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, к.т.н.,
доцент кафедры ПМиП

__________ Е.А. Геренштейн

“___”___________ 2020 г.

ДОПУСТИТЬ К ЗАЩИТЕ

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

__________ Л.Б. Соколинский

“___”___________ 2020 г.

Разработка игры в жанре «Аркада» для ОС MS Windows

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА
ЮУрГУ – 02.03.02.2020.308-025.ВКР

Научный руководитель,
к.ф.-м.н., доцент кафедры СП
__________А.В. Геренштейн

Автор работы,
студент группы КЭ-401
__________В.Е. Римач

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-401
Римачу Виктору Евгеньевичу,
обучающемуся по направлению
02.03.02 «Фундаментальная информатика и информационные технологии»

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

Разработка игры в жанре «Аркада» для ОС MS Windows.

2. **Срок сдачи студентом законченной работы:** 9.06.2020.
3. **Исходные данные к работе**

3.1. Официальный курс изучения Unity. URL: https://learn.unity.com.
3.2. Документация по C#. URL: https://docs.microsoft.com/ru-ru/dotnet/csharp.
3.3. Руководство Unity. URL:
https://docs.unity3d.com/ru/current/Manual/index.html.

4. **Перечень подлежащих разработке вопросов**

4.1. Обзор существующих аналогов игрового приложения.
4.2. Анализ программных средств реализации игрового приложения.
4.3. Проектирование игрового приложения.
4.4. Провести тестирование.

5. **Дата выдачи задания:** 03.02.2020.

Научный руководитель
к.ф.-м.н., доцент кафедры СП А.В. Геренштейн

Задание принял к исполнению В.Е. Римач

ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ ... 4
1. АНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ ... 6
1.1. Обзор аналогов ... 6
1.2. Обзор существующих средств для разработки платформы ... 6
2. ТРЕБОВАНИЯ К ПЛАТФОРМЕ ... 8
2.1. Техническое предложение ... 8
2.2. Функциональные требования ... 8
2.3. Нефункциональные требования ... 9
3. ПРОЕКТИРОВАНИЕ ... 10
3.1. Варианты использования системы ... 10
3.2. Файловая структура приложения ... 11
3.3. Диаграммы деятельности ... 12
4.1. Программные средства реализации ... 15
4.2. Реализация главной страницы ... 15
4.3. Реализация сраницы настроек ... 16
4.4. Реализация передвижения игрока ... 19
4.5. Реализация противников ... 20
4.5. Реализация паузы ... 24
4.6. Реализация игрового интерфейса ... 26
4.7. Реализация бонусов ... 27
4.8. Реализация проигрыша игрока ... 29
4.9. Реализация случайной генерации объектов ... 31
5. ТЕСТИРОВАНИЕ ПРИЛОЖЕНИЯ ... 34
5.1. Функциональное тестирование ... 34
ЗАКЛЮЧЕНИЕ ... 36
ЛИТЕРАТУРА ... 37

ВВЕДЕНИЕ

Актуальность темы

На сегодняшний день игровая индустрия имеет огромную аудиторию и занимает большую часть в сфере компьютерных технологий. История компьютерных игр берет свое начало с 50 – ых годов двадцатого века, и в настоящее время, игровая индустрия развивается с огромной скоростью. Ежегодно выпускаются тысячи игр, и некоторые из них приносят своим разработчикам миллионы долларов [1].

Жанр аркадных игр является одним из самых распространенных в игровой индустрии. Термином «Аркада» можно назвать игры, характеризующиеся коротким по времени, но интенсивным игровым процессом [2].

Цель и задачи

Целью данной работы является разработка игры для ОС MS Windows в жанре «Аркада».

Для достижения указанной цели необходимо решить следующие задачи.

1. Провести анализ требований и постановку задачи.
2. Провести обзор программных средств разработки.
3. Провести проектирование архитектуры игры.
4. Реализовать игру в жанре «Аркада» для платформы ОС MS Windows.
5. Протестировать разработанную игру.

