МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное автономное образовательное учреждение
высшего образования
**«Южно-Уральский государственный университет
(национальный исследовательский университет)»**
Высшая школа электроники и компьютерных наук
Кафедра системного программирования

РАБОТА ПРОВЕРЕНА

Рецензент
Руководитель отдела
информационных технологий и
связи ООО «Модерн Гласс»

_____ А. Г. Григорьев

“___”_____ 2020 г.

ДОПУСТИТЬ К ЗАЩИТЕ

Заведующий кафедрой,
д.ф.-м.н., профессор

_____ Л.Б. Соколинский

“___”_____ 2020 г.

Разработка компьютерной 2D-игры в жанре «Roguelite» на платформе Unity

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА
ЮУрГУ –09.03.04.2020.308-063.ВКР

Научный руководитель,
к.ф.-м.н., доцент кафедры СП
_____ С.А. Иванов

Автор работы,
студент группы КЭ-403
_____ В.С. Шадчин

Ученый секретарь
(нормоконтролер)
_____ И.Д. Володченко
“___”_____ 2020 г.

Челябинск-2020

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное автономное образовательное учреждение
высшего образования
«Южно-Уральский государственный университет
(национальный исследовательский университет)»
Высшая школа электроники и компьютерных наук
Кафедра системного программирования

УТВЕРЖДАЮ
Зав. кафедрой СП
_____ Л.Б. Соколинский
09.02.2020

ЗАДАНИЕ
на выполнение выпускной квалификационной работы бакалавра
студенту группы КЭ-403
Шадчину Василию Сергеевичу,
обучающемуся по направлению
09.03.04 «Программная инженерия»

1. Тема работы (утверждена приказом ректора от 24.04.2020 № 627)

Разработка компьютерной 2D-игры в жанре «Roguelite» на платформе Unity.

2. Срок сдачи студентом законченной работы: 06.06.2020.

3. Исходные данные к работе

3.1.Руководство Unity. [Электронный ресурс] URL: https://docs.unity3d.com/ru/current/Manual/index.html.

3.2.Nystrom R. Game programming patterns. – Genever Benning, 2014. – 354 с.

4. Перечень подлежащих разработке вопросов

4.1.Провести обзор аналогов.

4.2.Разработать техническое задание на разработку игры.

4.3.Спроектировать архитектуры приложения.

4.4.Реализовать приложение.

4.5.Протестировать приложение.

5. Дата выдачи задания: 08.02.2020.

Научный руководитель
к.ф.-м.н., доцент кафедры СП С.А. Иванов

Задание принял к исполнению В.С. Шадчин

ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ

Основные определения

Компонентно-ориентированное программирование – парадигма программирования, в которой компонент является основной структурной единицей.

Компонент – независимый модуль приложения, предназначенный для повторного использования.

Актуальность темы исследования

Игровая индустрия является одной из самых активно развивающихся в данное время. В 2019 году оборот российского рынка видеоигр достиг $2 млрд. В годовом выражении это рост на 15% [15]. Компьютерные игры давно стали неотъемлемой частью жизни людей.

Индустрия инди-игр переживает подъем, начиная со второй половины 2000-х годов [17]. Инди-игры – это игры, созданные отдельным разработчиком или небольшим коллективом без финансовой поддержки издателя. Распространение интернета и служб онлайн-дистрибуции позволило распространять игры, не осуществляя розничных продаж.

Один из наиболее популярных жанров в индустрии инди-игр является roguelike или roguelite. Этот жанр стал популярным, благодаря своей переиграбельности, каждая новая партия в игре отличается от предыдущей.

Цель и задачи исследования

Целью данной работы является разработка компьютерной 2D-игры в жанре "Roguelite" на платформе Unity.

Для достижения поставленной цели необходимо решить следующие задачи:

1) проведение обзора аналогов;
2) разработка техническое задание на разработку игры;
3) проектирование архитектуры приложения;
4) реализация приложения;
5) тестирование приложения.

Структура и объем работы

Данная работа состоит из введения, 4 глав и заключения. Объем работы составляет 35 страницы, объем библиографии – 22 источника.

1. АНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ

1.1. Обзор аналогов

В ходе выполнения дипломной работы были изучены несколько игр в жанре «roguelite».

The Binding of Isaac: Rebirth

Игра в жанре «roguelite» разработанная Nicalis, Inc. в 2014 году [10]. Является ремейком The Binding of Isaac. В игре игроку необходимо перемещаться по уровням вниз и уничтожать противников. По дороге игроку могут встретиться различные артефакты, монеты, брелоки, ключи, бомбы и другие предметы. По ходу игру игрок становится сильнее и может дойти до последнего босса. Если игрок умирает, то начинает сначала. После каждого прохождения игры общее количество уровней увеличивается, и появляются новые предметы, противники и боссы. К плюсам игры можно отнести большое количество предметов (547 предметов, не считая карты и брелоки) и разнообразные синергии между ними. Игровой процесс представлен на рисунке 1.

Рис. 1. Игровой процесс The Binding of Isaac: Rebirth

Enter the Gungeon

Игра в жанре «roguelite» разработанная Dodge Roll в 2016 году [4]. На январь 2020 года было продано более 3 миллионов копий игры [18] Игроку необходимо дойти до последнего уровня и победить главного босса. В игре присутствует множество видов оружия и артефактов, которые усиливают персонажа. Игра обладает очень быстрым и активным геймплеем и требует от игрока высокой скорости реакции. Игровой процесс представлен на рисунке 2.

Рис. 2. Игровой процесс Enter the Gungeon

Risk of Rain

Игра в жанре «roguelite» разработанная Hopoo Games в 2013 [7]. Игра имеет элементы платформера и roguelite одновременно. Игроку предоставляется возможность играть одним из 12 различных персонажей, каждый из которых обладает своими особенностями. Игрок перемещается по уровням, поднимая различные пассивные и активные артефакты. На каждом уровне присутствует таймер и чем больше времени игрок находится на уровне, тем сильнее становятся противники. Чтобы выйти с уровня необходимо найти портал и убить босса уровня. Игровой процесс представлен на рисунке 3.

Рис. 3. Игровой процесс Risk of Rain

1.2. Анализ существующих решений

В настоящее время используется большое количество игровых движков для разработки игр. Все они имеют свои достоинства и недостатки.

Unity

Unity [19] – это межплатформенная среда разработки компьютерных игр, разработанная компанией Unity Technologies в 2005 году и активно развивающаяся по сегодняшний день. Unity позволяет разрабатывать игры на различные платформы, в том числе персональные компьютеры, игровые консоли, мобильные устройства, интернет-приложения и другое.

Среда разработки имеет модульную систему компонентов. Каждый объект состоит из множества компонентов, где каждый компонент выполняет определенную задачу и может быть использован повторно в других объектах. Объекты находятся на сценах, из которых состоит приложение.

Unity используется как крупными разработчиками, такими как Blizzard [11] и Ubisoft [13], так и инди-студиями и начинающими разработчиками.

