Министерство науки и высшего образования Российской Федерации

Федеральное государственное автономное образовательное учреждение

высшего образования

«СЕВЕРО-ВОСТОЧНЫЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ

ИМЕНИ М.К. АММОСОВА»

Институт математики и информатики

Кафедра «Информационные технологии»

ДОПУСТИТЬ К ЗАЩИТЕ:

Зав. кафедрой \_\_\_\_\_\_\_\_\_\_/Н.В. Николаева/

Протокол №\_\_\_\_ от «\_\_\_\_» \_\_\_\_\_\_\_\_\_ 2020 г.

**МОБИЛЬНАЯ ИГРА В ЖАНРЕ ROGUELIKE**

**НА ПЛАТФОРМЕ UNITY**

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА

Направление подготовки: 09.03.01 Информатика и вычислительная техника

Профиль: Технологии разработки программного обеспечения

Выполнил: студент IV курса

группы БА-ИВТ-16 ИМИ СВФУ

Шамаев Семен Дмитриевич

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(подпись)

Научный руководитель: старший преподаватель «Информационные технологии» ИМИ СВФУ

Леверьев В.С.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(подпись)

Якутск 2020

**СОДЕРЖАНИЕ**

[ВВЕДЕНИЕ](#_heading=h.rv84qv9ffcin) 2

[ГЛАВА 1. ТЕОРЕТИЧЕСКАЯ ЧАСТЬ](#_heading=h.ewmhsbqlrkjt) 4

[1.1 Обзор предметной области](#_heading=h.g756odvkm8mw) 4

[1.2 Описание игры](#_heading=h.24z70vmhbisa) 6

[1.3 Обзор используемых технологий](#_heading=h.681as56ismhg) 7

[1.3.1 Unity](#_heading=h.gmp5vj6fq6fd) 7

[1.3.2 Microsoft Visual Studio](#_heading=h.x1w8i1wet2wc) 8

[1.3.3 C#](#_heading=h.x5yry8kilwbv) 9

[1.4 Обзор аналогов Unity](#_heading=h.ngapegb0r38q) 10

[1.4.1 Unreal Engine 4](#_heading=h.1fob9te) 10

[1.4.2 Godot](#_heading=h.9ioa0apc3wfo) 11

[1.4.3 Construct 3](#_heading=h.4weuw43rz7f9) 11

[1.5 Обзор игр в жанре roguelike](#_heading=h.ap0q1sl1d1c0) 12

[1.5.1 Rogue](#_heading=h.1nsohich56uh) 12

[1.5.2 Pixel Dungeon](#_heading=h.f4fgbbcnd0ff) 13

[1.5.3 Caves (Roguelike)](#_heading=h.msklz1c79q0u) 14

[Выводы по 1 главе](#_heading=h.105gyw7pnh3d) 15

[ГЛАВА 2 ПРАКТИЧЕСКАЯ ЧАСТЬ](#_heading=h.ozv1qci7l5w0) 16

[2.1 Описание основных принципов проекта](#_heading=h.uf3ba3yfg31p) 16

[2.1.1 Основные требования к проекту](#_heading=h.ogzsraopab9f) 16

[2.1.2 Структура проекта](#_heading=h.jwtudjsk850s) 17

[2.1.3 Интерфейс проекта](#_heading=h.6kqfj6waq1o) 19

[2.2 Этапы разработки](#_heading=h.sowl754b8i1o) 23

[2.2.1 Разработка концепта игры](#_heading=h.x9kn907egzab) 24

[2.2.2 Программирование](#_heading=h.gb13mqci5a65) 25

[2.2.3 Отрисовка графики](#_heading=h.igoxoju1rx1i) 32

[2.2.4 Работа со звуком](#_heading=h.x0mtj56zs79e) 34

[2.2.5 Тестирование игры](#_heading=h.wj97ftder6nu) 35

[2.2.6 Выпуск игры](#_heading=h.c6f5p5ns2w8i) 37

[Выводы по главе 2](#_heading=h.fvbv88d4a565) 37

[ЗАКЛЮЧЕНИЕ](#_heading=h.tyjcwt) 38

[СПИСОК ИСПОЛЬЗОВАННОЙ ЛИТЕРАТУРЫ И ИСТОЧНИКОВ](#_heading=h.3dy6vkm) 39

[ПРИЛОЖЕНИЕ A](#_heading=h.9kb7ltd265lt) 41

# ВВЕДЕНИЕ

В наши дни игровая индустрия является одной из самых быстроразвивающихся индустрий. Уже сложно представить себе человека не знакомого с компьютерными играми ведь каждый из нас так или иначе был связан с ними.

Мобильные игры являются лидером игрового рынка. Доля мобильных игр за прошедший год в мировом игровом рынке составляет 68,5 миллиардов долларов из 152 миллиардов, тогда как доля компьютерных игр составляет всего 35,7 миллиардов, то есть почти в два раза меньше по сравнению с мобильным рынком [16]. Учитывая данную статистику, можно смело сказать, что мобильные платформы на данный момент являются самой перспективной средой для разработки игр. Теперь каждая игровая компания, видя перспективность данного направления, выпускает игры под мобильные платформы, так как они приносят хорошую прибыль, не требуя больших вложений в разработку.

Почти у каждого из нас в кармане лежит смартфон, который уже не используется только для общения как раньше. Теперь смартфоны могут полноценно заменить собой десктопные устройства, а также всегда находятся под рукой. С развитием мобильных устройств развиваются мобильные игры. Раньше сложно было представить, что на мобильных устройствах будут выходить проекты, которые могут на равных конкурировать с компьютерными.

**Актуальность темы дипломной работы** состоит в возросшей популярности мобильных игр, в частности игр в жанре roguelike. А соответственно потребность в разработчиках мобильных игр будет расти.

**Целью** данной дипломной работы является разработка игры в жанре roguelike на платформе Unity.

**Задачи**, поставленные для достижения данной цели:

* Проанализировать игры в этом жанре;
* Выбрать инструменты разработки;
* Разработать игру;
* Протестировать игру;
* Выпустить игру в релиз.

Дипломная работа состоит из введения, двух глав, заключения, списка использованной литературы и приложений.

В первой главе был проведен анализ предметной области, обзор выбранных инструментов разработки и обзор аналогов.

Во второй главе рассмотрена практическая часть разработки, которая состоит из описания основных принципов проекта и этапов разработки.

# 

# ГЛАВА 1. ТЕОРЕТИЧЕСКАЯ ЧАСТЬ

## 1.1 Обзор предметной области

Игра — вид осмысленной непродуктивной деятельности, где мотив лежит не в результате её, а в самом процессе [8].

Компьютерная игра — программа, написанная на одном из языков программирования, с использованием движка, созданная для развлечения [5].

Мобильная игра — игровая программа для мобильных устройств, например сотовых телефонов, смартфонов, коммуникаторов, КПК и прочих (за исключением ноутбуков) [4].

Жанр — это род произведений в пределах какого-нибудь искусства, отличающийся особыми, только ему свойственными сюжетными, стилистическими признаками [6].

Игровая механика — это метод взаимодействия, изменяющий состояние игровой среды [7].

Roguelike — поджанр компьютерных ролевых игр, исторически происходящий от игры Rogue и характеризующийся рядом особенностей, позволяющих безошибочно отличить игру данного жанра.

Особенности жанра roguelike:

* [Принцип «одной жизни»](https://posmotre.li/%D0%9F%D0%B5%D1%80%D0%BC%D0%B0%D0%BD%D0%B5%D0%BD%D1%82%D0%BD%D0%B0%D1%8F_%D1%81%D0%BC%D0%B5%D1%80%D1%82%D1%8C). Так же, как и в реальности, смерть в roguelike — это [навсегда](https://posmotre.li/%D0%A1%D0%BC%D0%B5%D1%80%D1%82%D1%8C_%D0%BD%D0%B0%D0%B2%D1%81%D0%B5%D0%B3%D0%B4%D0%B0). Игра может вообще не иметь возможности сохранения, но чаще позволяется только сохранение для временного прерывания игры. При сохранении происходит выход из игры, а при загрузке сохраненное состояние стирается с диска. Это заставляет игрока бережней относиться к своему персонажу, так как если он умрет — все придется начинать сначала;
* [Случайная генерация мира](https://posmotre.li/index.php?title=%D0%A1%D0%BB%D1%83%D1%87%D0%B0%D0%B9%D0%BD%D0%B0%D1%8F_%D0%B3%D0%B5%D0%BD%D0%B5%D1%80%D0%B0%D1%86%D0%B8%D1%8F_%D0%BC%D0%B8%D1%80%D0%B0&action=edit&redlink=1). Лабиринты, названия свитков с заклинаниями, иногда даже параметры монстров — всё это создается случайным образом при начале новой игры. Вкупе с отсутствием сейвов это создает неопределенность даже ближайшего будущего — игрок не представляет, что его ждет за следующим поворотом, и вынужден всегда быть настороже;
* Пошаговость. Понятия реального времени не существует в принципе, игровое состояние изменяется только одновременно с каким-либо действием персонажа. Зато существует понятие скорости, так что медленный монстр успевает сдвинуться на одну клетку за то время, за которое шустрый персонаж пробежит десять. И наоборот. В последнее время этот атрибут сдаёт позиции, но всё-таки об играх, где надо думать быстро, неприлично говорить как о полноценных roguelike;
* Сюжет и квесты. Хотя существует и главный сюжетный квест, и побочные квесты, 99 % игрового процесса сводятся к уничтожению встречных монстров, что роднит roguelike с [Hack’n’Slash](https://posmotre.li/HackNSlash);
* Сложная и разнообразная игровая модель. Немало игроков любят жанр roguelike только за это. Возможности неоднозначного комбинирования зелий, предметов и ингредиентов, сложное взаимодействие с окружением, когда сильные атаки разрушают двери и стены. Проще всего достигается моделированием через комбинацию свойств, состояний, материалов всего вокруг и их взаимных реакций;
* Проблемы с едой, водой и т. п. Зачастую смерть игрового персонажа происходит не из-за ран монстров, а из-за того, что в пещерах появилось недостаточно еды для постоянного утоления голода [9].

