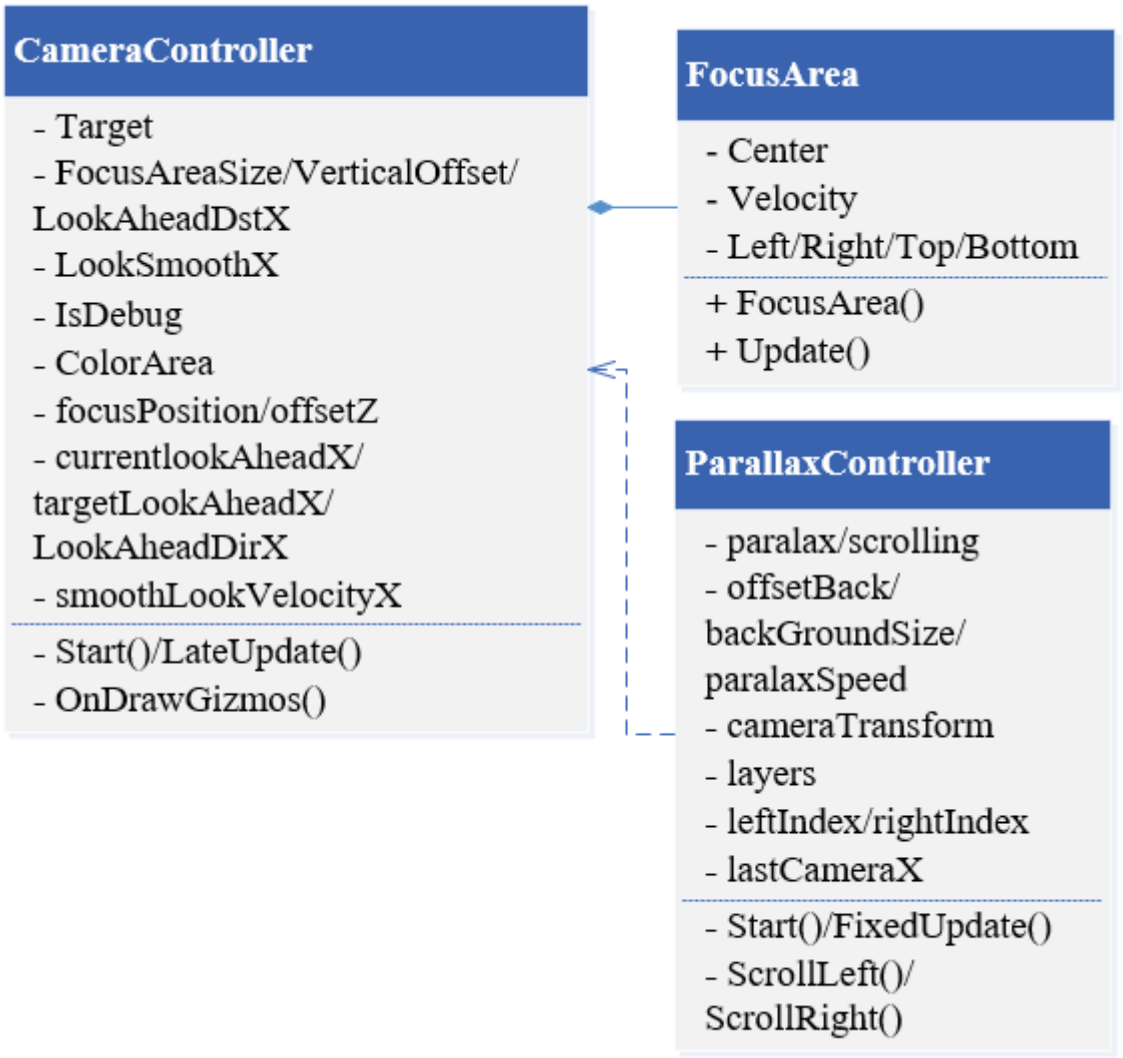


Рисунок 2.5 – Классы камеры и заднего фона

За управление камерой отвечает скрипт «CameraController», реализующий:

– функционал области, которая описана в классе «FocusArea». В данном пространстве персонаж может передвигаться без передвижения камеры;

– перемещение камеры со смещением в сторону движения персонажа;

– перемещение камеры вверх и вниз в зависимости от положения персонажа внутри определенной области и координате Y.

Эффект параллакса создается скриптом «ParallaxEffect». Он передвигает все слои фона в зависимости от положения камеры и при необходимости переставляет спрайты в сторону движения персонажа для создания видимости бесконечного фона.

Некоторые параметры игрового персонажа и врагов описываются специальными классами «CharacterStat» и «StatModifier», которые изображены на рисунке 2.6.

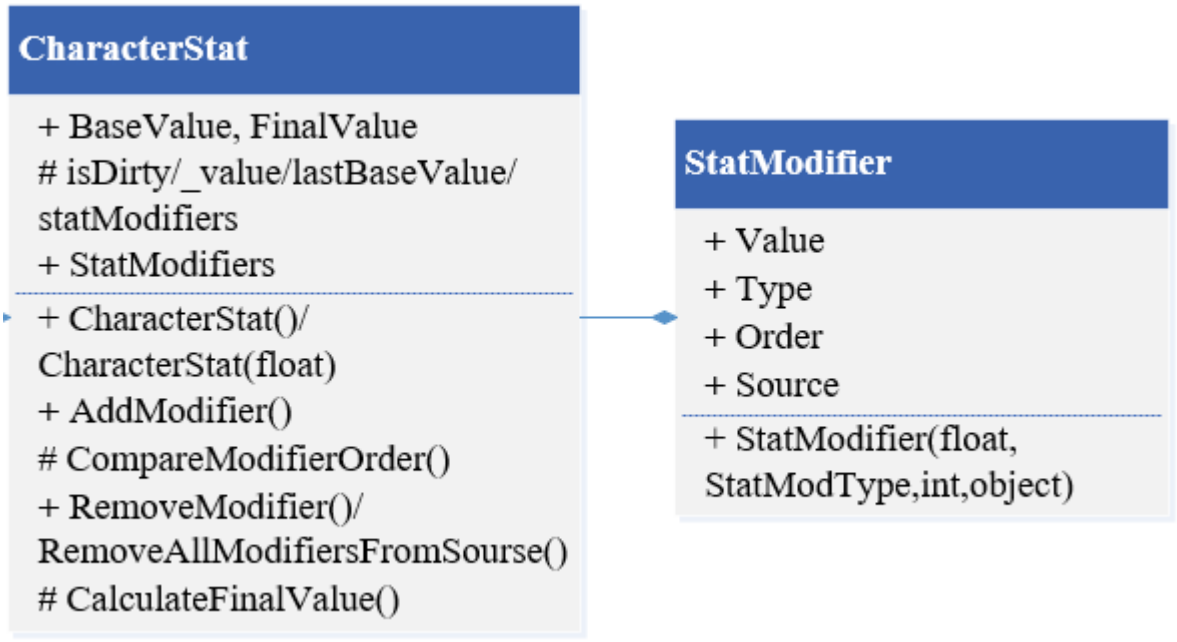


Рисунок 2.6 – Классы настраиваемых параметров

Классы «CharacterStat» и «StatModifier», отвечают за модификацию характеристик персонажей во время игры, будь то повышение уровня для

игрового персонажа, противника или получение каких-либо улучшений во время прохождения уровня. Иначе говоря, данные классы должны описывать как изначально заданное значение изменяется при добавлении на него различных модификаторов.

Класс «CharacterStat» содержит методы, позволяющие:

– добавлять модификатор одного из трех типов (простое значение, процентное значение, увеличение значения в несколько раз) на начальное значение параметра;

– удалять конкретный модификатор параметра;

– удалять все модификаторы параметра от определенного источника.

Для системы уровней различных модифицируемых характеристик был спроектирован класс «StatLevelSystem», который изображен на рисунке 2.7.