Unity имеет бесплатную версию, в которой есть некоторые ограничения, но присутствует практически весь функционал, и имеется возможность распространять игры при условии, что ежегодный доход с игры не превышает 100 000 долларов [20].

Достоинства:

1) мультиплатфоменность;
2) подробная документация;
3) простота использования и освоения;
4) большое сообщество разработчиков;
5) большие возможности по отладке приложения;
6) наличие большого количества пользовательских ассетов.

Недостатки:

1) закрытый исходный код;
2) отсутствие поддержки ссылок на внешние библиотеки.

Unreal Engine

Unreal Engine [22] – это игровой движок, разрабатываемый и поддерживаемый компанией Epic Games, впервые использованный в 1998 году для создания шутера Unreal. Unreal Engine позволяет разрабатывать игры для большинства платформ, а также применяться для работы с графикой в кинематографе. Движок бесплатный, но разработчики обязаны перечислять 5% от общемирового дохода, если ежеквартальная выручка превышает 3000 долларов.

Достоинства:

1) наличие визуального программирования;
2) возможность создание высококачественной графики;
3) открытый исходный код;
4) высокая производительность.

Недостатки:

1) малое количество ассетов;
2) более сложен в освоении;

3) менее удобен для разработки 2D игр.

Godot Engine

Godot Engine [16] – это открытый игровой движок, разрабатываемый сообществом Godot Engine Community. Движок позволяет выпускать игры для большинства современных платформ. В Godot используется собственный скриптовый язык программирования GDScript, который напоминает по синтаксису Python, но в отличие от него имеет чёткую типизацию переменных при объявлении.

Архитектура движка построена на концепции дерева из наследуемых сцен, которая позволяет легко изменять и расширять всю архитектуру проекта.

В Godot присутствует собственный интегрированный редактор скриптов с автоматическим дополнением кода и подсветкой синтаксиса.

Достоинства:

1) большие возможности по созданию 2D игр;

2) наличие визуального программирования.

Недостатки:

1) малое сообщество разработчиков, следовательно, малое количество информации по движку;

2) менее удобен для разработки 3D игр.

1.2. Краткая история и описание жанра roguelite

Предшественником жанра roguelite является roguelike, который был назван в честь игры Rogue 1980 года выпуска. Характерными особенностями данного жанра являются: пошаговое управление, случайно генерируемые уровни и необратимость смерти персонажа. В случае гибели персонажа игрок не может загрузить сохранение и должен начинать игру с самого начала. Многие из представителей жанра имеют ASCII графику, то есть каждый предмет закодирован своим символом. К примеру, персонаж обычно обозначается символом «@». К наиболее известным представите-

лям roguelike относятся Angband[2], ADOM[1], Dungeon Crawl Stone Soup[3] и Tales of Maj'Eyal [9].

В 2000-е годы с расцветом разработки инди-игр начали появляться первые roguelike-подобные игры с нечеткой жанровой принадлежностью. Подобные игры не являлись roguelike в классическом понимании этого термина, а являлись соединением элементов roguelike с другими жанрами. Для таких игр часто применяют термин roguelite или roguelike-like.

Обычно roguelite берут из жанра roguelike случайную генерацию внутриигровых событий и уровней, а также необратимость смерти персонажа, добавляя это в другие жанры. Примером дополнительного жанра roguelite может быть платформер (Rogue Legacy [8]), стратегия (FTL: Faster Than Light [5]), shoot 'em up (Nuclear Throne [6]) или практически любой другой жанр.

Вывод

В ходе анализа предметной области был проведен обзор аналогичных проектов и анализ существующих решений.

По итогу анализа существующих решений для разработки приложения была выбрана платформа Unity в силу своих достоинств и по причине наличия личного опыта использования.

Все материалы, полученные на основе обзора, позволяют определить требования к проектированию и реализации приложения.

2. ПРОЕКТИРОВАНИЕ

2.1. Анализ требований

Разрабатываемое игровое приложение – игра в жанре «Roguelite». Этот жанр характеризуется случайной генерацией событий по ходу игры, перманентностью смерти.

Игра должна быть сконцентрирована на собирании артефактов и улучшений, которые позволяют значительно изменять характеристики персонажа и его оружие. Улучшения позволяют изменить числовые характеристики персонажа или же поведение, количество и эффекты снарядов, которые выстреливает персонаж. Улучшения должны совмещаться вместе и образовывать синергии между собой. Таким образом, игрок должен выбирать подходящие улучшения с учетом уже полученных, которые позволят наиболее эффективно справляться с противниками. Противники также должны иметь различное поведение и характеристики.

2.3. Варианты использования

В ходе анализа требований была разработана UML-диаграмма вариантов использования, которая представлена на рисунке 4.

Актер *Игрок* – пользователь системы. Ему доступны различные способы взаимодействия с игровым миром.

Варианты использования следующие.

Перемещаться – игрок может изменять свои координаты на уровне посредством клавиш управления персонажем.

Атаковать – игрок может атаковать противников различными способами, которые зависит от улучшений, которые у него имеются.

Перейти на следующий уровень – игрок может перейти на следующий уровень, после того как уничтожит всех противников на текущем.

Подобрать улучшение – игрок может поднять улучшение, которое изменит его характеристики, посредством приближения к улучшению.