Тайловая графика — метод создания больших изображений (как правило, уровней в компьютерных играх). Изображение составляется из маленьких фрагментов одинаковых габаритов (тайлов).

Тайлы — небольшие изображения одинаковых размеров, которые и служат фрагментами большой картины. Обычно тайлов на один «мир» делают порядка нескольких сотен. Матрица клеток же составляется из номеров тайлов. Таким образом можно строить огромные двухмерные пространства, расходующие совсем немного памяти.

Тайловая графика близка к текстовому режиму. Чтобы сделать из текстового видеоадаптера тайловый, надо добавить ещё одну строку, еще один столбец, изменяемый шрифт, регистры прокрутки и аппаратные спрайты. Поэтому такая графическая система носила еще название «знакоместная графика» [18].

## 1.2 Описание игры

Действие игры происходит в подземелье, где игроку предстоит сражаться с чудовищами, нежитью, окультистами и демонами, заполонившими каждый уголок подземелий. Основная цель игрока — это нахождение сокровища на последнем уровне игры. После его нахождения игра заканчивается победой, а игрок обретает вечную славу. Второстепенной задачей игрока является выживание. Основной опасностью в подземелье являются не только чудовища, но еще и голод. Каждый ход персонажа отнимает у него очки сытости, которые восполняются только с помощью дефицитной еды, редко встречающейся в подземелье. Потеряв все очки сытости, игрок постепенно начинает терять здоровье после чего умирает.

Для достижения данных целей игроку предстоит обдумывать каждое свое действие, так как любая ошибка может привести к перманентной смерти, а соответственно к концу игры без права на продолжение. Также игрок должен грамотно распоряжаться своими ресурсами и выгодно торговаться.

Подземелье состоит из 25 уровней, отличающихся сложностью, окружением и типом врагов. Переход по уровням происходит после того, как игрок дойдет до лестницы, по которой он спускается еще ниже по подземелью. После 20 уровня игра переходит в адский режим, в котором сложность игры увеличивается в разы. Также в адском режиме весь уровень окрашивается в красный цвет, что сигнализирует игроку об опасности.

Обучение происходит посредством учебных табличек, иногда встречающихся в игре. В них хранятся базовые аспекты игры, советы и хитрости, которые помогут игроку легче ориентироваться в игре.

## 1.3 Обзор используемых технологий

### **1.3.1 Unity**

Для разработки игры используется игровой движок Unity.

Unity — это игровой движок, позволяющий создавать игры под большинство популярных платформ. С помощью данного движка разрабатываются игры, запускающиеся на персональных компьютерах (работающих под Windows, MacOS, Linux), на смартфонах и планшетах (iOS, Android, Windows Phone), на игровых консолях (PS, Xbox, Wii) [19].

Выбор данной среды разработки был обусловлен множеством факторов. Одной из них является его простота в использовании, движок имеет низкий порог вхождения, что позволяет легко начать работу с ним. Также одним из преимуществ данного движка является огромная библиотека плагинов и ассетов, которые заметно ускоряют процесс разработки. Можно также отметить, что Unity распространяется бесплатно.

Также Unity особенно хорошо подходит в следующих ситуациях:

* Когда важна скорость разработки;
* Когда требуется полный набор функций, но нет желания создавать собственный комплект инструментов;
* Когда игра не требует частой перерисовки сцены;
* Когда требуется точное управление действиями движка [1, c. 15].

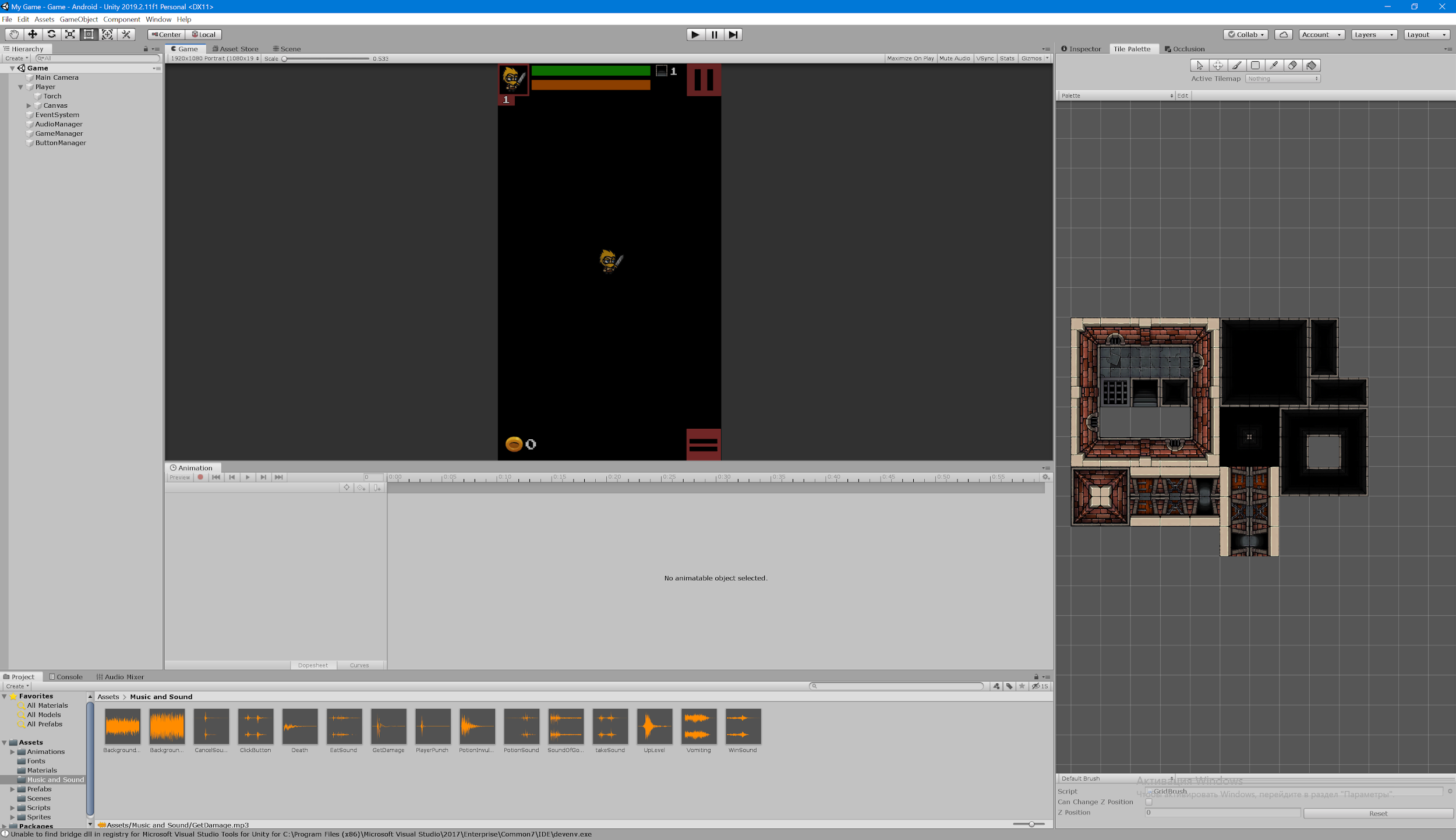


Рисунок 1 – Интерфейс Unity.

### **1.3.2 Microsoft Visual Studio**

Для реализации программного кода в движке Unity используется Microsoft Visual Studio, который имеет возможность интеграции в Unity.

Microsoft Visual Studio — инструментальная среда разработки, включающая в себя интегрированную среду разработки, редактор исходного кода, встроенный отладчик. Многие другие инструменты возможно получить благодаря подключению плагинов – сторонних расширений [17].

Выбор данной среды разработки был обусловлен количеством полезного функционала, среди которого IntelliSense и стандартные функции отладки.

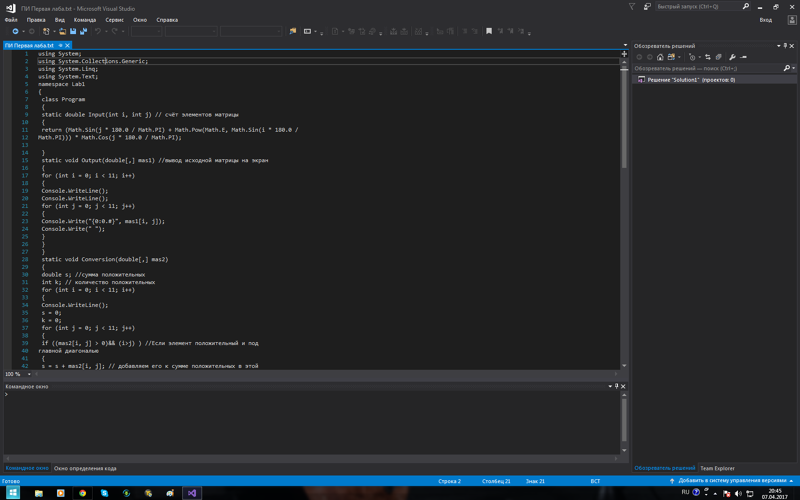


Рисунок 2 – Интерфейс Visual Studio.

### **1.3.3 C#**

Для написания скриптов в разработке игры используется язык программирования C#.

C# — это язык программирования, сочетающий объектно-ориентированные и контекстно-ориентированные концепции. C# относится к семье языков с C-подобным синтаксисом, из них его синтаксис наиболее близок к C++ и Java. Язык имеет строгую статическую типизацию, поддерживает полиморфизм, перегрузку операторов, указатели на функции-члены классов, атрибуты, события, свойства, исключения, комментарии в формате XML. Переняв многое от своих предшественников — языков C++, Delphi, Modula и Smalltalk [10].