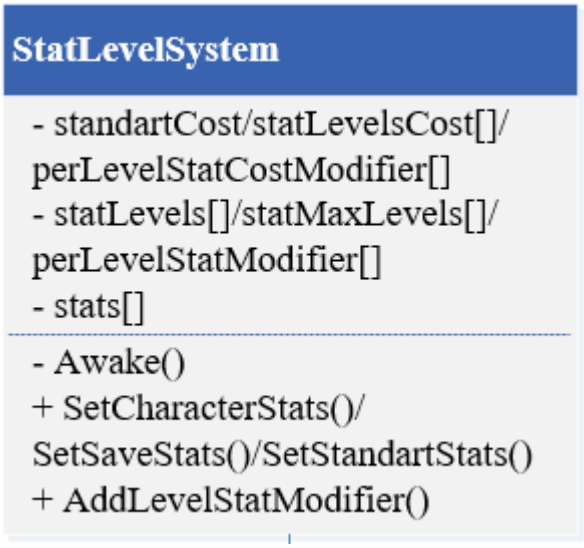


Рисунок 2.7 – Класс системы уровней модифицируемых характеристик

Данный класс описывает, какие модификаторы накладываются на значение характеристики при увеличении уровня этой характеристики.

Для разрушаемых объектов, главного персонажа и платформ были спроектированы классы, изображенные на рисунке 2.8

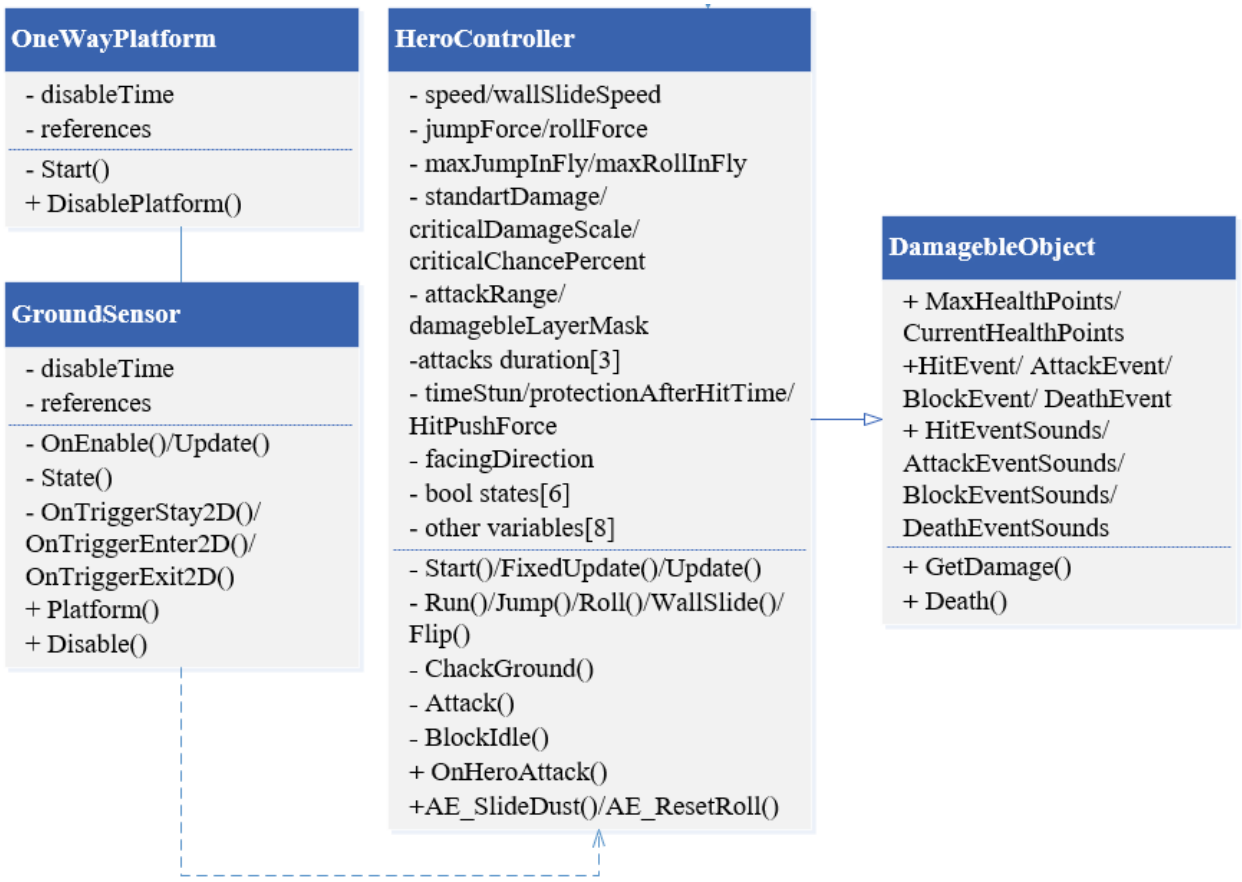


Рисунок 2.8 – Классы разрушаемых объектов, игрока, и платформы

Класс, который наследуют все объекты, с возможностью получать урон и уничтожаться называется «DamagebleObjects». В нем присутствуют методы получения урона и уничтожения объекта, с возможностью их перезаписи, для более гибкой настройки, а также общие для данного типа объектов параметры:

– максимальное здоровье;

– текущее здоровье;

– события Unity, которые могут вызываться дочерними классами, когда это необходимо;

– наборы звуков для конкретных событий.

За функционал игрового персонажа отвечает класс «HeroController», имеющий множество настраиваемых полей для возможности корректировки его статичных характеристик (например, модификатор силы прыжка или кувырка, скорость скольжения по стене, сила отбрасывания при получении урона и т.д.). Помимо этого, данный класс описывает такие функции как:

– бег;
– прыжки;
– спуск с платформы;
– перекаты;
– скольжение по стене;
– поворот объекта;
– атака;
– блокирование.

Класс «GroundSensor» служит для проверки, находится ли игрок в воздухе или же стоит на земле или платформе. Также, если игрок стоит на платформе он передает ссылку на класс платформы «OneWayPlatform», для вызова метода ее отключения, если игрок нажмет на соответствующую кнопку.

В игре также должны присутствовать враги как ближнего, так и дальнего боя. На рисунке 2.9 изображены классы врагов их поля зрения и снарядов.