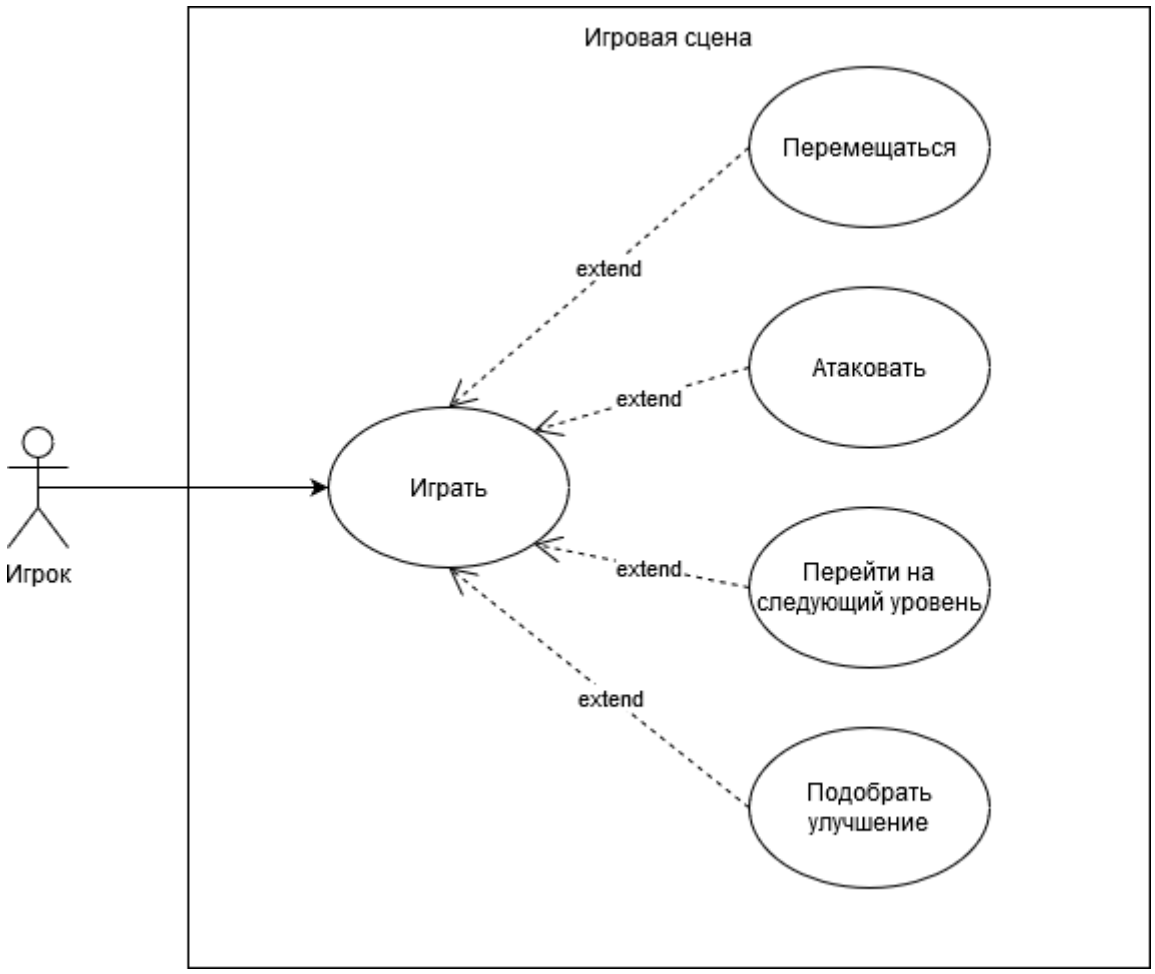

Рис. 4. UML-диаграмма вариантов использования на игровой сцене

Вывод

В ходе проектирования игрового приложения по результатам данных анализа предметной области были разработаны требования для приложения, на основе которых была составлена диаграмма вариантов использования

3. РЕАЛИЗАЦИЯ

3.1. Файловая структура приложения

Проект приложения состоит из набора каталогов, содержащих файлы спрайтов, скриптов, сцен и других игровых компонентов. Файловая структура приложения представлена на рисунке 5.

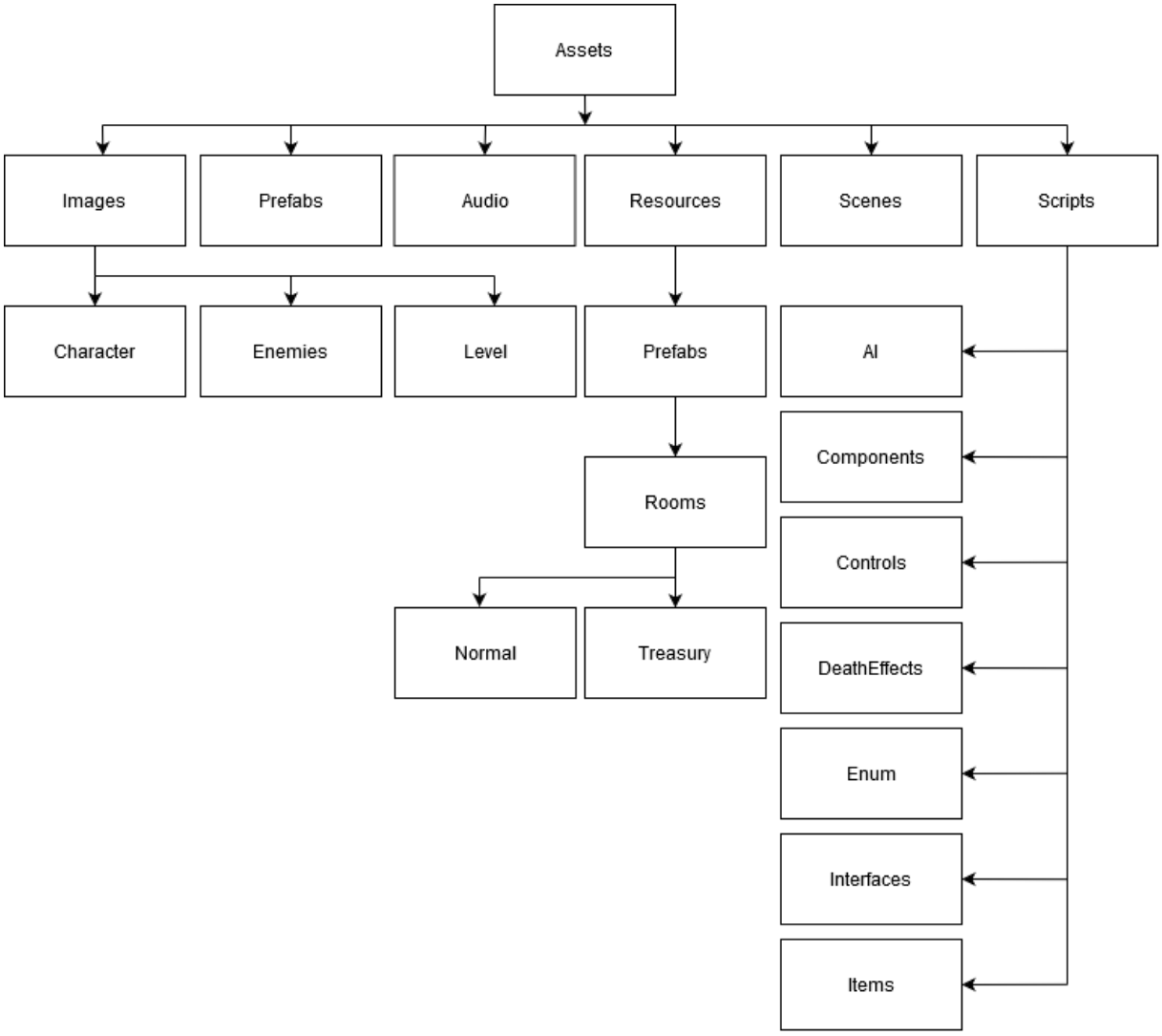

Рис. 5. Файловая структура приложения

Директория Images используется для хранения различных спрайтов и текстур. Изображения рассортированы по директориям в зависимости от предназначения.

Директория Prefabs используется для хранения префабов (шаблонов игровых объектов).

Директория Audio используется для хранения звуков и музыки.

Директория Resources используется для хранения ресурсов, которые загружаются динамически по ходу игры, к примеру, комнаты, которые подгружаются генератором.

Директория Scenes используется для хранения игровых сцен.

Директория Scripts используется для хранения всех скриптов и компонентов.