Выбор языка программирования стоял между UnityScript и C#. UnityScript больше не поддерживается в Unity, но существует компилятор, который позволяет работать с данным языком программирования. Выбор пал на C#, так как пользоваться UnityScript нецелесообразно из-за отсутствия поддержки языка.

## 1.4 Обзор аналогов Unity

### **1.4.1 Unreal Engine 4**

Unreal Engine один из главных конкурентов Unity. Это движок, в котором разработку может вести даже человек, не разбирающийся в программировании благодаря визуальному редактору BluePrints с помощью которого можно создать игру, не написав ни одной строчку кода. Он обладает множеством полезных и передовых инструментов и открытым исходным кодом. Также игры, созданные на Unreal Engine, куда более производительны.

Основным недостатком Unreal Engine является высокий порог вхождения из-за сложности интерфейса, который подходит больше для опытных разработчиков. Сам движок требует много ресурсов, а недостаток документации плохо сказывается на скорости разработки.

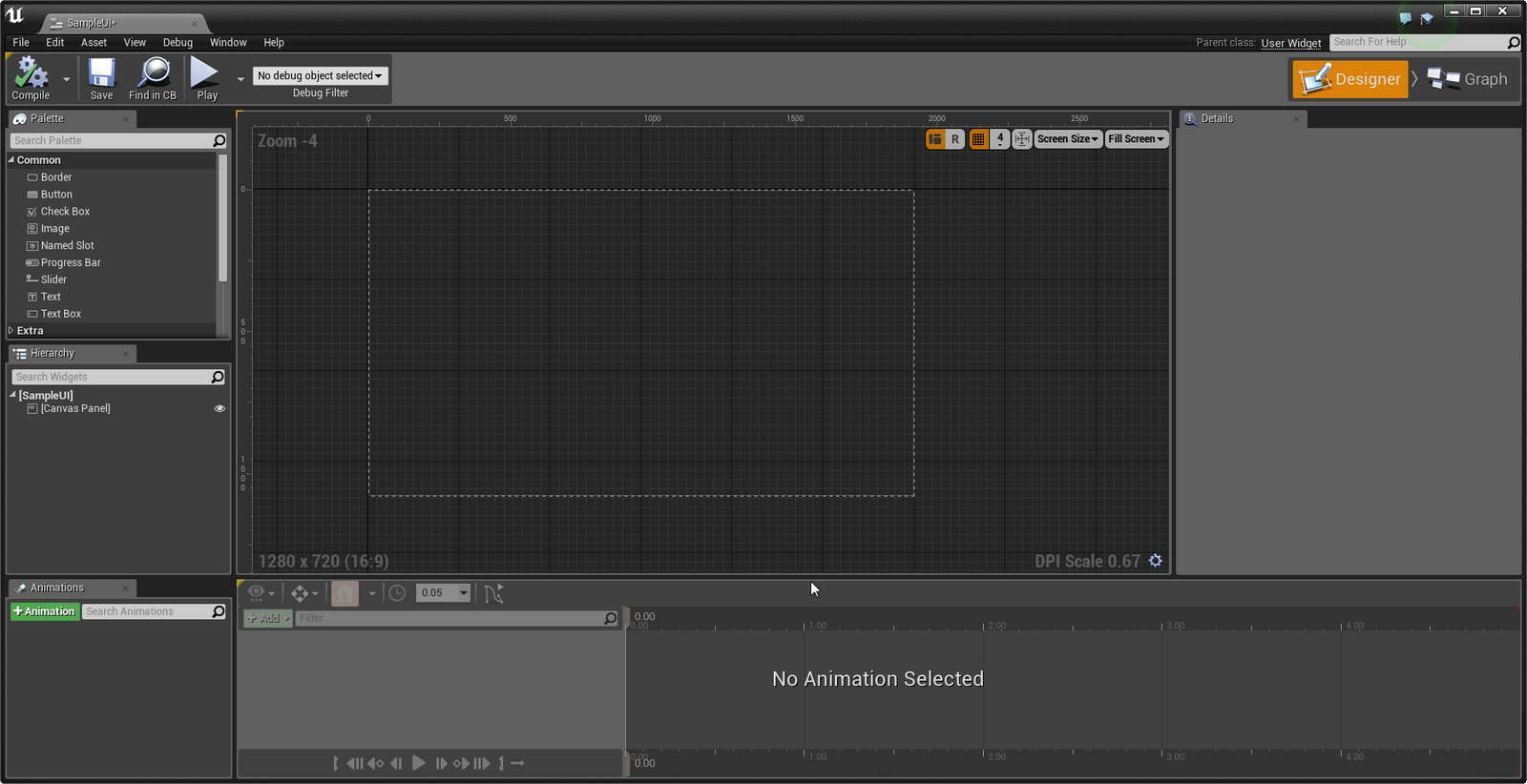


Рисунок 3 – Интерфейс Unreal Engine.

### **1.4.2 Godot**

Godot — еще один игровой движок. Из достоинств можно отметить малый вес самого движка и его неплохую оптимизацию. Также движок обладает удобным и настраиваемым интерфейсом, а открытый исходный код позволит модернизировать движок по своему усмотрению.

Несмотря на эти достоинства, Godot обладает рядом недостатков. Одним из них является собственный язык программирования GDScript, который хоть и является неплохим и слегка походит на Python, но который все равно предстоит выучить тем, кто хочет разрабатывать игры на этом движке. Также стоит отметить недостаток документации.

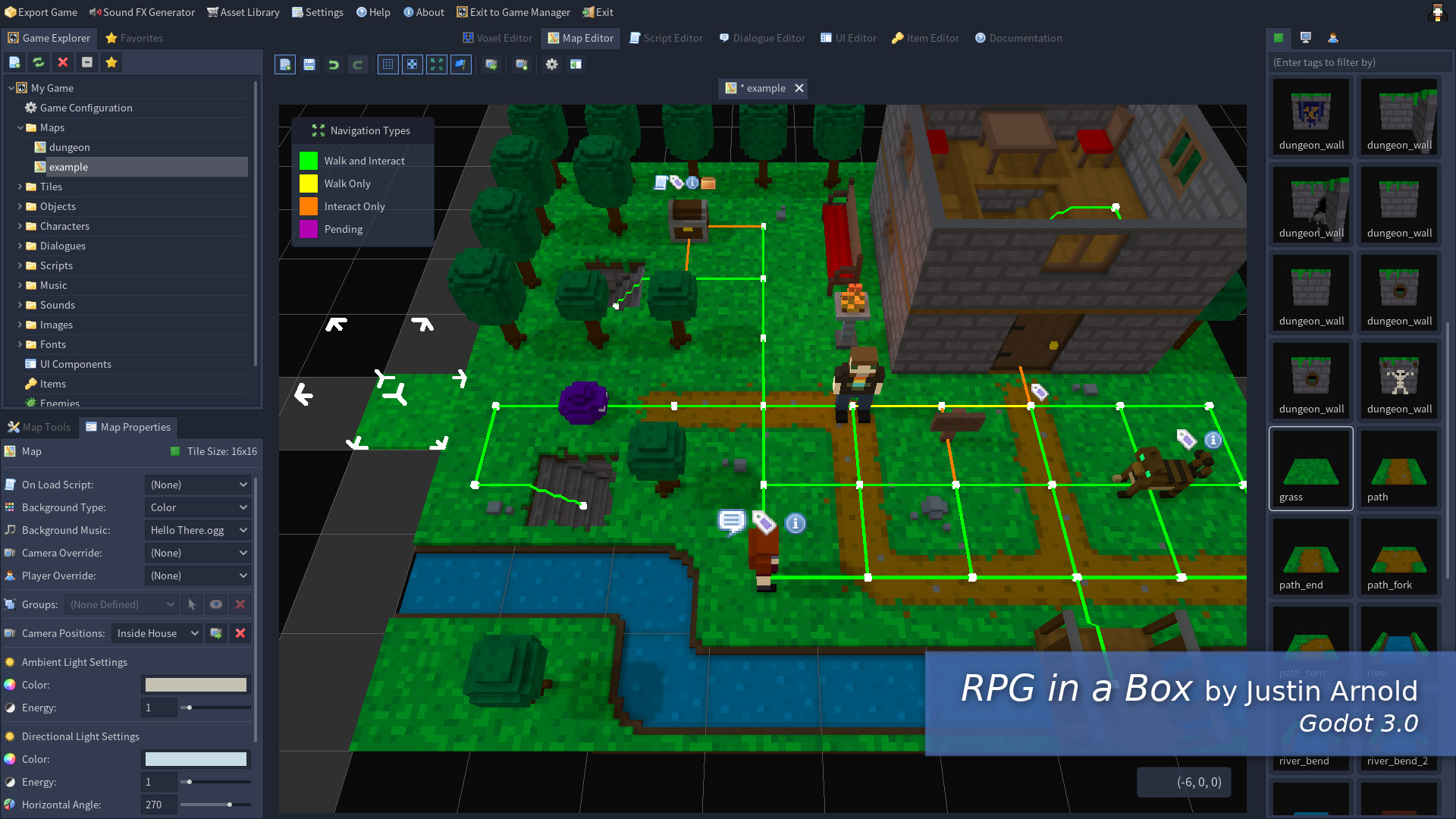


Рисунок 4 – Интерфейс Godot.

### **1.4.3 Construct 3**

Construct — это конструктор для создания игр, больше подходит для новичков, так как не требует написание кода от разработчика и обладает удобным и понятным интерфейсом. Следовательно в конструкторе можно настраивать физику, управление, и другие игровые аспекты буквально за одну минуту.

Из-за своей простоты конструктор обладает рядом недостатков, однако главным недостатком является то, что для полной активации всех функций конструктора придется купить лицензию. Также конструктор довольно нестабилен из-за чего происходят постоянные вылеты.