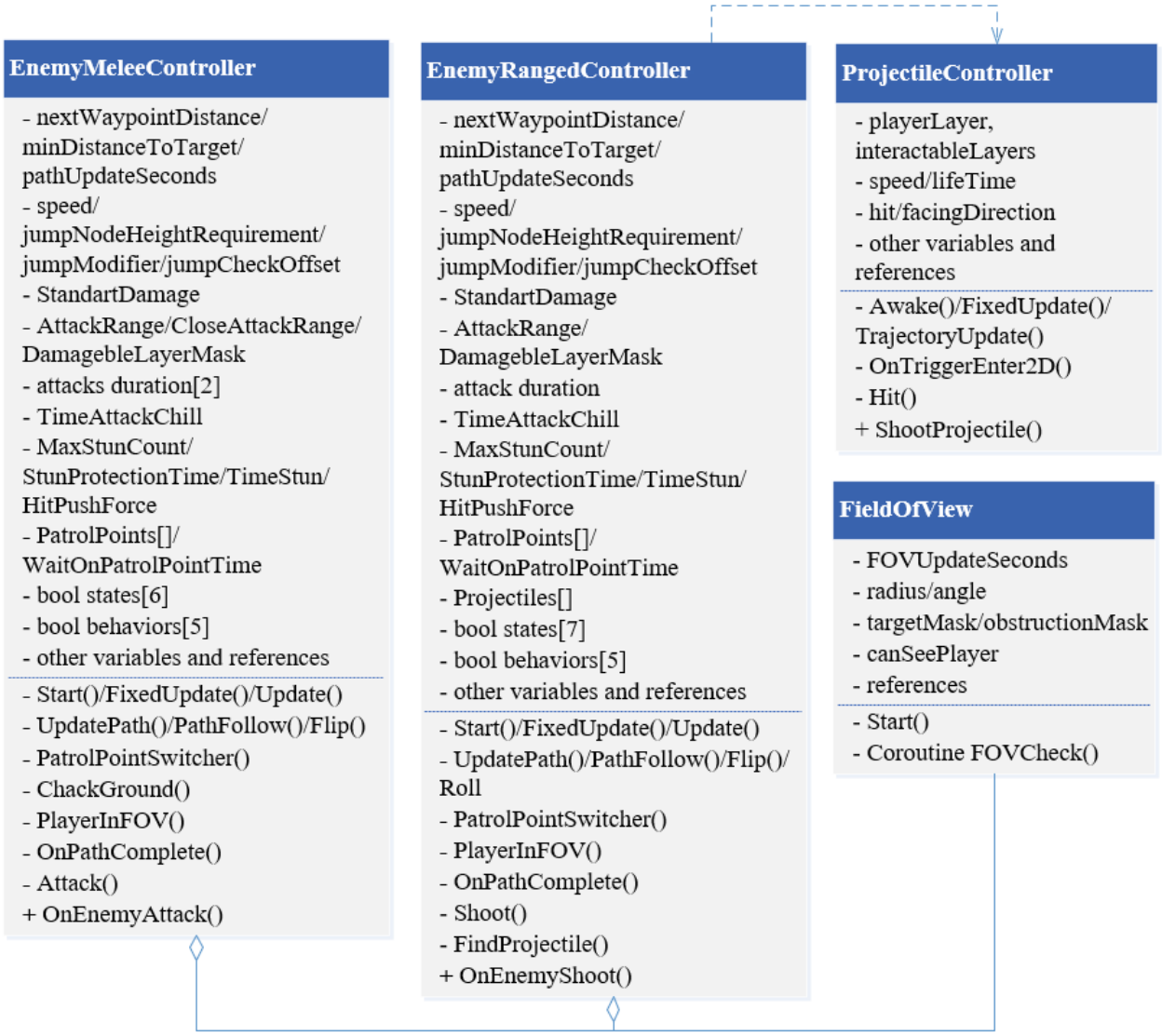


Рисунок 2.9 – Классы врагов, поля зрения и снарядов

Классы «EnemyMeleeController» и «EnemyRangedController» имеют гибкие настройки параметров, дающие возможность использовать данные классы для всех врагов данного типа. Также, данные классы, как и класс игрового персонажа, изображенный на рисунке 2.8, наследуют класс разрушаемых объектов «DamagebleObjects» и используют классы «GroundSensor» и модифицируемых параметров. Помимо этого, у врагов присутствует класс «FieldOfView», который позволяет им фиксировать игрока, когда тот заходит в их поле зрения и между врагом и игроком нет препятствий, мешающих обзору.

Основные функции, которые описывают классы врагов:

– поиск пути до цели;

– патрулирование между точками, расставленными на карте;

– преследование игрока по созданному пути с преодолением возможных препятствий;

– атака, если игрок находится в зоне поражения;

– возврат к зоне патрулирования, если игрок покинул поле зрения противника и не появлялся там определенное время.

Класс «ProjectileController» описывает поведение снарядов врага дальнего боя, включая их физику.

Для реализации возможности взаимодействия с неигровыми персонажами и объектами на карте были спроектированы классы, изображенные на рисунке 2.10.

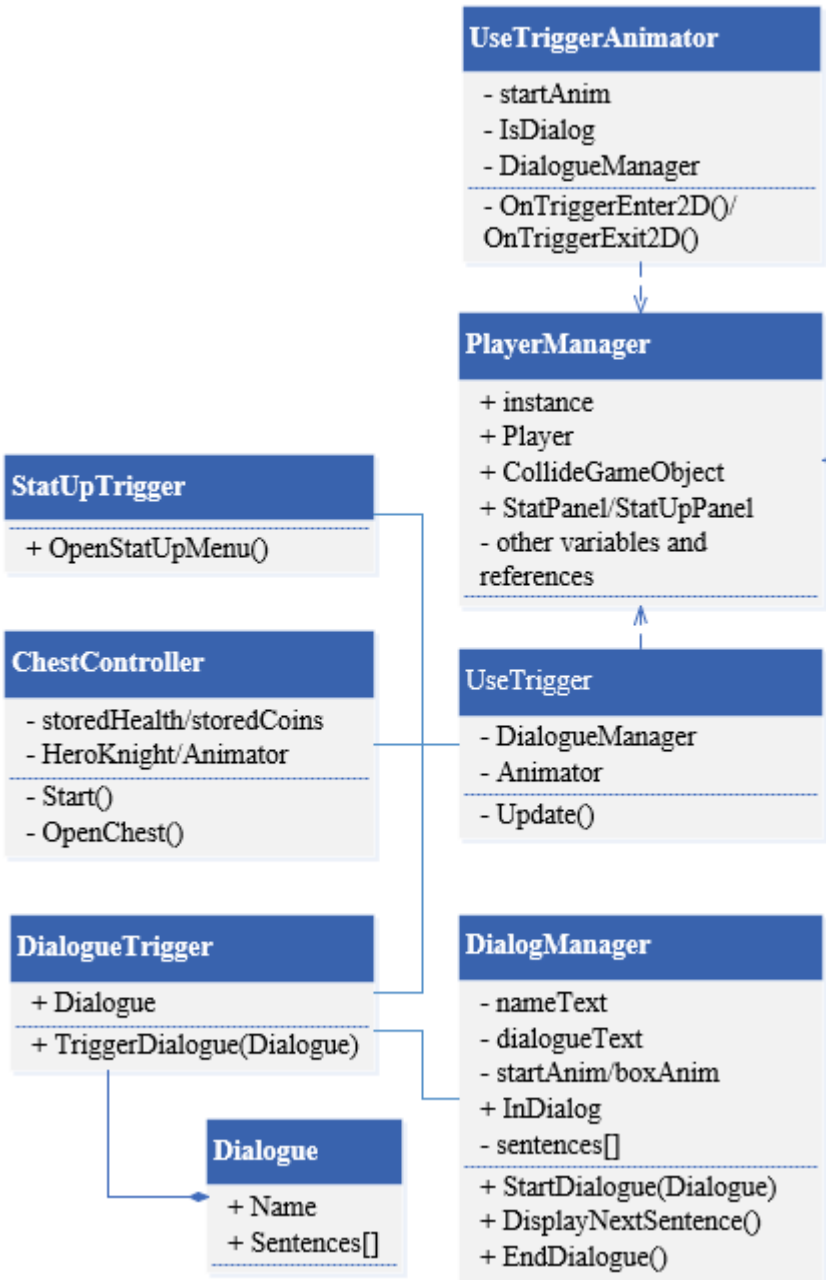


Рисунок 2.10 – Классы взаимодействия с объектами и классы самих объектов

Класс «UseTriggerAnimator» служит для определения факта вхождения игрового персонажа в поле объекта, с которым можно взаимодействовать и передает ссылку на этот объект в класс «PlayerManager», чтобы другие классы могли получить к нему доступ.

При нажатии на кнопку, в зависимости от объекта, рядом с которым находится неигровой персонаж, будет выполнено определенное действие:

– если игрок находится радом с неигровым персонажем, с которым можно поговорить, будет вызван метод «TriggerDialogue» класса

«DialogueTrigger» принадлежащий объекту, с которым было произведено взаимодействие;

– если игрок находится радом с неигровым персонажем, который должен улучшать характеристики, будет вызван метод «OpenStatUpMenu» класса «StatUpTrigger» принадлежащий объекту, с которым было произведено взаимодействие;

– если игрок находится радом с сундуком, будет вызван метод «OpenChest» класса «ChestController» принадлежащий объекту, с которым было произведено взаимодействие.

Класс «Dialogue» описывает сущность диалога, в которой содержатся имя говорящего и все предложения.

Класс «DialogManager» отвечает за управление окном диалога и имеет следующие функции:

– начало диалога, с заполнением заранее созданного окна диалога именем собеседника и текстом его первого предложения;

– отображением следующего по списку предложения при нажатии клавиши;

– окончание диалога если все предложения были отображены или если игровой персонаж покинул зону диалога.

Для сохранения игрового прогресса также были разработаны классы, изображенные на рисунке 2.11.