3.2. Создание компонентов для игровых сущностей

Приложение разрабатывается согласно компонентно-ориентированной парадигме программирования. Каждый объект является контейнером, который заполняется различными компонентами. Большинство игровых сущностей, такие как главный персонаж, противники и снаряды, используют одни и те же компоненты, что позволяет избежать повторения кода и позволяет быстро конструировать новые сущности и значительно изменять их характеристики динамически по ходу игрового процесса. В ходе реализации игрового приложения были разработаны следующие модули:

Move

Представление компонента в инспекторе Unity приведено на рисунке 6. Это основной компонент, отвечающий за передвижение игрового объекта. Данный компонент не указывает, куда и когда передвигается объект, а лишь содержит внутреннее устройство процесса передвижения. Это позволяет не повторять код передвижение в каждом компоненте, который как-либо передвигает объект, и изменять механику передвижения независимо от остальных компонентов. Все компоненты, которые управляют передвижением объекта, обращаются к данному. Компонент содержит параметр Speed, который отвечает за скорость передвижения объекта.

Рис. 6. Компонент Move

LookAtPlayer

Представление компонента в инспекторе Unity приведено на рисунке 7. Компонент, отвечающий за поворот объекта. Если объект имеет данный компонент, то он всегда будет направлен к первому найденному игроку.

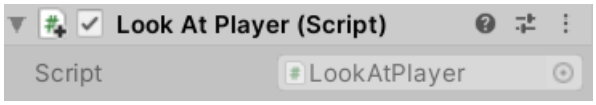

Рис. 7. Компонент LookAtPlayer

Shooter

Представление компонента в инспекторе Unity приведено на рисунке 8.

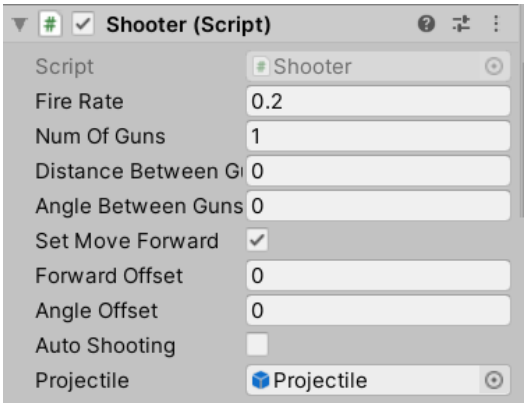

Рис. 8. Компонент Shooter

Компонент shooter отвечает за стрельбу объекта. Под стрельбой подразумевается создание определенного количества других объектов и опциональное задание им скорости и направления. Иногда это можно использовать не только для стрельбы, но и для любого другого повторяемого создания других объектов. К примеру, противники могут создавать своих кло-

нов с помощью этого компонента. Компонент имеет большое количество настроек и позволяет задать разнообразные паттерны стрельбы. На одном объекте может находиться несколько компонентов данного типа, если требуется использовать разные виды стрельбы. Другие компоненты могут запрашивать выстрел двумя способами на выбор: через угол или через вектор. Если другому компоненту необходимо совершить выстрел, то он обращается к методу Shoot, показанному на рисунке 9.

```
public void Shoot(float angle, bool must = false)
    {
        if (Time.time - lastShootTime > fireRate || must)
        {
            for (int i = 0; i < numOfGuns; i++)
            {
                lastShootTime = Time.time;
                lastProjectile = Instantiate(projectile,
gameObject.transform.position, Quaternion.identity);
                lastProjectile.transform.Rotate(new
Vector3(0, 0, angle +
((angleBetweenGuns * (numOfGuns-1)) / 2) - angleBetweenGuns *
i + angleOffset));
                lastProjectile.transform.Translate(
-(distanceBetweenGuns * (numOfGuns - 1)) / 2 + i *
distanceBetweenGuns, forwardOffset, 0);
                lastProjectile.SetActive(true);
                if (setMoveForward)
                {
                    moveVector = (lastProjectile.transform.
TransformPoint(Vector2.up) -
lastProjectile.transform.position).normalized;
                    lastProjectile.
GetComponent<Move>().SetMove(moveVector);
                }
                lastProjectile.transform.parent = gameObject.
transform.parent;
            }
        }
    }
```

Рис. 9. Метод Shoot

Компонент имеет следующие параметры:

Fire rate – параметр, указывающий минимальное время в секундах, которое проходит между выстрелами. Если другой компонент запросит выстрел, пока еще не прошло данное время, то shooter не произведет выстрел.

NumOfGuns – параметр отвечающих за количество создаваемых снарядов.

DistanceBetweenGuns – параметр, отвечающий за расстояния между точками, из которых производятся выстрелы.

AngleBetweenGuns – параметр, отвечающий за угол между направлениями выстрелов. Между всеми выстрелами угол одинаковый.

SetMoveForward – параметр, который указывает, необходимо ли сразу после создания снаряда задать ему движение вперед или нет.

ForwardOffset – дополнительное расстояние, на которое сдвигается точка выстрела вперед по своему направлению. Это помогает настроить точку выстрела в зависимости от формы и размера объекта.

AngleOffset – дополнительный сдвиг на определенный угол. Позволяет задать угол, который направлен не ровно по заданному направлению.

AutoShooting – параметр, который указывает необходимо ли стрелять автоматически с максимально возможной скоростью стрельбы, заданной в параметре Fire rate или же стрельба будет производиться только по запросу от другого компонента.

Projectile – ссылка на объект, который необходимо создавать. Этот объект должен являться потомком данного.

Пример игрового уровня, где противник используют компонент Shooter, который показан на рисунке 10.

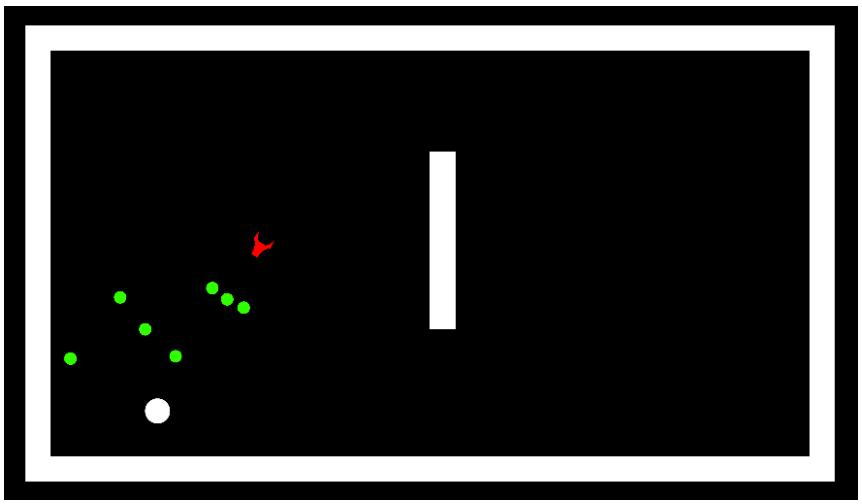

Рис. 10. Использование компонента Shooter противником

PlayerController

Представление компонента в инспекторе Unity приведено на рисунке 11. Компонент, отвечающий за управление персонажем с помощью клавиатуры. Данный компонент содержит ссылки на два других компонента (Shooter и Mov), которые необходимы ему для работы, так как напрямую учувствуют в управлении персонажем.