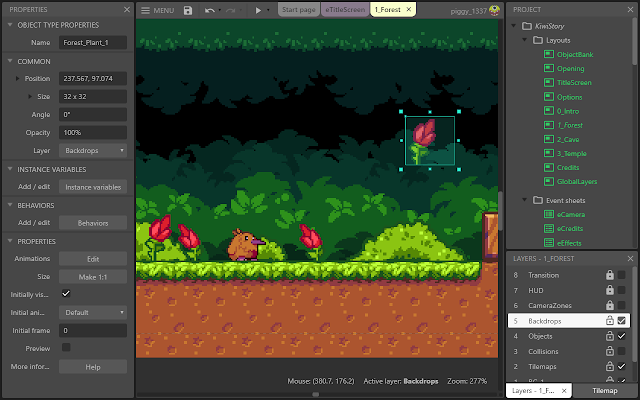


Рисунок 5 – Интерфейс Construct 3.

## 1.5 Обзор игр в жанре roguelike

### **1.5.1 Rogue**

Rogue — компьютерная игра, написанная в 1980 году. Ее основной темой является исследование подземелий. Она была необычайно популярной на университетских Unix-системах в начале 1980-х годов и породила целый жанр roguelike [11].

Графика в данной игре представляла собой набор ASCII-символов, которые позволяли изображать на экране простейшие графические элементы.

Суть игры заключалась в исследовании подземелий, созданных случайным образом для улучшения реиграбельности, а целью игры был поиск амулета Йендора, который находился на самом нижнем ярусе подземелья. Игроку предстояло исследовать не только подземелья, но и почти все элементы игрового окружения, так как в игре не было обучения. Например, для того, чтобы узнать эффект найденного зелья игроку приходилось его выпить, осознавая, что эффект от зелья может быть негативным. Также игра имела механику перманентной смерти благодаря которой игрок начинал ценить каждое свое действие, потому что оно могло привести к смерти, а соответственно к полному концу игры. Позднее эти механики стали классическими для жанра roguelike.

Данная игра является легендой в своем жанре, но в наши дни считается морально устаревшей. Пример процесса игры Rogue показан на рисунке 6.



Рисунок 6 – игра Rogue.

### **1.5.2 Pixel Dungeon**

Pixel Dungeon — мультиплатформенная компьютерная игра в жанре традиционного roguelike с открытым исходным кодом [12].

Pixel Dungeon представляет собой классическую roguelike. Игра обладает процедурной генерацией уровней, перманентной смертью и самостоятельным изучением эффектов элементов игрового окружения.

В начале игры дается выбор между 4 классами персонажа, среди которых воин, разбойник, маг и охотница. Стиль игры будет зависеть во многом от начального выбора класса персонажа.

Суть игры также заключается в поиске амулета Йендора, который дается в конце игры, однако не обладает никакими полезными свойствами и лишь занимает место в инвентаре.

Игру хвалят за реиграбельность, разнообразие контента и свободу выбора, но ругают за примитивную графику и перегруженность механик. Пример процесса игры показан на рисунке 7.



Рисунок 7 – Игра Pixel Dungeon.

### **1.5.3 Caves (Roguelike)**

Caves (Roguelike) — игра в жанре классического пошагового Roguelike, выполненная в пиксель-арт стиле. Главной отличительной особенностью игры является наличие пещер и возможность прокопать себе путь киркой. В игре соседствуют как магия, так и технологии [15].

Игра хвалят за приятную графику и большую вариативность, ругают за плохую реиграбельность и неудобное управление. Пример процесса игры показан на рисунке 8.

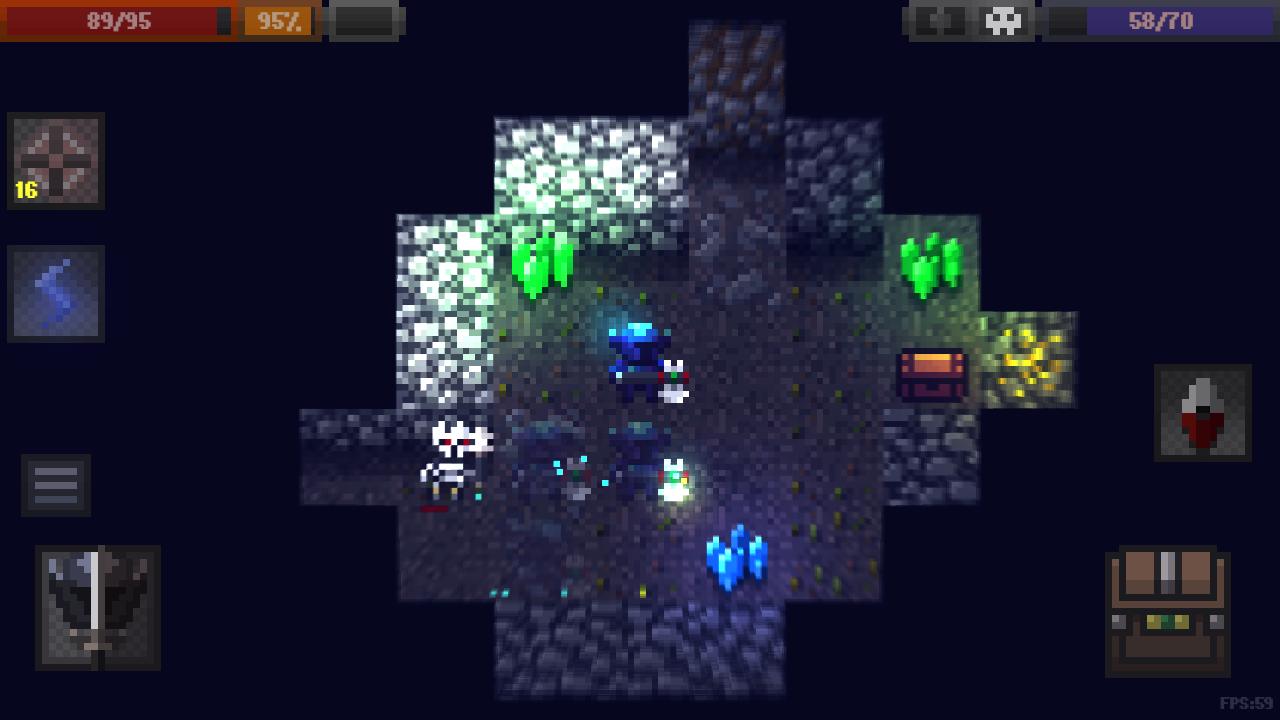


Рисунок 8 – игра Caves (Roguelike).

## Выводы по 1 главе

В первой главе была рассмотрена вся теоретическая часть проекта. Был проведен обзор предметной области, были обозрены все выбранные инструменты разработки и их аналоги, а также аналоги игр в жанре roguelike. Было написано описание разрабатываемой игры.

# ГЛАВА 2 ПРАКТИЧЕСКАЯ ЧАСТЬ

## 2.1 Описание основных принципов проекта

### **2.1.1 Основные требования к проекту**

Основные требования к проекту состоят из:

1. Игра должна иметь возможность запуска на мобильных устройствах на операционных системах Android и iOS;
2. Игра должна быть разработана на платформе Unity;
3. Игра должна обладать удобным и интуитивно понятным интерфейсом;
4. Игра должна соответствовать всем требованиям к играм жанра roguelike в числе которых:
   * Игра должна быть [пошаговой](https://ru.wikipedia.org/wiki/%D0%9F%D0%BE%D1%88%D0%B0%D0%B3%D0%BE%D0%B2%D0%B0%D1%8F_%D1%82%D0%B0%D0%BA%D1%82%D0%B8%D0%BA%D0%B0);
   * Игровые уровни должны генерироваться случайным образом;
   * Игра должна содержать «[перманентную смерть](https://ru.wikipedia.org/wiki/%D0%9F%D0%B5%D1%80%D0%BC%D0%B0%D0%BD%D0%B5%D0%BD%D1%82%D0%BD%D0%B0%D1%8F_%D1%81%D0%BC%D0%B5%D1%80%D1%82%D1%8C)»;
   * Игра должна иметь единый режим и единый набор команд для всех игровых ситуаций;
   * Игра должна предоставлять игроку не какой-то единый линейный путь, а свободу со множеством вариантов прохождения;
   * Игрок должен самостоятельно исследовать найденные предметы и открывать их свойства;
5. В игре должен быть реализован AI игровых персонажей;
6. В игре должен быть реализован алгоритм градации сложности с каждым пройденным уровнем;
7. Игра должна иметь систему прокачки персонажа и его снаряжения;
8. Игра должна иметь систему торговли;
9. Игра должна обладать функцией сохранения и загрузки.

### **2.1.2 Структура проекта**

Проект игры состоит из папок, в которых хранятся файлы разных типов. Основной папкой является папка Asset, в которой хранятся все ассеты проекта. Всего в ней присутствуют 8 папок, хранящих в себе разного рода файлы, среди них:

1. Папка Animations хранит в себе все анимации, включая анимации персонажа, врагов и другого игрового окружения;
2. Папка Fonts хранит в себе разнообразные шрифты;
3. Папка Materials хранит в себе материалы;
4. Папка Music and Sound хранит в себе фоновую музыку и звуковые эффекты в формате MP3;
5. Папка Prefabs хранит в себе экземпляры объектов;
6. Папка Scenes хранит в себе сцены проекта;
7. Папка Scripts хранит в себе скрипты;
8. Папка Sprites хранит в себе спрайты.

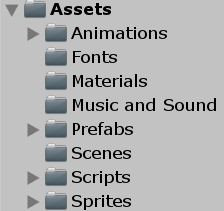


Рисунок 9 – Содержимое папки Assets.

Игра состоит из 2 сцен:

* MainMenu (Главное меню);
* Game (Игровая сцена).