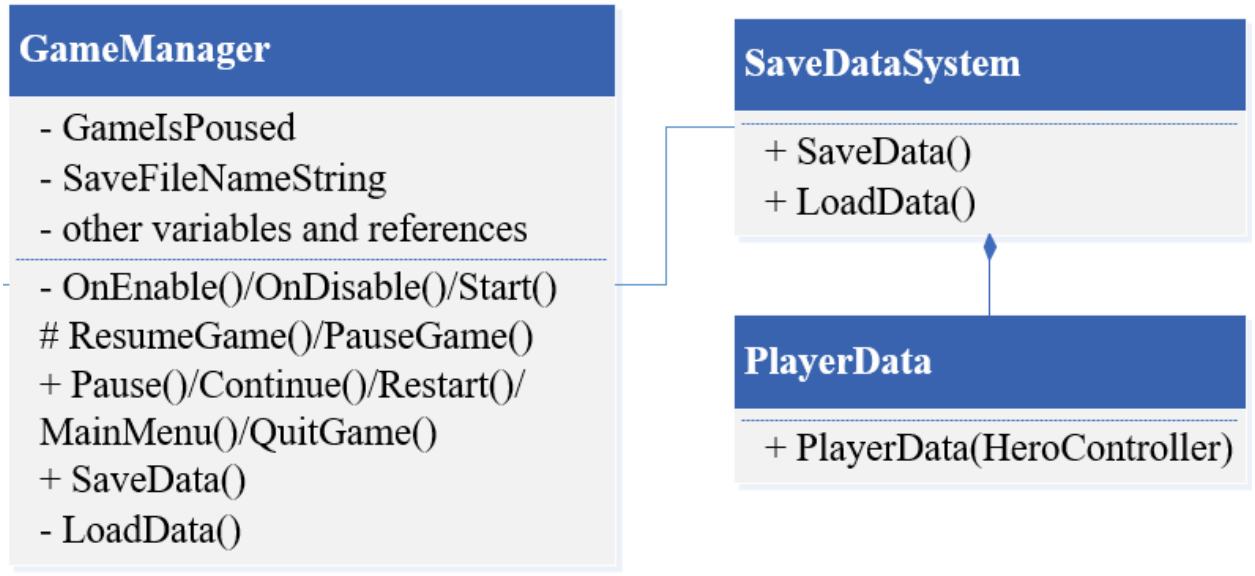


Рисунок 2.11 – Классы сохранения игры и игрового менеджера

Класс «SaveDataSystem» служит для реализации функций сохранения игры. «PlayerData» – это модель данных для сохранения игры в бинарный файл на локальном носителе.

Класс «GameManager» описывает следующий функционал:

– остановка игрового времени, когда включена пауза;

– возобновление игрового времени при выключении паузы;

– перезагрузка сцены при гибели игрового персонажа;

– выход на сцену главного меню игры;

– полных выход из приложения;

– сохранение игрового прогресса;

– загрузка прогресса из локального носителя в игру.

Данный класс должен загружаться первым при запуске сцены, для выполнения загрузки сохранения из файла.

Для создаваемой игры были спроектированы классы игрового меню, которые должны вызываться во время игрового процесса. Данные классы изображены на рисунке 2.12.

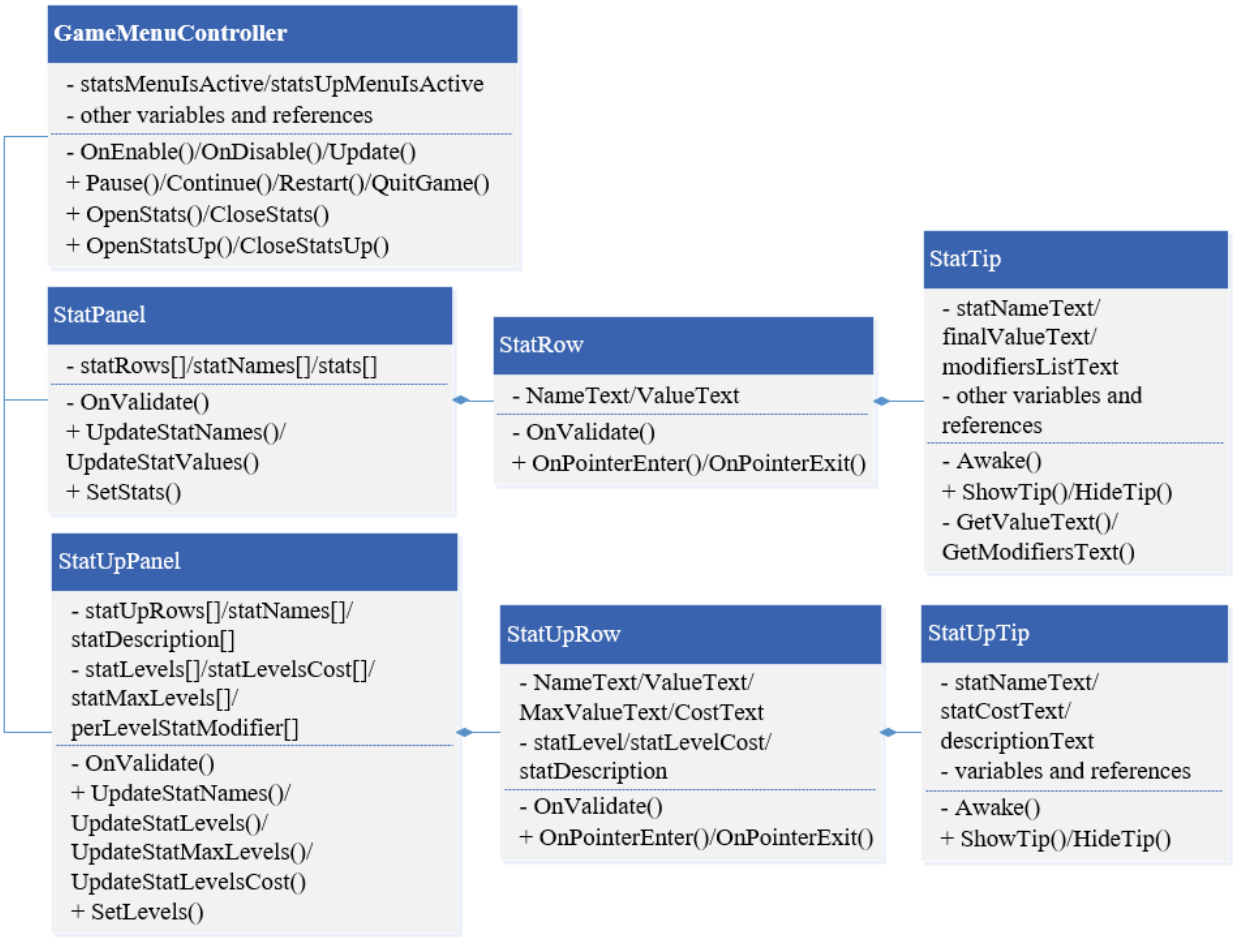


Рисунок 2.12 – Классы внутриигрового меню

Класс «GameMenuController» вызывает различные меню, когда это необходимо, будь то меню паузы, меню параметров игрового персонажа или меню повышения уровня параметров.

Остальные изображенные на рисунке 2.12 классы спроектированы для системы отображения параметров в окнах их просмотра или улучшения. Они построены по схожей системе, но передают в себе отличающиеся данные и выполняют различные действия с ними.

Классы «StatPanel», «StatRow», «StatTip» служат для передачи самих параметров и их модификаторов в отведенные для этого окна, а также преобразования их в удобочитаемый вид.

Классы «StatUpPanel», «StatUpRow», «StatUpTip» служат для передачи текущих и максимальных уровней параметров, а также стоимости их улучшений.

Помимо внутриигрового меню необходимо главное меню, которое появляется первым при запуске программы. На рисунке 2.13 изображены классы главного меню игры.