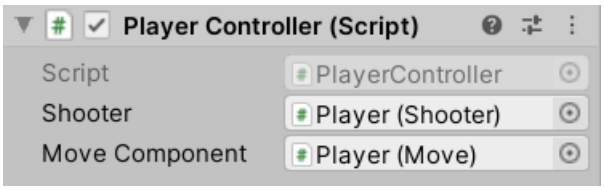

Рис. 11. Компонент PlayerController

Life

Представление компонента в инспекторе Unity приведено на рисунке 12. Компонент, отвечающий за очки здоровья. Любой объект, который имеет данный компонент, получает возможность получать урон. Данный компонент имеет следующие параметры:

MaxHP – максимально количество очков здоровья, которые может получить объект.

HP – количество очков здоровья, которое имеет объект в данный момент.

Status – параметр необходимый для определения, к какой группе относится объект. Всего существует 3 группы объектов: Ally, Enemy и Neutral. В зависимости от группы определяется, может ли объект наносить урон другому объекту. Объекты из одинаковых групп не могут наносить урон друг другу, но могут нанести урон объекту другой группы. Это позволяет избежать ситуации, когда противники уничтожают друг друга.

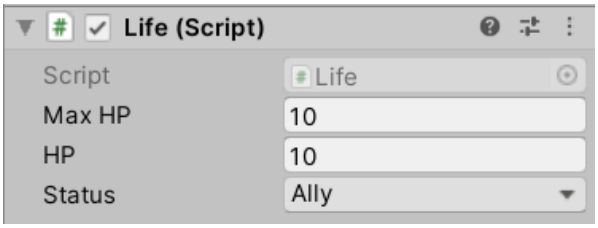

Рис. 12. Компонент Life

DamageDealer

Представление компонента в инспекторе Unity приведено на рисунке 13. Данный компонент отвечает за нанесение урона касанием. Таким образом он должен присутствовать на снарядах и противниках, которые имеют атаки ближнего боя. Компонент наносит урон только тем объектам, которые имеют компонент Life. Компонент имеет следующие параметры:

Damege – количество урона, который наносит данный объект другому при касании.

Status – это параметр схожий с одноименным на компоненте Life. При столкновении объекта имеющего DamageDealer с объектом, который имеет компонент Life, происходит сравнение их параметров Status. Если они разные, то наносится урон.

DestroyAfterDealDamage – параметр, отвечающий за необходимость уничтожить объект при нанесении урона. Если данный параметр равен false, то объект просто пройдет сквозь после нанесения урона.

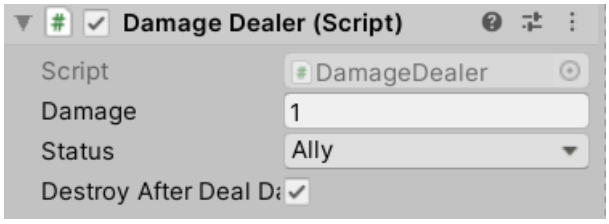

Рис. 13. Компонент DamageDealer

DelayedDeath

Представление компонента в инспекторе Unity приведено на рисунке 14. Данный компонент уничтожает объект через определенное время после его создания. Он имеет только один параметр Time. Этот параметр отображает время в секундах, через которое объект должен быть уничтожен.

Рис. 14. Компонент DelayedDeath

ShootAI

Представление компонента в инспекторе Unity приведено на рисунке 15. Компонент управляющих стрельбой. Данный компонент автоматически стреляет в сторону игрока, когда удовлетворяется условия стрельбы. Компонент имеет следующие параметры:

Shooter – ссылка на компонент Shooter. Паттерн стрельбы определяется именно настройками данного компонента.

Range – расстояние, начиная с которого компонент начинает стрельбу по игроку.

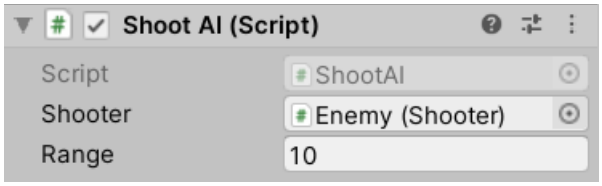

Рис. 15. Компонент ShootAI

MoveAI

Представление компонента в инспекторе Unity приведено на рисунке 16. Компонент, управляющий движением противника. Противник двигается к игроку, пока не приблизится на расстояние, указанное в параметре RangeToStop или ближе.

Рис. 16. Компонент MoveAI

ShootOnDeath

Представление компонента в инспекторе Unity приведено на рисунке 18. Данный компонент является эффектом уничтожения. Подобные эффекты срабатывают перед уничтожением объекта посредством вызова статической функции DestroyWithDeathEffect, изображенной на рисунке 17.

```
      public static async void DestroyWithDeathEffects(GameObject
      destroyedObject,   float   time   =   0,   DeathCause   deathCause   =
DeathCause.All)
          {
              await Task.Delay((int)(time * 1000));
              if(destroyedObject != null)
              {
                  IDeathEffect[] deathEffect =
      destroyedObject.GetComponents<IDeathEffect>();
                  foreach (IDeathEffect effect in deathEffect)
                  {
                      if(effect.AllowedDeathCause ==
      DeathCause.All || effect.AllowedDeathCause == deathCause)
                      {
                          effect.ActivateEffect();
                      }
                  }
                  Object.Destroy(destroyedObject);
              }
          }
```

Рис. 17. Функция DestroyWithDeathEffect

Компонент имеет следующие параметры:

Shooter – ссылка на компонент Shooter. При срабатывании эффекта будет использоваться паттерн стрельбы из указанного компонента shooter.

AllowedDeathCause – причина уничтожения объекта, при которой данный компонент может сработать. Если причина не будет совпадать с указанной, то данный эффект просто игнорируется. На данный момент существует 4 вида причин: time, collision, kill и all. Time срабатывает только если объект уничтожается по причине слишком долгого времени существования, к примеру это может быть эффект компонента DelayedDeath. Collision срабатывает, если объект уничтожился из-за столкновения, такое может случиться при установке параметра DestroyAfterDealDamage в компоненте DamageDealer. Kill срабатывает при уничтожении объекта посредством снижения его количества очков здоровья. All срабатывает при любой причине из вышеперечисленных.

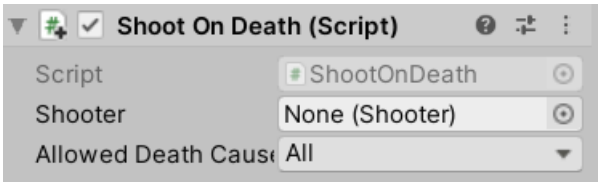

Рис. 18. Компонент ShootOnDeath

3.3. Создание генератора уровней

В игре персонаж последовательно перемещается из комнаты в комнату и не может возвращаться обратно. Комната выбирается из заранее созданной библиотеки ресурсов в зависимости от количества уже пройденных комнат и зоны, в которой находится игрок. Пример комнаты с несколькими стенами и двумя противниками представлен на рисунке 19.