Сцена MainMenu состоит из следующих объектов:

1. MainCamera — основная камера, показывающая что будет видеть игрок во время игры;
2. Canvas — абстрактное пространство, в котором производится настройка и отрисовка UI;
3. Background Menu Music — фоновая музыка;
4. EventSystem — система событий, отправляющая события к объектам проекта;
5. SaveLoadManager — объект, отвечающий за сохранение и загрузку игры. Обладает свойстом DontDestroyOnLoad, что позволяет ему переходить из сцены в сцену.



Рисунок 10 – Содержимое сцены MainMenu.

Сцена Game состоит из следующих объектов:

1. Main Camera — основная камера, показывающая что будет видеть игрок во время игры;
2. Player — объект игрока. В нем также содержится Canvas;
3. EventSystem — система событий, отправляющая события к объектам проекта;
4. AudioManager — объект, отвечающий за воспроизведение фоновой музыки и звуковых эффектов;
5. GameManager — объект, хранящий в себе все данные игры и отвечающий за игровой процесс;
6. ButtonManager — объект, отвечающий за взаимодействие с кнопками и интерфейсом в целом.



Рисунок 11 – Содержимое сцены Game.

### **2.1.3 Интерфейс проекта**

Пользовательский интерфейс проекта реализован с помощью компонентов UI Unity, таких как Canvas, Button, Text, Slider, Image.

Компонент Canvas представляет собой абстрактное пространство, в котором производится настройка и отрисовка UI. Все UI-элементы должны быть потомками игровых объектов, к которым присоединен Canvas [13].

Button представляет из себя стандартную кнопку для вызова определенного события по типу запуска игры.

Text — это компонент UI, позволяющий отображать текст на экране.

Slider — это элемент управления, позволяющий пользователю выбирать числовое значение из заданного диапазона, перетаскивая мышь [14].

Image — это компонент UI, позволяющий отображать спрайты на экране.

Главное меню игры состоит из 4 компонентов Button и одного компонента Text.

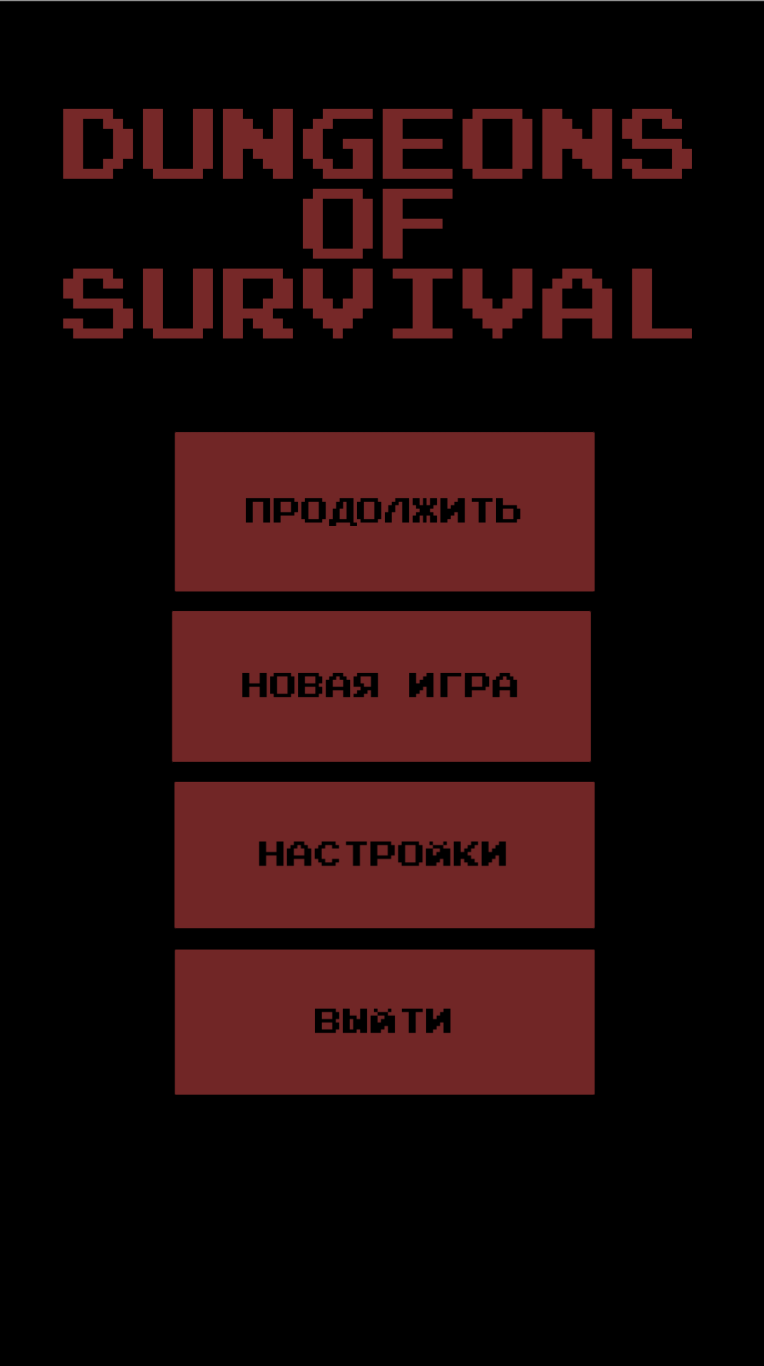


Рисунок 12 – Главное меню игры.

Первый компонент Button отвечает за загрузку уже начатой игры, второй компонент отвечает за создание новой игры, третий компонент отвечает за открытие окна настроек и последний отвечает за выход из игры. Компонент Text - это название игры.

Окно настроек состоит из двух Slider и одного Button.

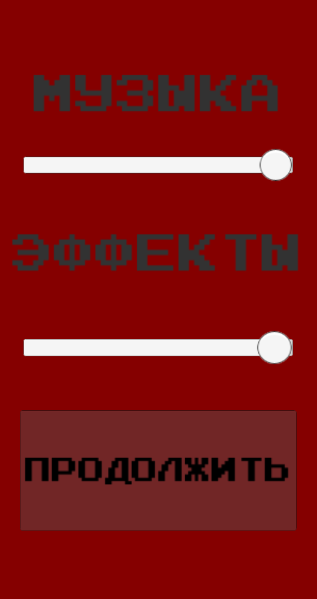


Рисунок 13 – Окно настроек.

Оба Slider отвечает за регулировку громкости игры. Первый регулирует музыку, второй регулирует громкость эффектов. Button отвечает за выход из окна настроек. Значения регулировки компонентов Slider сохраняются после выхода из игры.

Интерфейс игры состоит из двух Button, двух Slider, 3 Image и 5 Text.



Рисунок 14 – Интерфейс игры.

Первый Button отвечает за открытие окна паузы, а второй за открытие инвентаря игрока. Полоски здоровья и сытости игрока были созданы с помощью Slider, так как он отлично подходит для этой задачи. Остальные компоненты отвечают за отображение характеристик игрока, его пейзажа, количества золотых монет и номера уровня, на котором находится игрок.

Меню паузы состоит из 4 Button.



Рисунок 15 – Меню паузы.

Первый Button отвечает за продолжение игры и закрытие окна паузы, второй отвечает за открытие окна настроек, третий за возвращение в главное меню, а последний за выход из игры.

Инвентарь игрока состоит из 25 Button.



Рисунок 16 – Инвентарь игрока.

Каждая ячейка представляет из себя Button, чтобы предметы, которые находятся в инвентаре можно было задействовать прямо в нем. Максимальный размер каждой ячейки составляет 128 единиц. Инвентарь врагов и контейнеров отличается лишь количеством ячеек.

Для удобства в управлении в игре используется портретная ориентация экрана, так как при альбомной ориентации некоторые части экрана остаются недосягаемыми для игрока. При портретной ориентации игрок может задействовать весь экран целиком, что хорошо подходит для разрабатываемой игры, так как в процессе игры задействуется вся площадь экрана. Альбомная ориентация же больше подходит для мобильных игр, в которых элементы управления расположены по бокам, а в разрабатываемой игре они отсутствуют. Пример такой игры показан на рисунке 17.



Рисунок 17 – PUBG Mobile.

## 2.2 Этапы разработки

Разработку игры можно поделить на 6 основных этапов:

1. Разработка концепта игры, в которую входят выбор жанра игры, выбор среды разработки, платформы и целевой аудитории;
2. Программирование (написание скриптов);
3. Отрисовка графики, то есть отрисовка спрайтов, интерфейса, элементов игрового окружения;
4. Работа со звуком;
5. Тестирование игры и проверка работоспособности продукта;
6. Выпуск игры.

### **2.2.1 Разработка концепта игры**

Концепт игры — это то, как выглядит игра на бумаге, то есть просто наброски. Это первый и самый важный этап разработки игры.

В первую очередь на данном этапе разработки необходимо выбрать жанр игры. В данном случае выбор пал на жанр игр roguelike, так как он весьма популярен в наши дни особенно на мобильном рынке.

Также необходимо выбрать сеттинг, то есть среду, в которой происходит действие игры. В качестве сеттинга было выбрано средневековое фэнтези, потому что оно является классическим для жанра roguelike.

Дальше идет выбор среды разработки, то есть среды, на которой будет разрабатываться игра. В качестве среды был выбран кроссплатформенный движок Unity, так как он прост в использовании и всеми необходимыми инструментами для создания игры в жанре roguelike.

После идет выбор платформы, для которой будет разрабатываться игра. В этом случае была выбрана платформа Android, так как она лучше всех подходит для инди-разработчиков. Также на Android весьма популярны игры жанра roguelike.

Также не стоит забывать о выборе целевой аудитории, так как во многом от этого зависит путь, по которому будет разрабатываться игра. В качестве целевой аудитории был выбран контингент подростков мужского пола от 12 до 18 лет. Жанр roguelike в основном популярен у лиц мужского пола, а сложность игрового процесса данного жанра не отпугнет лиц подросткового возраста.