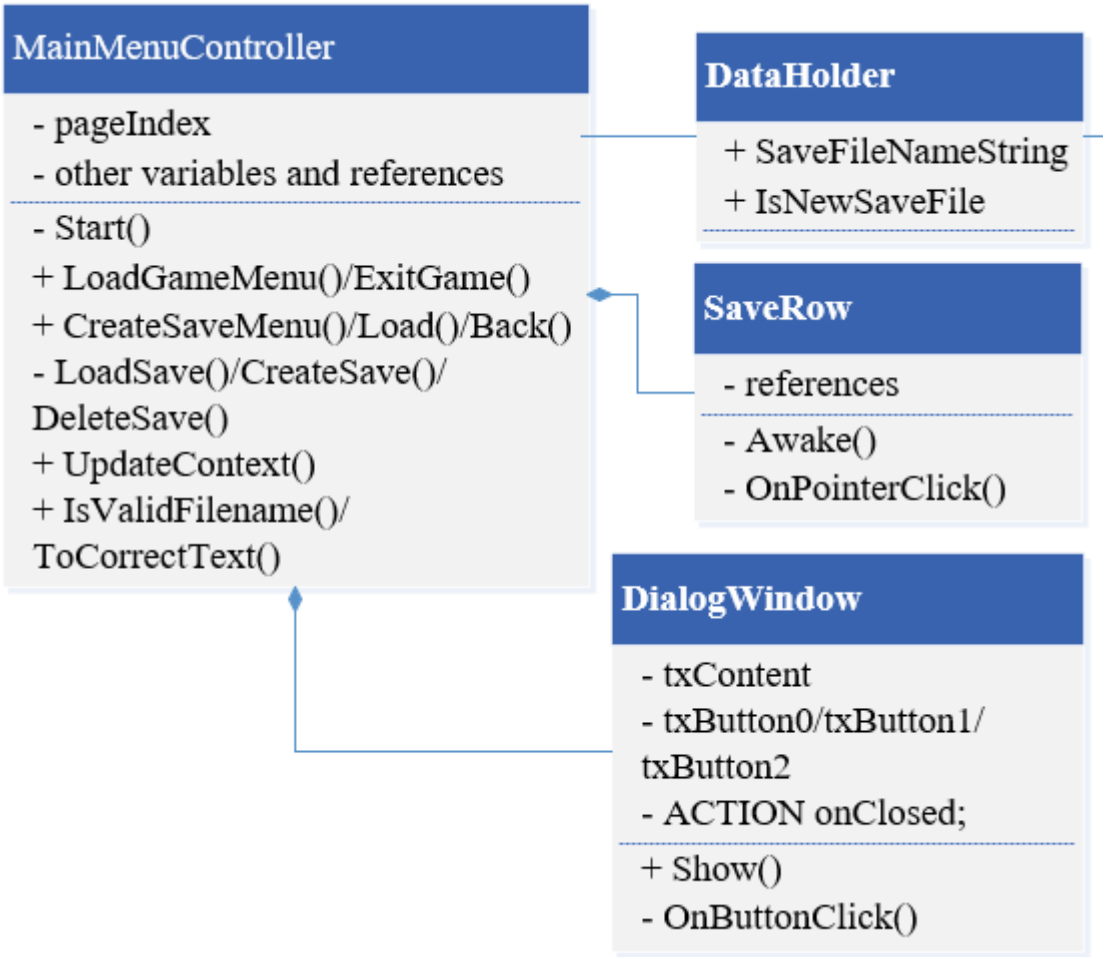


Рисунок 2.13 – Классы главного меню игры

Данные классы в общем выполняют следующие функции:

– переключение между окнами меню;

– поиск сохранений на локальном носителе;

– передача названия файла сохранения в простой класс «dataholder» откуда оно будет взято игровым менеджером «GameManager» упомянутом ранее, для загрузки данных;

– создание новых сохранений;

– валидацию названия сохранения при его создании;

– загрузку игровой сцены.

# 3 РАЗРАБОТКА

## 3.1 Основа игры

Разработка компьютерных игр во многом схожа с разработкой обычных приложений. Использование схожих шаблонов программирования или архитектур это стандартная практика. Однако в отличие от других программ, которые просто простаивают большую часть времени, ожидая каких-либо действий со стороны пользователя, игры должны продолжать работать даже когда пользователь не предоставляет никакого ввода. Таким образом даже если пользователь просто сидит и смотрит в экран своего устройства игра продолжает свою работу, персонажи передвигаются, а эффекты и анимации проигрываются. Ключевое отличие игры от стандартных приложений как раз и заключено во вводе данных. В играх ввод обрабатывается, но не ожидается, а цикл продолжает работать вплоть до момента выключения игры [18]. На рисунке 3.1 изображен стандартный игровой цикл.

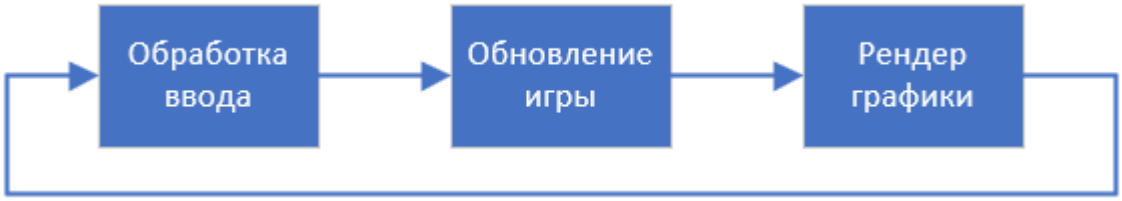


Рисунок 3.1 – Игровой цикл

В игровом цикле присутствуют три основных этапа:

1) обработка пользовательского ввода без блокировки;

2) обновление состояния игры;

3) рендер изображения.

Помимо всего вышесказанного игровой цикл должен отвечать еще и за ход внутриигрового времени, а также управлять скоростью игрового процесса. Поскольку современные системы и их мощности сильно отличаются друг от

друга, это просто необходимо, иначе игры работали бы совершенно с разной скоростью на разных устройствах [18].

Используемая среда разработки Unity уже содержит в себе описание игрового цикла со всеми необходимыми инструментами взаимодействия [19].

## 3.2 Игровой персонаж

Игровой персонаж – это игровой объект, которым может управлять пользователь и через него взаимодействовать с внутриигровым миром.

Данный объект содержит несколько дочерних объектов, изображенных на рисунке 3.2 вместе с графической демонстрацией персонажа.

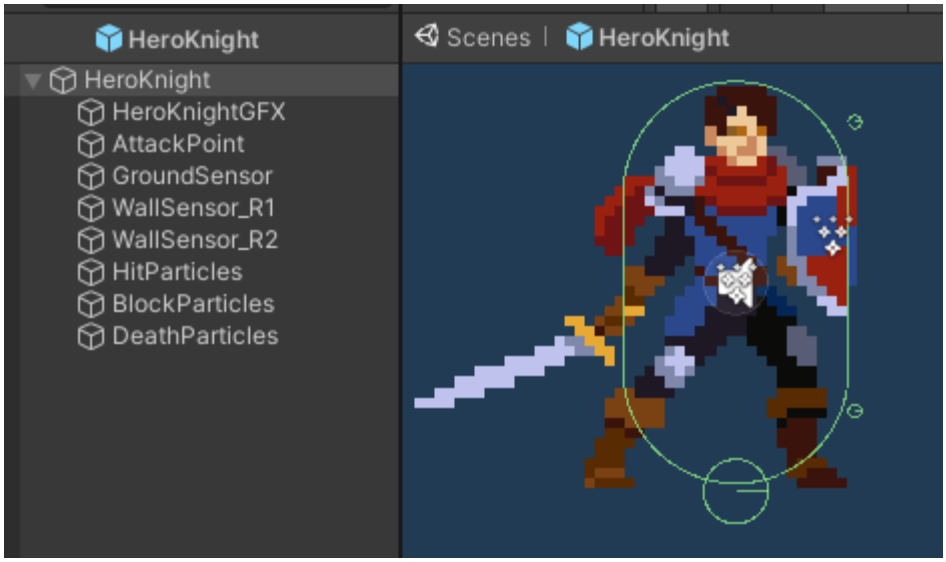


Рисунок 3.2 – Объект игрового персонажа и его структура

В структуру объекта игрового персонажа входят следующие элементы:

1) главный объект «HeroKnight» – отвечает за всю основную логику персонажа, в нем находятся следующие компоненты:

а) скрипт «HeroController» – описывающий как объект реагирует на ввод пользователя, а также боевую систему;

б) скрипт «StatLevelSystem» – реализующий логику повышения уровней характеристик;

в) «CapsuleCollider2D» – для обработки коллизий с другими объектами;

г) «RigidBody2D» – для обработки игровой физики;

д) «AudioSource» – для воспроизведения звуков.