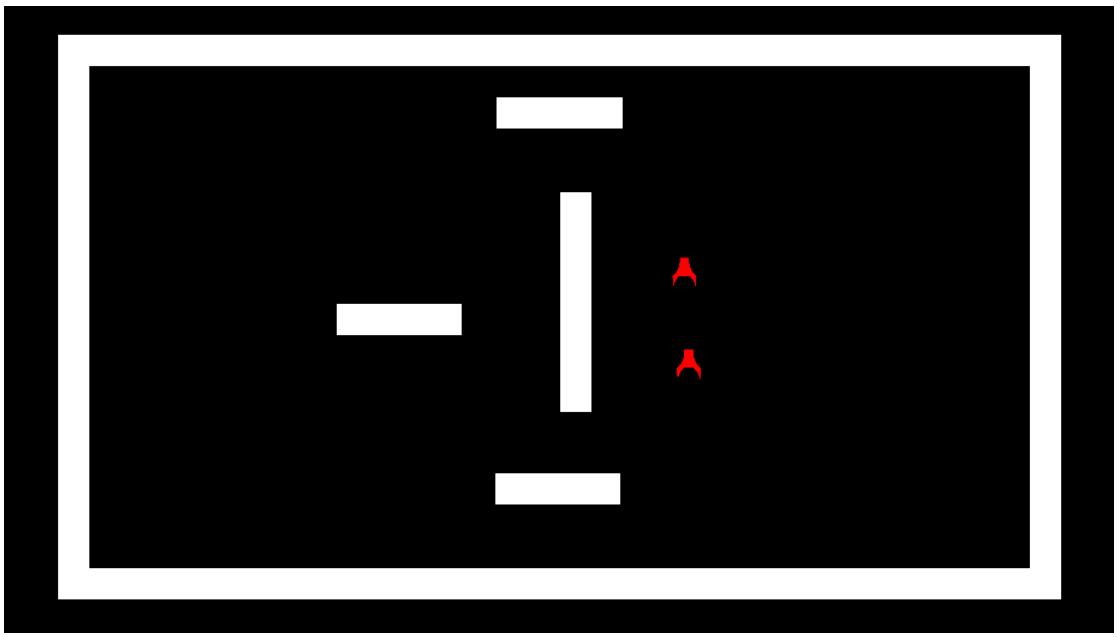

Рис. 19. Пример комнаты

После того как игрок уничтожит всех противников создается портал, который ведет в следующую комнату. Генератор выбирает комнату, удовлетворяющую параметрам, и создает ее методом CreateRoom, показанном на рисунке 20.

```
      void CreateRoom(string path)
          {
              player.transform.parent = gameObject.transform;
              Destroy(room);
              room = Instantiate(Resources.Load(path) as GameObject,
gameObject.transform.position, Quaternion.identity);
              room.transform.parent = gameObject.transform;
              player.transform.position =
room.transform.Find("Spawn").transform.position;
              player.transform.parent = room.transform;
          }
```

Рис. 20. Метод CreateRoom

Компонент, содержащих этот метод, привязывается к объекту, который содержит в себе текущую комнату и игрока.

Все комнаты хранятся в специальных директориях разделенные по типу и предназначению. Каждая комната имеет выход, который вначале

скрыт, и активируется только после уничтожения всех противников. Выход содержит на себе компонент Exit, который обращается к генератору при приближении игрока и таким образом создает следующую комнату. Метод, создающий новую комнату, показан на рисунке 21.

```
        void OnTriggerEnter2D(Collider2D col)
        {
          if(col.gameObject.tag == "Player")
          {
              Generator generator =
gameObject.transform.parent.parent.gameObject.
GetComponent<Generator>();
              generator.NextLevel();
          }
      }
```

Рис. 21. Метод OnTriggerEnter2D

3.4. Создание улучшений

Благодаря компонентно-ориентированной архитектуре приложения значительно упрощается создание улучшений, которые каким-либо образом изменяют характеристики персонажа. Каждое улучшение является предметом, который применяется к персонажу при приближении. Каждое улучшение выглядит по-разному, благодаря которому игрок может их отличать друг от друга. Улучшения имеют на себе компонент Item, в котором указано, какой компонент стоит привязать к персонажу, получившему данное улучшение. Эти компоненты могут, как и единоразово изменять характеристики персонажа при получении, так и влиять на него постоянно. Также эти компоненты всегда имеют интерфейс IItem. Ниже приведены некоторые примеры подобных компонентов:

ItemNumOfGuns

Данный компонент увеличивает количеств выстрелов, которые делает персонаж на единицу. Также он изменяет расстояние между выстрелами,

чтобы они не производились из одной точки и не перекрывали друг друга. Код компонента показан на рисунке 22.

```
public class ItemHPUp : MonoBehaviour, IItem
{
    void Start()
    {
        gameObject.GetComponent<Life>().maxHP += 1;
        gameObject.GetComponent<Life>().HP += 1;
    }
}
```

Рис. 22. Компонент ItemNumOfGuns

Изменение характеристик происходит посредством обращения к компоненту Life, который находится на персонаже. Стрельба персонажа при поднятии данного улучшения показана на рисунке 23. Как видно теперь персонаж выстреливает по 2 выстрела за раз, вместо одного.

Рис. 23. Стрельба персонажа при поднятии улучшения ItemNumOfGuns

ItemExplosiveProjectiles

Данный компонент добавляет снарядам новое поведение. Когда снаряды уничтожаются, то они распадаются на 10 своих клонов, которые разлетаются во все стороны. Код компонента показан на рисунке 24.

При активации данного улучшения создается новый снаряд, который является копией уже существующего и его потомком. На первоначальном снаряде создается компонент Shooter, у которого задаются количество выстрелов, угол между выстрелами и ссылка на новый созданный снаряд. Далее на первоначальный снаряд добавляется компонент ShootOnDeath, в котором указывается ссылка на созданный Shooter. Стрельба персонажа при поднятии данного улучшения показана на рисунке 25.

```
public class ItemExplosiveProjectiles : MonoBehaviour, IItem
{
    Transform newProjectile, oldProjectile;
    Shooter shooter;
    ShootOnDeath shootOnDeath;
    void Start()
    {
        oldProjectile = transform.GetChild(0);
        newProjectile = Instantiate(oldProjectile,
gameObject.transform.position, Quaternion.identity);
        newProjectile.SetParent(oldProjectile, false);
        shooter = oldProjectile.gameObject.
AddComponent(Type.GetType("Shooter")) as Shooter;
        shooter.projectile = newProjectile.gameObject;
        shooter.numOfGuns = 10;
        shooter.angleBetweenGuns = 36;
        shootOnDeath = oldProjectile.gameObject.
AddComponent(Type.GetType("ShootOnDeath")) as ShootOnDeath;
        shootOnDeath.shooter = shooter;
    }
}
```