### **2.2.2 Программирование**

Следующим не менее важным этапом является программирование. Программирование осуществляется на языке программирования C# в среде разработки Microsoft Visual Studio.

В ходе работы над игрой было написано 16 разных скриптов. Каждый из скриптов отвечает за ту или иную задачу.

Первым написанным скриптом был скрипт Generator. отвечающий за генерацию игрового поля и объектов на нем. Данный скрипт создает заданное количество комнат и коридоров между ними, а также игровые объекты такие, как вражеские NPC, контейнеры, в которых хранятся полезные для игрока предметы, переходы между уровнями и т.д. Алгоритм его работы можно описать так:

1. Скрипт создает игровое поле заданного размера, которое представляет из себя двумерный массив;
2. Генерирует координаты комнат и коридоров, их ширину и длину;
3. По этим координатам задает каждой ячейке массива тип тайла;
4. Создает объект в каждой ячейке массива по типу тайла;
5. Определяет позиции, на которых можно создать дополнительные объекты;
6. Создает их в заданных позициях.

После перехода на следующий уровень скрипт удаляет все данные, хранящиеся в нем и генерация запускается заново уже с новыми входными данными, генерирующимися случайно, такими, как количество комнат, коридоров, врагов, торговых лавок и т.д.

Пример уровня, созданного с помощью генерации показан на рисунке 18.

Также скрипт Generator использует для генерации два скрипта Room и Corridor, в которых задаются координаты комнат и коридоров.



Рисунок 18 – Пример уровня, созданного с помощью генерации.

Следующий скрипт Player отвечает за управление персонажем, взаимодействием с игровыми объектами и улучшением характеристик, а также хранит в себе данные игрока такие, как уровень персонажа, его очки здоровья и сытости, статус и т.д.

Управление персонажем реализовано с помощью волнового алгоритма (Приложение A). Конечной точкой для игрока является место нажатия по экрану. Самый оптимальный путь к этой точке вычисляется с помощью волнового алгоритма после чего персонаж начинает идти по нему. Волновой алгоритм вычисляет оптимальный путь каждый раз когда игрок переходит из одной клетки в клетку, так как во время движения на пути игрока может появиться вражеский NPC из-за чего старый маршрут окажется неверным из-за появления еще одного препятствия. Нахождение оптимального пути каждый раз после перехода из клетки в клетку решает эту проблему. Пример работы алгоритма показан на рисунке 19.

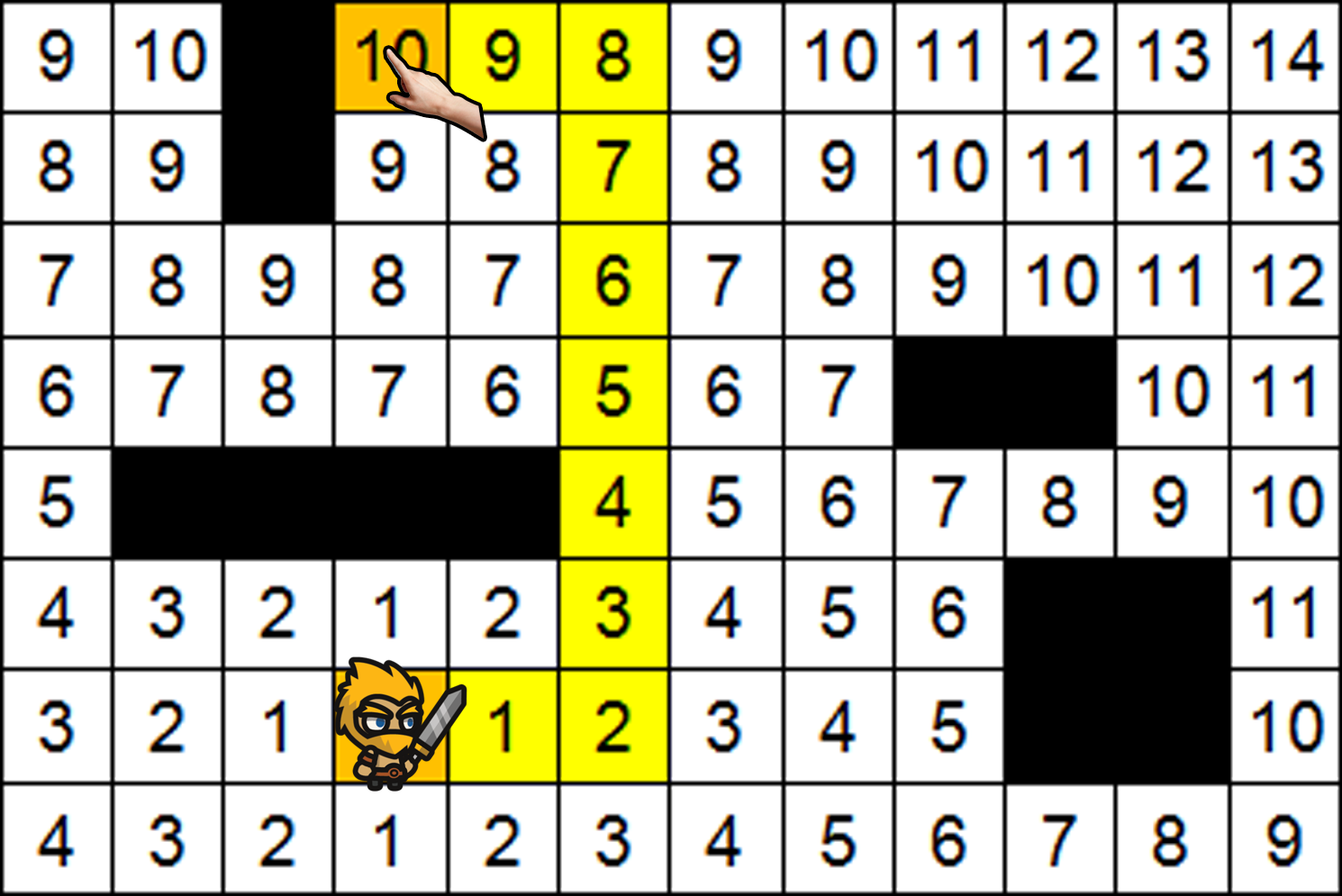


Рисунок 19 – Оптимальный путь к месту нажатия.

Также в скрипте Player описано взаимодействие игрока с игровыми объектами такими, как вражеские NPC, контейнеры, торговые лавки с предметами, зелья и т.д. Например, встретив торговую лавку, игрок может нажать на нее и поторговать или, встретив вражеского NPC, ударить его.

Скрипт Player также отвечает за поднятие уровня персонажа. Скрипт высчитывает сколько нужно очков опыта для поднятия уровня персонажа, если очков хватает, то игроку предоставляются очки навыков, которые используются для повышения характеристик, среди которых:

1. Нападение, которое повышает входящий урон по вражеским NPC;
2. Защита, которая снижает входящий урон по игроку;
3. Ловкость, которая увеличивает шанс промаха по игроку;
4. Выносливость, которая увеличивает максимальное количество очков здоровья и сытости, а также продлевает продолжительность эффекта зелий.

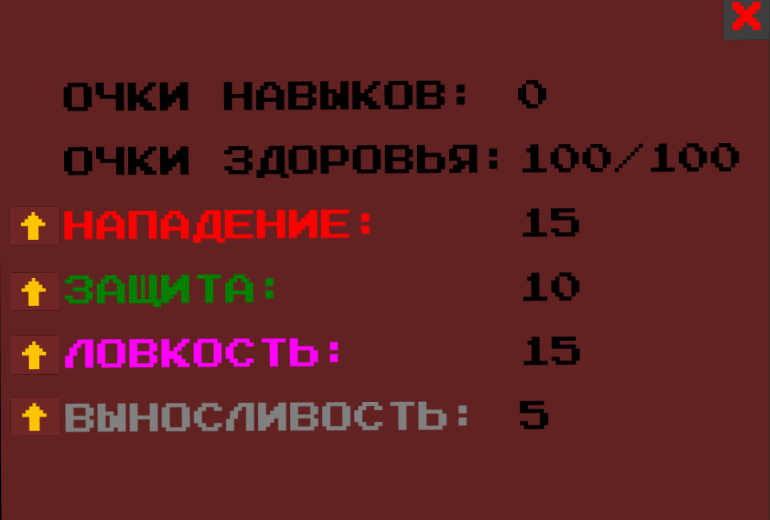


Рисунок 20 – Окно характеристик игрока.

Также в скрипте описаны методы выпивания зелья, смерти игрока, вызывающей конец игры и обновления показателей здоровья и сытости.

Не менее важным скриптом является скрипт Enemy, отвечающий за AI вражеских NPC. Также как и в скрипте Player движение задается с помощью волнового алгоритма (Приложение A). Однако в данном случае конечной точкой является сам игрок. Пример показан на рисунке 21.

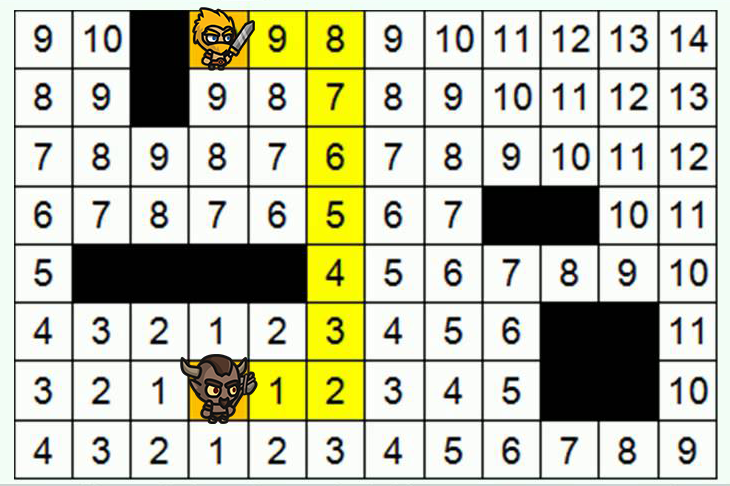


Рисунок 21 – Вражеский NPC высчитывает оптимальный путь к игроку.