2) «HeroKnightGFX» – отвечает за графическое представление объекта на сцене и содержит в себе следующие компоненты:

а) «SpriteRenderer» – отвечающий за рендер спрайта;

б) «Animator» – отвечающий за анимацию;

в) скрипт «ColorSwap» – для замены цветов на спрайте.

3) «AttackPoint» – служит центром области, в которой врагам будет наноситься удар;

4) «GroundSensor» – объект, в котором расположен скрипт «GroundSensor». С помощью данного скрипта определяется, стоит ли игрок на какой-либо поверхности или находится в воздухе;

5) «WallSensor» – это объекты похожие на «GroundSensor», определяющие находится ли игрок у стены или нет;

6) «HirParticles», «BlockParticles», «DeathParticles» – это объекты с компонентом «Particle System», они служат для графической демонстрации ранения, блокирования удара и гибели соответственно.

### 3.3 Неигровой персонаж

Неигровой персонаж – в данной игре, это персонаж, с которым можно взаимодействовать. Существует два типа неигровых персонажей – с первым можно провести диалог, а второй вызывает окно улучшения характеристик. Данный объект с его структурой изображен на рисунке 3.3.

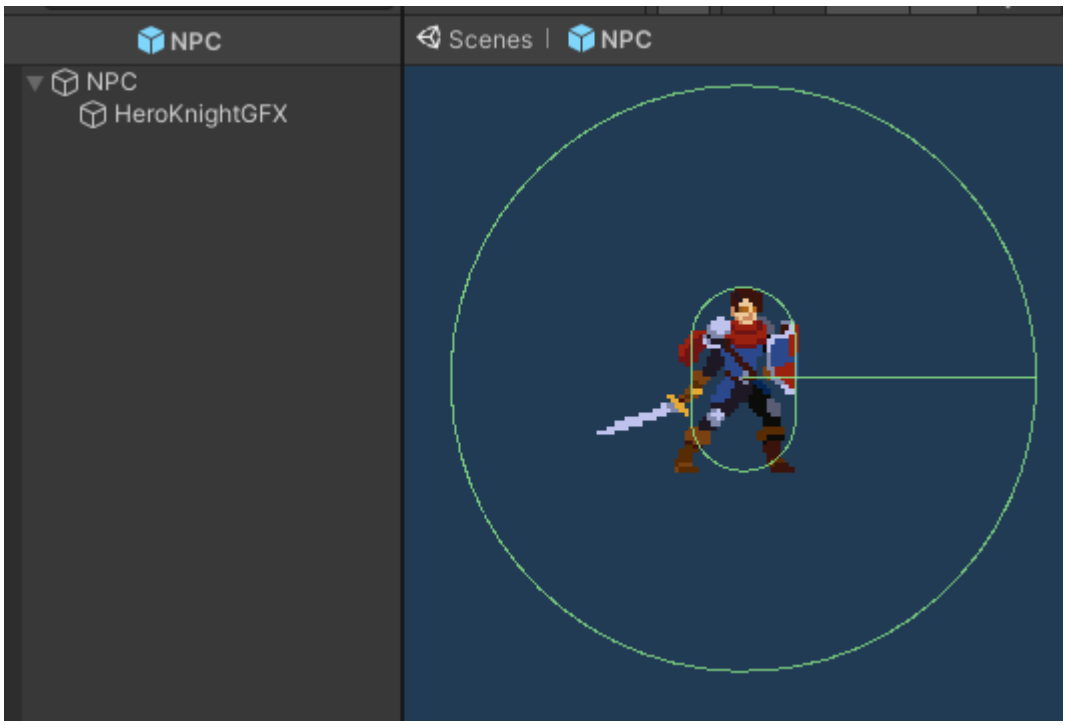


Рисунок 3.3 – Объект неигрового персонажа и его структура

В структуру объекта неигрового персонажа входят следующие элементы:

1) главный объект «HeroKnight» – отвечает за всю основную логику персонажа, в нем находятся следующие компоненты:

а) скрипт «DialogueTrigger» – в нем содержатся предложения, которые будет выводить на экран менеджер диалогов;

б) скрипт «UseAnimator» – для фиксирования вхождения в зону взаимодействия;

в) «CircleCollider2D» (в режиме триггера) – для обработки коллизий с другими объектами;

2) «HeroKnightGFX» – отвечает за графическое представление объекта на сцене и содержит в себе следующие компоненты:

а) «SpriteRenderer» – отвечающий за рендер спрайт;

б) «Animator» – отвечающий за анимацию;

в) Скрипт «ColorSwap» – для замены цветов на спрайте;

г) «CapsuleCollider2D» – для обработки коллизий с другими объектами.

## 3.4 Поиск пути

Поиск пути противников осуществляется с использованием бесплатной библиотеки «A* Pathfinding Project», используя алгоритм поиска пути A*.

Алгоритм A* – это мощный алгоритм в сфере ИИ с широким спектром применения. Он является самым популярным способом для нахождения кратчайшего пути из-за очень гибкой системы реализации. Сегодня этот алгоритм применяется в самых различных областях, в том числе и в разработке игр.

На рисунке 3.4 изображен граф, построенный с помощью компонента «PathFinder» из библиотеки «A* Pathfinding Project», а также настройки этого компонента.

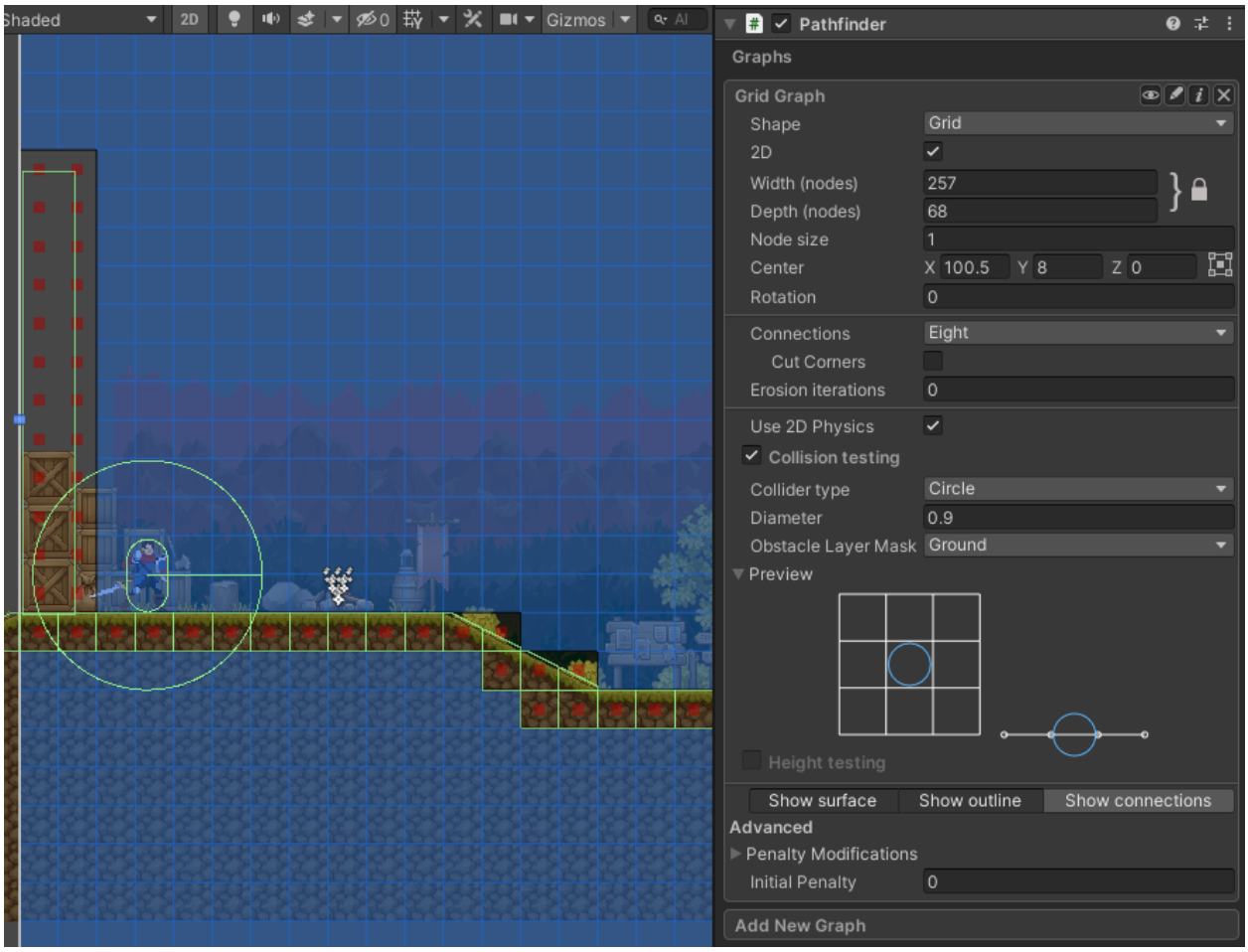


Рисунок 3.4 – Компонент «PathFinder» и построенный граф

Граф необходимо построить только один раз. В дальнейшем, во время игрового цикла, с помощью компонента «Seeker» можно будет управлять построением пути до цели из другого скрипта. С помощью этого способа и был реализован ИИ противников.