Рис. 24. Компонент ItemExplosiveProjectiles

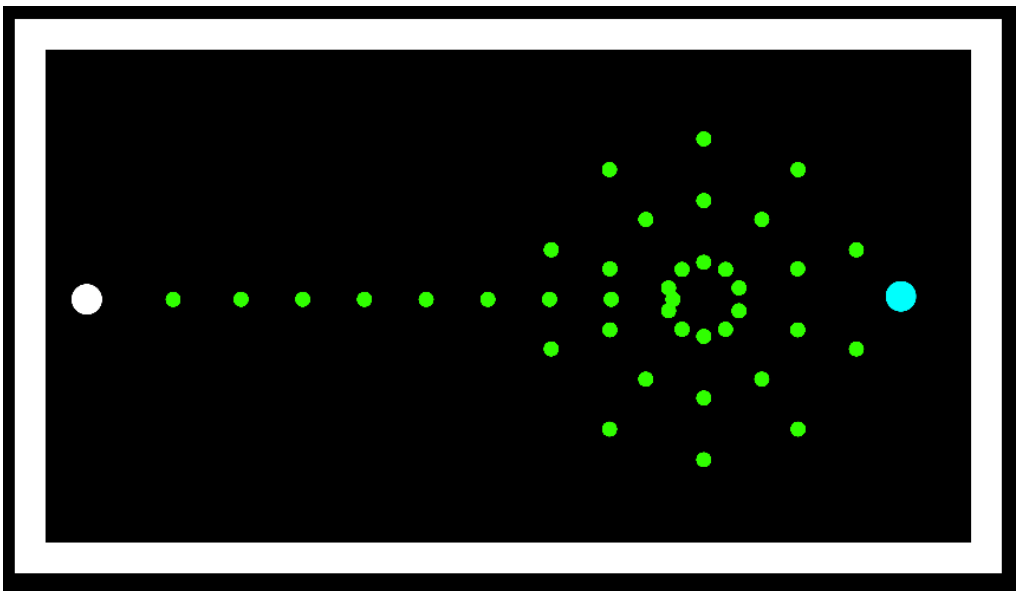

Рис. 25. Стрельба персонажа при поднятии улучшения ItemExplosiveProjectiles

Таким образом, создание улучшений не требует написание большого количества нового кода, за счет использования компонентов и изменения их параметров. Улучшения не перезаписывают друг друга и работают вместе одновременно, что позволяет создавать синергии между ними. На этом и основан основной принцип игры. Игрок может создавать очень сложные комбинации из улучшений, которые могут кардинально изменить стиль игры. К примеру, стрельба персонажа, поднявшего оба улучшения одновременно, показана на рисунке 26.

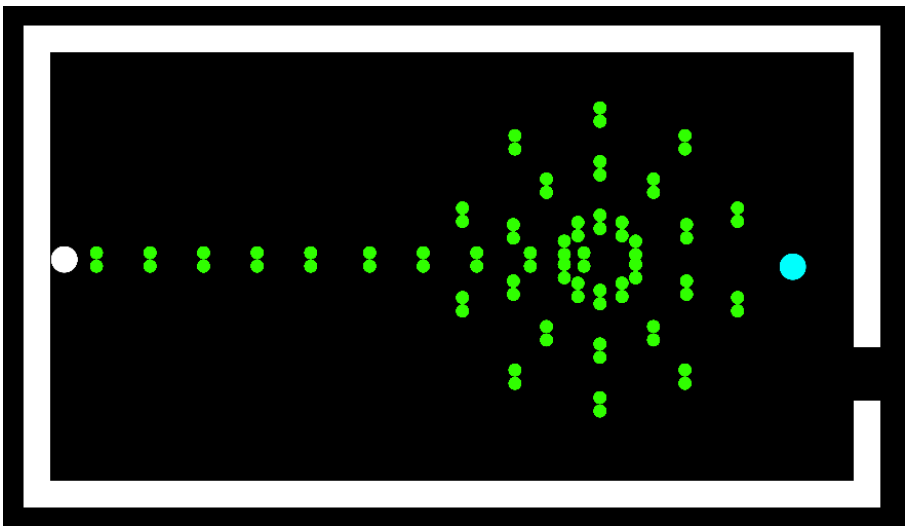

Рис. 26. Стрельба персонажа при поднятии нескольких улучшений

3.4 Создание интерфейса

С помощью векторного графического редактора CorelDRAW 2019 [12] был создан экран смерти и экран победы, показанные на рисунке 27, и добавлены в игру посредством инструмента Unity UI [21].

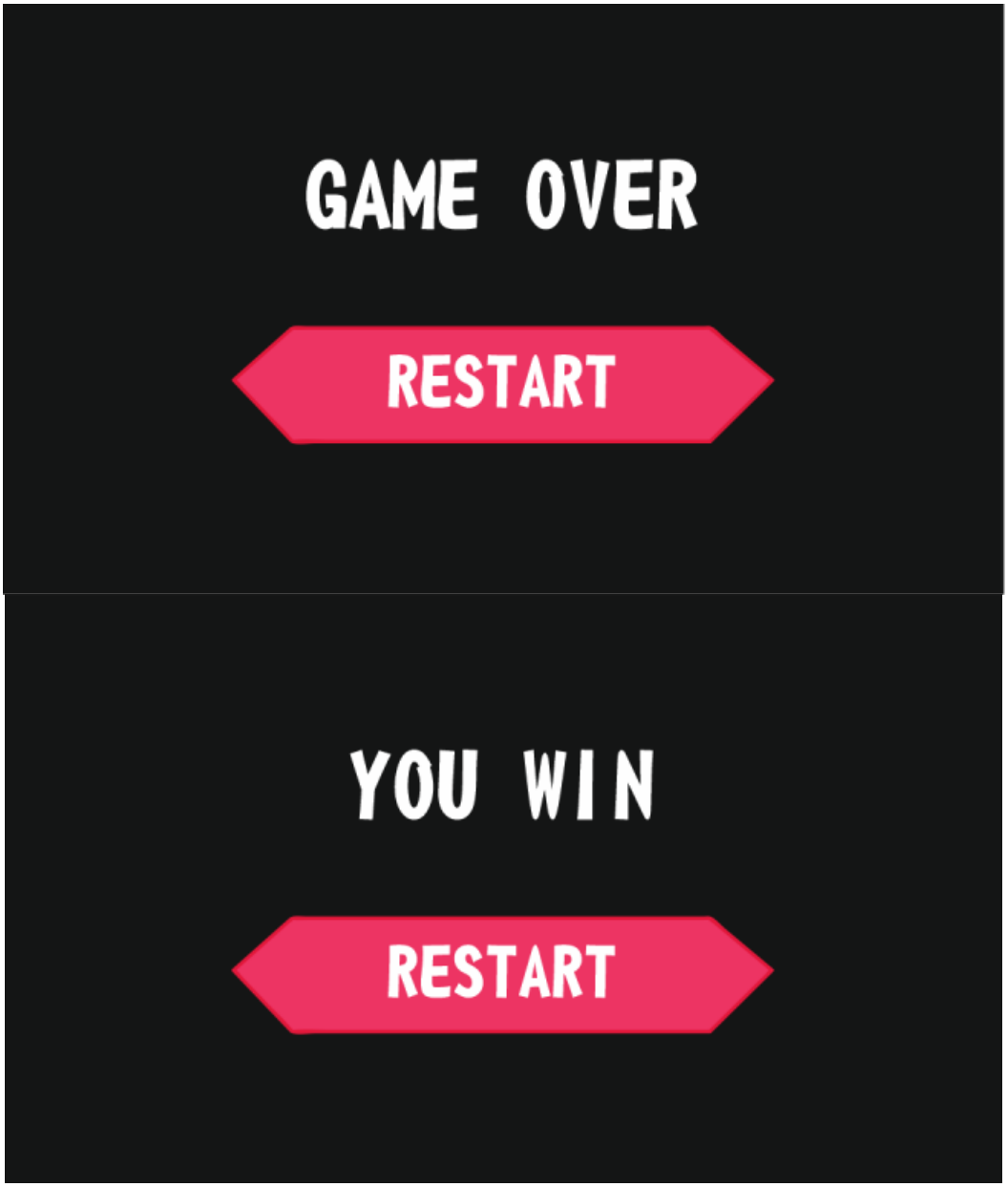

Рис. 27. Экран смерти и экран победы

Вывод

В ходе реализации была продемонстрирована файловая структура приложения, показывающая набор каталогов проекта игрового приложения. Далее была описана реализация игрового приложения.