Основным скриптов в игре является скрипт GameManager, который соединяет все скрипты воедино. Он отвечает за процесс игры в целом и содержит следующие методы:

1. Start, который инициализирует игру;
2. PauseGame, который ставит игру на паузу. Написан для того, чтобы игрок не мог взаимодействовать с игрой пока находится в окне паузы или в окне повышения уровня;
3. NextGame, который выводит игру из паузы. Используется, когда игрок выходит из окна паузы или повышения уровня;
4. ToNewLevel, который запускается после перехода игрока на новый уровень, Удаляет весь ранее созданный уровень и создает новый;
5. GameOver, который отвечает за окончание игры после смерти игрока. Дает выбор между началом новой игры или выходом в главное меню;
6. Win, который отвечает за появление текста, сигнализирующего о победе игрока. Дает выбор между началом новой игры или выходом в главное меню;
7. SpawnHitTexts, который выводит на экран элемент Text, содержащий в себе количество нанесенного урона по вражескому NPC или игроку.

Скриптом, отвечающим за сохранение и загрузку игры является скрипт SaveLoadManager. Данный скрипт использует сериализацию данных, то есть перевод данных в последовательность битов и дальнейшего восстановления состояния игры из битовой последовательности. На рисунке 22 можно увидеть пример сериализации данных.

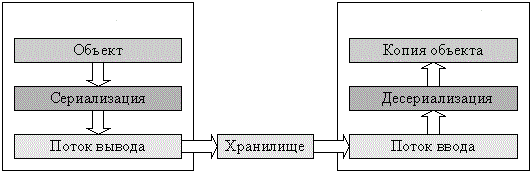


Рисунок 22 – Сериализация данных.

Игра сохраняется только во время выхода из нее и загружается после ее продолжения, а прошлый файл сохранения удаляется. Сделано это для того, чтобы игроки не использовали «дойку», то есть не использовали сохранение для своей личной выгоды. Например для того, чтобы бесконечно пытаться один сложный этап уровня пока не пройдут его. Получается так, что в игре нельзя возвращаться к прошлому промежутку времени благодаря чему игроки будут обдумывать каждое свое действие, так как смерть будет неотвратимой.

Скрипт DataBase содержит в себе информацию о всех предметах, с которыми можно взаимодействовать в инвентаре. Их ID, тип, изображение, стоимость, название и другие параметры, зависящие от типа предмета.

Inventory — скрипт, отвечающий за инвентарь игрока. Содержит в себе все данные об предметах в инвентаре. Также отвечает за взаимодействие с этими предметами. Пример инвентаря показан на рисунке 16.

InventoryEnemy отвечает за инвентари врагов и контейнеров. Позволяет забирать предметы из него и передавать их в инвентарь игрока.

Shop задает механику торговли. Позволяет игроку торговать с лавками торговцев, покупая оттуда предметы за игровое золото. Всего в игре существует 5 типов торговых лавок:

* Лавка оружейника, в которой продается оружие;
* Лавка бронника, в которой продается броня;
* Лавка щитов, в которой продаются щиты;
* Продуктовая лавка, в которой продается еда;
* Лавка зелий, в которой продаются зелья.

Пример торговой лавки показан на рисунке 23.



Рисунок 23 – Лавка бронника.

Buyer задает механику продажи предметов с инвентаря. Позволяет продавать предметы игрока в лавках скупщика. Лавки скупщика появляются в игре довольно редко. Пример лавки скупщика показан на рисунке 24.

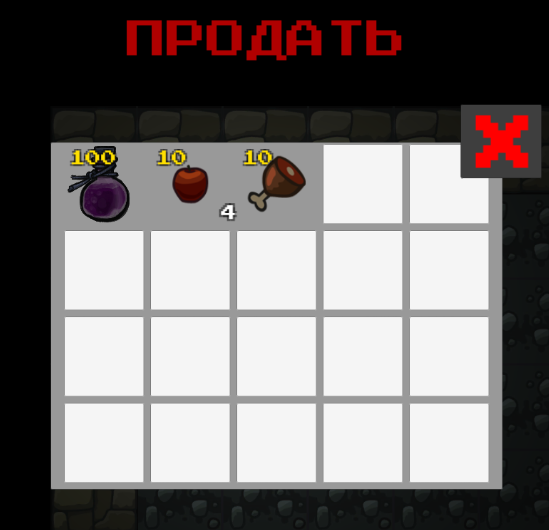


Рисунок 24 – Данные предметы можно продать скупщику.

Скрипт CameraMove позволяет масштабировать и передвигать камеру прямо во время игры. Сделано это для удобства игроков.

Скрипт AudioManager отвечает за воспроизведение фоновой музыки и звуковых эффектов. Также отвечает за увеличение и уменьшение их громкости.

Скрипт ButtonManager отвечает за взаимодействие с элементами Button.

Скрипт Sliders отвечает за взаимодействие с элементами Slider и сохранением их состояния после выхода из игры.

Скрипт Singleton — это скрипт, который гарантирует, что в проекте будет создан только один экземпляр класса и предоставляет глобальный доступ к нему из других классов.

### **2.2.3 Отрисовка графики**

Еще один этап разработки игры — это отрисовка графики. Графика игры состоит из тайлов соответственно в ней реализована тайловая графика. Сделано это из-за пошаговости игры и для более легкой реализации генерации уровней.

Игра состоит из поля площадью 200x200 игровых единиц. Все они заполняются тайлами, кроме тех, в которых ничего нет.



Рисунок 25 – Пример тайлов пола и стены.

В этих тайлах генерируются игровые объекты такие, как вражеские NPC, контейнеры с предметами, торговые лавки и т.д.



Рисунок 26 – Пример спрайтов игровых объектов.

Спрайты персонажей и других игровых объектов были взяты с сайта craftpix.net.

Также была проведена работа с анимацией. Анимации создавались с помощью окна Animation и компонента Animator.



Рисунок 27 – Пример создания анимации.



Рисунок 28 - Пример анимации движения.

Всего в игре присутствуют 34 анимации. 27 из них анимации персонажей, 7 анимации графического интерфейса.

Также в ходе разработки была проведена работа с освещением. Был использован тип освещения Realtime, который проводит расчеты прямо во время работы игры. Это сильно нагружает процессор в отличие типа освещения Baked, который проводит расчеты заранее благодаря чему не влияет на производительность игры. Несмотря на это, был выбран тип освещения Realtime, так как он выдает более реалистичное освещение. Пример работы освещения можно увидеть на рисунке 29.



Рисунок 29 – Освещение.

### **2.2.4 Работа со звуком**

Следующим этапом разработки игры является работа со звуком. На данном этапе происходит подбор фоновой музыки и звуковых эффектов.

В качестве фоновой музыки используется 2 разные композиции, используемые в главном меню и в игровом процессе. Фоновая музыка проигрывается постоянно, повторяясь заново после ее окончания. Сам звуковой файл обрезан так, что игрок не замечает как он начался заново.

Почти у каждого действия в игре есть свой звуковой эффект будь то удар по врагу или поедание еды. Всего в игре 13 звуковых эффектов среди которых:

1. Звук отмены действия;
2. Звук нажатия по кнопке;
3. Звук смерти игрока;
4. Звук поедания еды;
5. Звук удара по игроку;
6. Звук эффекта зелья;
7. Звук испития зелья;
8. Звук золота;
9. Звук взятия предмета из инвентаря;
10. Звук поднятия уровня;
11. Звук рвоты;
12. Звук победы;
13. Звук удара по врагу.

Звуковые эффекты и фоновая музыка были взяты с сайта zvukipro.com.

### **2.2.5 Тестирование игры**

Тестирование игры — это один из последних этапов разработки игры. Здесь происходит проверка игры на наличие ошибок и их исправление.

В ходе тестирования было выявлено большое количество ошибок и недочетов, среди них:

1. Ошибка генерации уровня, связанная с переполнением массива. Это случалось, когда созданная комната создавалась вне массива 200x200. Ошибку удалось устранить с помощью ограничения создания комнат вне массива;
2. Ошибка, связанная с интеллектом вражеских NPC, когда они атаковали игрока, не закончив свой ход из-за чего занимали 2 клетки на поле. Ошибка была решена с помощью условия, которое проверяет закончил ли ход вражеский NPC;
3. Ошибка, связанная с обнулением счетчика пройденных уровней после загрузки игры. Ошибка возникала из-за того, что десериализованный поток вывода загружал в счетчик нулевое значение;
4. Ошибка генерации уровня, когда вражеские NPC появлялись в координате, в которой находиться игрок. Ошибку удалось устранить посредством отсутствия генерации вражеских NPC в начальной стартовой комнате;
5. Ошибка, связанная с инвентарем игрока, когда в редких случаях вещи одного типа добавлялись в разные ячейки инвентаря. Ошибку возникала из-за того, что некоторые предметы не имели своего идентификатора;
6. Ошибка, связанная с состоянием игры, когда она находилась в состоянии паузы, но игрок все равно мог передвигаться по игровому полю. Ошибка была решена с помощью уменьшения масштаба времени до нуля благодаря чему игра теперь полноценно находится в состоянии паузы;
7. Ошибка, связанная с балансом игры, когда прохождение подземелий давалось игроку либо очень легко, либо очень сложно. Ошибку удалось решить с помощью балансировки игровых характеристик вражеских NPC и игрока, также игровых предметов таких, как мечи, щиты, доспехи.