### 3.5 Противники

В игре присутствует три типа противников, два ближнего и один дальнего боя. Все противники изображены на рисунке 3.5.

Рисунок 3.5 – Объекты противников

Поскольку у всех типов противников одинаковая структура объекта, рассмотрим только один тип противника. Структура объекта противника дальнего боя изображена на рисунке 3.6.

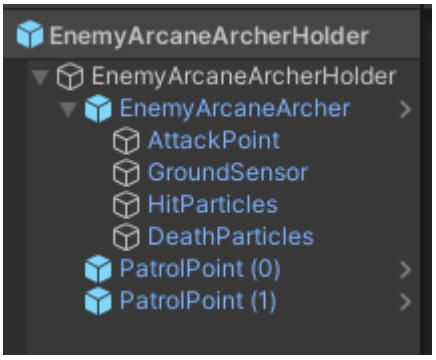


Рисунок 3.6 – Структура объекта противника дальнего боя

В структуру объекта противника дальнего боя входят следующие элементы:

1) главный объект «EnemyArcaneArcherHolder» – хранит в себе объект самого противника и 2 точки для патруля. При желании количество точек может быть увеличено;

2) «EnemyArcaneArcher» – отвечает за всю основную логику противника, в нем находятся следующие компоненты:

а) скрипт «EnemyRangedController» – который описывает ИИ противника и его боевую систему;

б) скрипт «FieldOfView» – для фиксирования вхождения игрока в поле зрения противника;

в) скрипт «Seeker» – компонент библиотеки, с помощью которого строится путь от противника к цели;

г) «SpriteRenderer» – отвечающий за рендер спрайта;

д) «Animator» – отвечающий за анимацию;

е) «AudioSource» – для воспроизведения звуков;

ж) «CapsuleCollider2D» – для обработки коллизий с другими объектами;

з) «RigidBody2D» – для обработки игровой физики;

и) «CapsuleCollider2D» – для обработки коллизий с другими объектами.

3) «AttackPoint» – служит центром области, где игроку будет наноситься удар;

4) «GroundSensor» – объект, в котором расположен скрипт «GroundSensor». Его действие аналогично одноименному скрипту из объекта игрового персонажа;

5) «HirParticles», «DeathParticles» – это объекты с компонентом «Particle System», служащие для графической демонстрации ранения и гибели соответственно.

Для противников дальнего боя были созданы снаряды, изображенные на рисунке 3.7 вместе со структурой объекта.

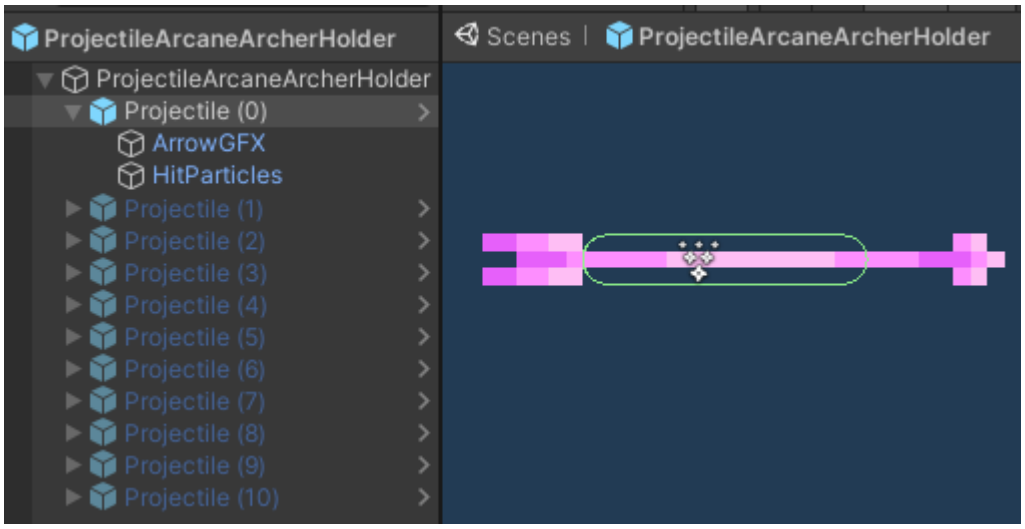


Рисунок 3.7 – Объект снарядов

Для реализации снарядов использован шаблон оптимизации – пул объектов. Это позволяет избежать постоянного инициирования и уничтожения снарядов на сцене, а значит снизить нагрузку на систему и обойти постоянное выделение и освобождение памяти под объекты.

В структуру объекта снарядов входят следующие элементы:

1) «ProjectileArcaneArcherHolder» – хранит в себе пул снарядов;

2) «Projectile» – в нем находятся следующие компоненты:

а) скрипт «Projectile» – описывает логику поведения снаряда;

б) «RigidBody2D» – для обработки игровой физики;

в) «CapsuleCollider2D» – для обработки коллизий с другими объектами;

3) «ArrowGFX» – отвечает за графическое представление объекта на сцене и содержит в себе «SpriteRenderer»;

4) «HitParticles» – это объект с компонентом «Particle System», он служит для графической демонстрации столкновения снаряда с объектами.

## 3.6 Интерфейс

В игре присутствует главное меню, имеющее четыре различных окна:

1) главное окно;

2) окно создания нового сохранения;

3) окно выбора существующего сохранения и его загрузки;

4) диалоговое окно подтверждения действия.

Все окна главного меню изображены на рисунках 3.8–3.11.


Рисунок 3.8 – Главное окно


Рисунок 3.9 – Окно создания нового сохранения


Рисунок 3.10 – Окно выбора существующего сохранения и его загрузки


Рисунок 3.11 – Диалоговое окно подтверждения действия

Помимо главного меню в приложении есть внутриигровое меню, оно включает:

1) окно паузы, при вызове которого включиться пауза, а при отключении игра возобновится;

2) окно просмотра текущих характеристик игрового персонажа, с всплывающей подсказкой, при наведении на какую-либо характеристику;

3) окно повышения уровней характеристик игрового персонажа, с всплывающей подсказкой, при наведении на какую-либо характеристику.

Все окна внутриигрового меню изображены на рисунках 3.12–3.14.


Рисунок 3.12 – Окно паузы


Рисунок 3.13 – Окно просмотра текущих характеристик с всплывающей подсказкой


Рисунок 3.14 – Окно повышения уровней характеристик игрового персонажа с всплывающей подсказкой

## 3.7 Демонстрация игры

Общий вид созданного уровня изображен на рисунке 3.15.