4. ТЕСТИРОВАНИЕ

Функциональное тестирование

Функциональное тестирование – это тестирование программного обеспечения в целях проверки реализуемости функциональных требований, то есть способности программного обеспечения в определенных условиях решать задачи, в соответствии с зафиксированными требованиями пользователя. Функциональные требования определяют, что именно делает ПО, какие задачи оно решает [14]. Результаты тестирования представлены в таблице 1.

Табл. 1. Результаты функционального тестирования

№ п/ п	Категория набора тестов	Ожидаемый результат	Результат
1	Проверка перемещения персонажа	При нажатии кнопок «W», «A», «S», «D» персонаж перемещается по игровому уровню.	Пройден
2	Стрельба персонажа	При нажатии кнопок со стрелками персонаж стреляет в сторону, в которую указывает стрелка. При зажатии кнопки персонаж стреляет с максимально доступной ему скорострельностью.	Пройден
3	Переход на следующий уровень	После уничтожения всех противников на уровне появляется портал, который ведет в следующую зону. Когда персонаж задевает портал загружается следующий уровень.	Пройден
4	Активация улучшений	Когда персонаж задевает улучшение, то оно применяется к персонажу. Было применено улучшение увеличения количества выстрелов. После его применение персонаж должен стрелять двумя снарядами вместо одного.	Пройден
5	Активация нескольких связанных улучшений	Когда персонаж поднимает несколько связанных улучшений, то последующие улучшения не должны отменять предыдущие. Сначала было применено улучшение увеличения количества выстрелов. После было применено улучшение разрывных снарядов. После этого персонаж должен стрелять двумя снарядами, оба из которых разрываются на несколько своих копий.	Пройден

Вывод

Для реализованного игрового приложения было проведено функциональное тестирование. Все тесты были выполнены успешно, проблем выявлено не было.

ЗАКЛЮЧЕНИЕ

В ходе разработки компьютерной игры в жанре «Roguelite» были проанализированы существующие методы реализации и жанрово близкие приложения.

В ходе разработки системы были решены следующие задачи:

1) проведен обзор аналогов;
2) разработано техническое задание на разработку игры;
3) спроектировано приложение;
4) реализовано приложение;
5) проведено функциональное тестирование.

В будущем планируется провести дальнейшую работу с приложением, в том числе по добавлению новых функций и улучшению качества приложения. Для этого предполагается добавления новых улучшений, системы накладываемых эффектов, новые уровни и противники.

ЛИТЕРАТУРА

1. Официальный сайт игры ADOM. [Электронный ресурс] URL: https://www.adom.de/home/index.html (дата обращения: 23.05.2020).

2. Официальный сайт игры Angband. [Электронный ресурс] URL: https://rephial.org/ (дата обращения: 23.05.2020).

3. Официальный сайт игры Dungeon Crawl Stone Soup. [Электронный ресурс] URL: https://crawl.develz.org/ (дата обращения: 23.05.2020).

4. Официальный сайт игры Enter the Gungeon. [Электронный ресурс] URL: https://dodgeroll.com/gungeon/ (дата обращения: 23.05.2020).

5. Официальный сайт игры FTL: Faster Than Light. [Электронный ресурс] URL: https://subsetgames.com/ftl.html (дата обращения: 23.05.2020).

6. Официальный сайт игры Nuclear Throne. [Электронный ресурс] URL: http://nuclearthrone.com/ (дата обращения: 23.05.2020).

7. Официальный сайт игры Risk of Rain. [Электронный ресурс] URL: https://riskofraingame.com / (дата обращения: 23.05.2020).

8. Официальный сайт игры Rogue Legacy. [Электронный ресурс] URL: http://www.cellardoorgames.com/roguelegacy/ (дата обращения: 23.05.2020).

9. Официальный сайт игры Tales of Maj'Eyal. [Электронный ресурс] URL: https://te4.org/ (дата обращения: 23.05.2020).

10. Официальный сайт игры The Binding of Isaac: Rebirth. [Электронный ресурс] URL: https://bindingofisaac.com/ (дата обращения: 23.05.2020).

11. Официальный сайт Blizzard [Электронный ресурс] URL: https://www.blizzard.com/ (дата обращения: 23.05.2020).

12. Официальный сайт CorelDRAW. [Электронный ресурс] URL: https://www.coreldraw.com/ru/ (дата обращения: 29.05.2020).

13. Официальный сайт Ubisoft [Электронный ресурс] URL: https://www.ubisoft.com/ (дата обращения: 23.05.2020).

14. Тамре Л. Введение в тестирование программного обеспечения. – М.: Вильямс, 2003. –368с.

15. «Яндекс»: оборот игрового рынка в России достиг $2 млрд в 2019 году. [Электронный ресурс] URL: https://app2top.ru/money/yandeks-oborot-igrovogo-ry-nka-v-rossii-dostig-2-mlrd-v-2019-godu-158475.html (дата обращения: 23.05.2020).

16. Godot Engine. [Электронный ресурс] URL: https://godotengine.org/ (дата обращения: 23.05.2020).

17. Indie Game Developers Rise Up. [Электронный ресурс] URL: https://www.forbes.com/2008/11/20/games-indie-developers-tech-ebiz-cx_mji_1120indiegames.html#7c0fefae73a6 (дата обращения: 23.05.2020).

18. Sal Romano. Exit the Gungeon coming to consoles, PC in early 2020. [Электронный ресурс] URL: https://www.gematsu.com/2020/01/exit-the-gungeon-coming-to-consoles-pc-in-early-2020 (дата обращения: 23.05.2020).

19.Unity [Электронный ресурс] URL: https://unity.com/ (дата обращения: 23.05.2020).

20. Unity plans compare [Электронный ресурс] URL: https://store.unity.com/compare-plans (дата обращения: 23.05.2020).

21. Unity UI. [Электронный ресурс] URL: https://docs.unity3d.com/Packages/com.unity.ugui@1.0/manual/index.html (дата обращения: 29.05.2020).

22. Unreal Engine. [Электронный ресурс] URL: https://www.unrealengine.com/ (дата обращения: 23.05.2020).