### **2.2.6 Выпуск игры**

Выпуск игры — последний этап разработки игры. Релизная версия проекта собирается с помощью окна Build Settings, где можно выбрать платформу, для которой разрабатывалась игра. В данном случае это платформа Android или iOS. После сборки проекта создается .apk или .ipa файлы, с помощью которых можно установить разработанную игру на данные платформы.

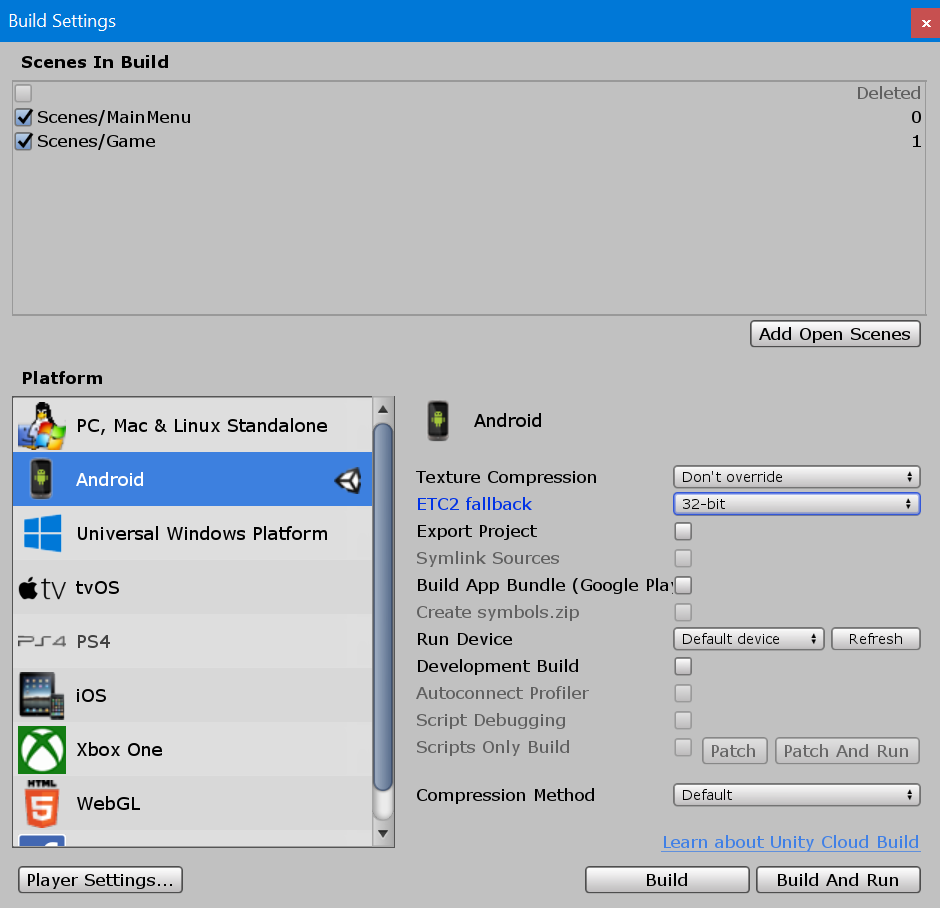


Рисунок 30 – Окно Build Settings.

## Выводы по главе 2

Во второй главе была рассмотрена вся практическая часть проекта. Были описаны все требования к проекту. Был проведен анализ структуры проекта и обзор всех этапов разработки игры.

# ЗАКЛЮЧЕНИЕ

В результате выполнения выпускной квалификационной работы была разработана игра “Dungeons of Survival” в жанре roguelike на платформе Unity. Разработанная игра протестирована на смартфоне под управлением операционной системы Android.

В первой главе была рассмотрена вся теоретическая часть проекта. Был проведен обзор предметной области, были обозрены все выбранные инструменты разработки и их аналоги, а также аналоги игр в жанре roguelike. Было написано описание разрабатываемой игры.

Во второй главе была рассмотрена вся практическая часть проекта. Были описаны все требования к проекту. Был проведен анализ структуры проекта и обзор всех этапов разработки игры.

Разработанная игра соответствует всем канонам жанра roguelike, предоставляет игроку разнообразный геймплей и свободу выбора.

Подводя итоги, можно сказать, что все поставленные задачи решены и цель работы достигнута.

# СПИСОК ИСПОЛЬЗОВАННОЙ ЛИТЕРАТУРЫ И ИСТОЧНИКОВ

Учебники и учебные пособия:

1. Д. Мэннинг/ П. Батфилд-Эддисон: Unity для разработчика. Мобильные мультиплатформенные игры, 2018. - 352 c.
2. Хокинг Дж. Unity в действии. Мультиплатформенная разработка на C#. 2-е межд. изд. — СПб.: Питер, 2019. — 352 с.
3. Бонд Дж. Г. Unity и С#. Геймдев от идеи до реализации. 2-е изд. — СПб.: Питер, 2019. — 928 с.

Электронные ресурсы:

1. Мобильные игры — это… — URL: https://dic.academic.ru/dic.nsf/ruwiki/1042635 (дата обращения: 28.03.2020).
2. Компьютерная игра — URL: http://cyclowiki.org/wiki/Компьютерная\_игра (дата обращения: 20.03.2020).
3. Жанр — это… Что такое жанр? — URL: https://dic.academic.ru/dic.nsf/ushakov/799138. (дата обращения: 20.03.2020).
4. Игровая механика — Манжеты гейм-дизайнера — URL: https://gdcuffs.com/glossary/game\_mechanics/ (дата обращения: 20.03.2020).
5. Теория игр — URL: https://gamesisart.ru/theory\_game.html (дата обращения: 20.03.2020).
6. Roguelike — URL: https://posmotre.li/Roguelike (дата обращения: 04 октября 2018).
7. Язык программирования С# — URL: http://progopedia.ru/language/csharp/ (дата обращения: 15.04.2020).
8. Игра Rogue — URL: https://ru.wikipedia.org/wiki/Rogue (дата обращения: 20.04.2020).
9. Pixel Dungeon — URL: https://ru.wikipedia.org/wiki/Pixel\_Dungeon (дата обращения: 21.04.2020).
10. Холст (Canvas) — URL: https://docs.unity3d.com/ru/current/Manual/class-Canvas.html (дата обращения: 04.05.2020).
11. Slider — Unity — URL: https://docs.unity3d.com/ru/current/Manual/script-Slider.html (дата обращения: 04.05.2020).
12. Caves (Roguelike) wiki — URL: https://caves-roguelike.fandom.com/ru/wiki/Caves\_(roguelike)\_вики (дата обращения: 05.05.2020).
13. Newzoo: global games market — URL: https://gameworldobserver.com/2019/06/24/newzoo-global-games-market-will-reach-152-1-billion-2019/ (дата обращения: 06.05.2020).
14. Microsoft Visual Studio — Wikipedia — URL: https://ru.wikipedia.org/wiki/Microsoft\_Visual\_Studio (дата обращения: 06.05.2020).
15. Тайловая графика — Wikipedia — URL: https://ru.wikipedia.org/wiki/Тайловая\_графика (дата обращения: 07.05.2020).
16. О Unity: Обзор движка — URL: https://itkeys.org/about-unity/ (дата обращения: 08.05.2020).

# ПРИЛОЖЕНИЕ A

*Волновой алгоритм:*

private (int a, int b) FindWave(int startX, int startY, int targetX, int targetY)

{

isStep = false;

int x, y,step=0;

int stepX = 0, stepY = 0;

int[,] cMap = new int[Generator.Instance.MapColumns, Generator.Instance.MapRows];

for (x = 0; x < Generator.Instance.MapColumns; x++)

for (y = 0; y < Generator.Instance.MapRows; y++)

{

if (Generator.Instance.tiles[x][y] != Generator.TileType.Floor &&

Generator.Instance.tiles[x][y] != Generator.TileType.CorridorFloor &&

Generator.Instance.tiles[x][y] != Generator.TileType.Drop)

cMap[x, y] = -2;

else

cMap[x, y] = -1;

}

cMap[startX, startY] = -1;

if(cMap[targetX, targetY] == -2)

{

return (startX, startY);

}

cMap[targetX,targetY]=0;

while (true)

{

for (x = startX - 6; x < startX + 6; x++)

for (y = startY - 6; y < startY + 6; y++)

{

if (cMap[x, y] == step)

{

if (x - 1 >= 0)

if (cMap[x - 1, y] == -1)

cMap[x - 1, y] = step + 1;

if (y - 1 >= 0)

if (cMap[x, y - 1] == -1)

cMap[x, y - 1] = step + 1;

if (x + 1 < Generator.Instance.MapColumns)

if (cMap[x + 1, y] == -1)

cMap[x + 1, y] = step + 1;

if (y + 1 < Generator.Instance.MapRows)

if (cMap[x, y + 1] == -1)

cMap[x, y + 1] = step + 1;

}

}

step++;

if (cMap[startX, startY] != -1)

break;

if (step > 20\*20)

return (startX, startY);

}

x = startX;

y = startY;

step = int.MaxValue;

if (x - 1 >= 0)

if (cMap[x - 1, y] >= 0 && cMap[x - 1, y] < step)

{

step = cMap[x - 1, y];

stepX = x - 1;

stepY = y;

return (stepX,stepY);

}

if (y - 1 >= 0)

if (cMap[x, y - 1] >= 0 && cMap[x, y - 1] < step)

{

step = cMap[x, y - 1];

stepX = x;

stepY = y - 1;

return (stepX,stepY);

}

if (x + 1 < Generator.Instance.MapRows)

if (cMap[x + 1, y] < step && cMap[x + 1, y] >= 0)

{

step = cMap[x + 1, y];

stepX = x + 1;

stepY = y;

return (stepX,stepY);

}

if (y + 1 < Generator.Instance.MapColumns )

if (cMap[x, y + 1] < step && cMap[x, y + 1] >= 0)

{

step = cMap[x, y + 1];

stepX = x;

stepY = y + 1;

return (stepX,stepY);

}

return (startX,startY);

}