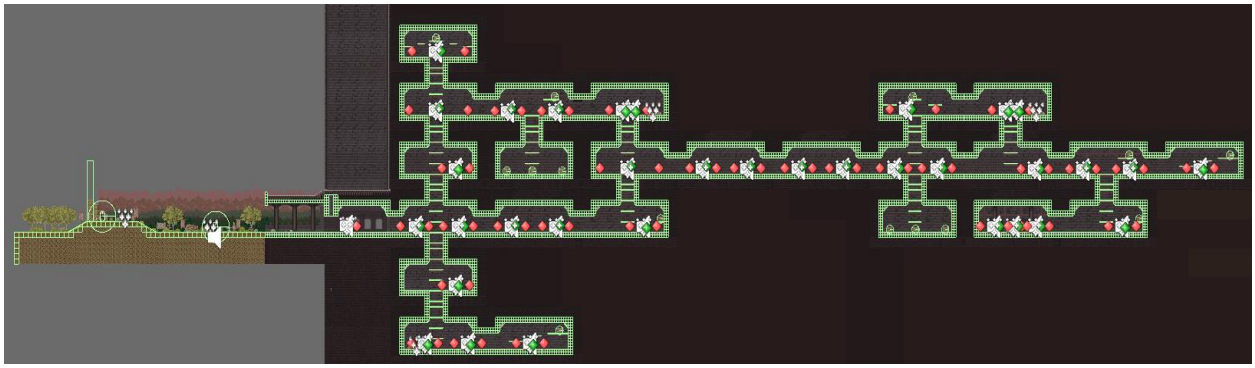

Рисунок 3.15 – Общий вид созданного уровня

Игрок начинает в левой части карты, за пределами замка, в котором и будет происходить большая часть геймплея. Начало игры изображено на рисунке 3.16.

Рисунок 3.16 – Начало игры

Если пройти левее, то игрок обнаружит неигрового персонажа, с которым можно поговорить. На рисунке 3.17 изображён диалог с неигровым персонажем.


Рисунок 3.17 – Диалог с неигровым персонажем

Если же пойти направо, то там будет стоять другой неигровой персонаж, у которого можно повышать уровни характеристик за монеты. Повышение уровней характеристик изображено на рисунке 3.18.


Рисунок 3.18 – Повышение уровней характеристик

Если пройти еще дальше, то там будет замок, при входе в который двери за спиной закроются и останется только идти вперед. Вход изображен на рисунке 3.19.


Рисунок 3.19 – Вход в замок

Как видно на рисунке противник уже идет в сторону игрока, и как только он подойдет ближе, начнет атаковать. На рисунке 3.20 изображена атака врага.


Рисунок 3.20 – Атака врага

Боевая система подразумевает возможность уничтожения врагов. Уничтоженный враг изображен на рисунке 3.21.


Рисунок 3.21 – Уничтоженный враг

# Заключение

В процессе выполнения выпускной квалификационной работы проведен анализ предметной области. В нем был подробно рассмотрен жанр 2D платформеров и один из его наиболее популярных поджанров.

Проведен обзор аналогов, в котором были рассмотрены наиболее популярные и актуальные представители жанра 2D платформеров.

Кроме этого, был проведен обзор различных сред разработки видеоигр с описанием их основных плюсов и минусов и выделением особенностей. Было проведено сравнение рассматриваемых сред разработки и выявлена самая подходящая по выделенным критериям. В результате сравнения Unity занял первое место, в связи с чем он и был выбран как среда для разработки 2D платформера.

В ходе проектирования системы были определены функциональные требования к создаваемой игре с оглядкой на проведенный ранее обзор успешных аналогов. Была описана логика боевой системы, как со стороны игрока, так и со стороны противника, а также разработан ИИ для противников с помощью диаграмм деятельности UML. Помимо этого, спроектирована архитектура классов будущей игры с помощью схемы классов UML.

На завершающем этапе была выполнена разработка видеоигры в жанре 2D платформер на Unity. Подробно описаны основные объекты сцены. Рассмотрен интерфейс приложения. Также была проведена демонстрация полученных результатов.

В ходе выполнения данной работы были достигнуты поставленные цели, выполнены все запланированные задачи, также закреплены профессиональные навыки и умения в сфере индустриальной разработки программного обеспечения.

В дальнейшем разработанную игру можно продолжать наполнять различным новым контентом и интересными механиками, улучшать ее внешний вид и увеличивать ее продолжительность.

## Список использованных источников

1. Комплексное исследование рынка производителей видеоигр в Москве Октябрь 2021г // Исследование агентства креативных индустрий («АКИ») [Электронный ресурс]: сайт «АКИ». URL: https://moscow-creative.ru/finance/research/issledovanie-industrii-po-i-kompyuternyix-igr (дата обращения: 28.05.2022).

2. GamesIndustry.biz presents… The Year in Numbers 2021 // Исследование GamesIndustry.biz [Электронный ресурс]: сайт GamesIndustry.biz. URL: https://www.gamesindustry.biz/articles/2021-12-21-gamesindustry-biz-presents-the-year-in-numbers-2021 (дата обращения: 31.05.2022).

3. ОБЪЕМ РОССИЙСКОГО РЫНКА ВИДЕОИГР ДОСТИГ 177,4 МЛРД РУБЛЕЙ В 2021 ГОДУ // Исследование игрового бренда MY.GAMES [Электронный ресурс]: сайт MY.GAMES. URL: https://my.games/ru/news/183 (дата обращения: 31.05.2022).

4. Premature Evaluation: Dead Cells [Электронный ресурс]: Статья. URL: https://www.rockpapershotgun.com/dead-cells-review-early-access (дата обращения: 28.05.2022).

5. Обзор Dead Cells [Электронный ресурс]: Статья. URL: https://www.igromania.ru/article/30319/Dead_Cells._Tysyacha_i_odin_sposob_vyyti_po_UDO.html (дата обращения: 28.05.2022).

6. Art Design Deep Dive: Using a 3D pipeline for 2D animation in Dead Cells [Электронный ресурс]: Блог художника студии Motion Twin. URL: https://www.gamedeveloper.com/production/art-design-deep-dive-using-a-3d-pipeline-for-2d-animation-in-i-dead-cells-i- (дата обращения: 28.05.2022).

7. Обзор игры Ori and the Blind Forest: самая очаровательная игра 2015 года [Электронный ресурс]: Статья. URL: https://hi-news.ru/games/obzor-igry-ori-and-the-blind-forest-samaya-ocharovatelnaya-igra-2015-goda.html (дата обращения: 28.05.2022).

8. Детали разработки. Как создавался Ori and the blind forest [Электронный ресурс]: Статья. URL: https://stopgame.ru/blogs/topic/103226/

9. Rogue Legacy Review [Электронный ресурс]: Статья. URL: https://www.ign.com/articles/2013/07/26/rogue-legacy-review (дата обращения: 28.05.2022).

10. Rogue Legacy Making Death and Grinding Fun [Электронный ресурс]: Статья. URL: https://www.gameinformer.com/games/rogue_legacy/b/pc/archive/2013/07/16/rogue-legacy-review.aspx (дата обращения: 29.05.2022).

11. Rogue Legacy turned a profit [Электронный ресурс]. – Режим доступа: https://www.vg247.com/2014/03/18/rogue-legacy-turned-a-profit-within-an-hour-on-sale/ (дата обращения: 29.05.2022).

12. The making of Rogue Legacy [Электронный ресурс]: Статья. URL: https://www.eurogamer.net/articles/2013-07-29-the-making-of-rogue-legacy (дата обращения: 29.05.2022).

13. Where did cuphead come from? [Электронный ресурс]: Статья. URL: https://killscreen.com/previously/articles/where-did-cuphead-come/ (дата обращения: 29.05.2022).

14. Unity Game Engine [Электронный ресурс]: Сайт Unity Game Engine URL: https://unity.com/ru (дата обращения: 30.05.2022).

15. ФУНКЦИИ GAMEMAKER [Электронный ресурс]: Сайт GameMaker. URL: https://gamemaker.io/ru/gamemaker/features (дата обращения: 29.05.2022).

16. MAKE GAMES WITH CONSTRUCT 3 [Электронный ресурс]: Сайт CONSTRUCT 3. URL: https://www.construct.net/en (дата обращения: 29.05.2022).

17. Игровой цикл: как удержать игрока у экрана [Электронный ресурс]: Сайт SkillBox Media. URL https://skillbox.ru/media/gamedev/igrovoy_tsikl_kak_uderzhat_igroka_u_ekrana/ (Дата обращения 24.06.2022).

18. Robert Nystrom. Game Programming Patterns. Draft2Digital, 2014. 389с.

19. Order of execution for event functions [Электронный ресурс]: Сайт UnityDocumentation. URL: https://docs.unity3d.com/Manual/ExecutionOrder.html (дата обращения: 24.06.2022).