Структура и объем работы

Работа состоит из введения, 5 разделов, заключения, списка библиографии. Объем работы составляет 38 страниц, объем библиографии 15 – источников.

Краткое содержание работы

Во введении описаны актуальность исследуемой темы, цели и задачи исследования, структура и объем работы.

В первой главе рассматриваются аналоги разрабатываемого приложения. Выделены их достоинства и недостатки. А также рассматриваются существующие средства для разработки.

Во второй главе описаны требования к разрабатываемой системе.

В третьей главе спроектирована диаграмма вариантов использования, приведена файловая структура приложения и диаграмма деятельности.

В четвертой главе описаны инструментальные средства разработки, взаимодействие компонентов приложения, приведены скриншоты интерфейса.

В пятой главе проведено функциональное тестирование разработанной системы.

В заключении перечислены основные результаты работы.

1. АНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ

В данном разделе был выполнен обзор аналогов приложения.

1.1. Обзор аналогов

В ходе выполнения дипломной работы были проанализированы несколько популярных игр в жанре аркада.

Pacman [3]

Игра, разработанная в 1980 году компанией Namco для аркадных автоматов, в которой управляя Пакманом требуется съесть все точки в лабиринте, избегая привидений.

FlatOut 2 [4]

Видеоигра в жанре аркадного автосимулятора, разработанная кампанией Bugbear Entertainment. Во FlatOut 2 представлены 5 режимов игры: «Карьера», «Заезд», «Одиночная гонка», «Трюки» и «Дерби», а также игра по глобальной или локальной сети с реальными людьми. Главной особенностью этой игры является то, что большинство объектов на трассе разрушаемы.

Quake III Arena [5]

Аркадный 3D шутер от первого лица, разработанный студией id Software и изданный компанией Activision. Цель одиночной игры состоит в прохождении уровней, сражаясь с другими «Гладиаторами». Также в игре присутствует эмуляция мультиплеерас теми же режимами игры: «Каждый сам за себя», «Командный бой», «Один на один», «Захват флага».

1.2. Обзор существующих средств для разработки платформы

На данный момент существует несколько платформ для создания игр для ОС MS Windows, наиболее распространенными являются следующие:

Unreal Engine [6]

С помощью Unreal Engine можно создавать игры для большого количества платформ и операционных систем, таких как Mac OS, Linux, Microsoft Windows, а также консолей Playstaion 3, Playstation 2, Playstation

Portable, Wii, Xbox360 и др.

Плюсы Unreal Engine.

1. Кроссплатформенность.
2. Новые инструменты выходят с каждым обновлением.
3. Большое сообщество разработчиков.

Минусы Unreal Engine.

1. Высокие аппаратные требования.
2. Если игра становиться довольно популярной, то разработчики получают 5% от доходов с игры.

CryEngine [7]

Игровой движок, разработанный немецкой компанией Crytek. CryEngine 3 изначально является кроссплатформенным движком, ориентирован на IBM PC – совместимые компьютеры и игровые консоли Xbox и Playstation.

Unity [8]

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

Вывод

Рассмотрены аналоги разрабатываемого приложения. По рассмотренным аналогам можно сделать вывод, что аркадами могут быть игры разных типов, однако главное отличие аркад в простом и понятном управлении, с ограниченным количеством действий. Также были рассмотрены популярные платформы для создания игр для ОС MS Windows и после их анализа для разработки приложения была выбрана среда разработки Unity.

2. ТРЕБОВАНИЯ К ПЛАТФОРМЕ

2.1. Техническое предложение

Разрабатываемое приложение – игра в жанре «аркада», являющаяся аркадным шутером от первого лица, в котором игроку предстоит встретиться с бесконечно генерирующимися противниками, на карте будут случайно распределяться укрытия, а также будут случайно появляться различные бонусы.

Цель игры

Основной целью игрока является набрать наибольшее возможное количество очков, избегая снарядов противников.

2.2. Функциональные требования

Разрабатываемое приложение должно удовлетворять следующим функциональным требованиям.

1. При запуске приложения пользователю должно высветиться меню, в котором можно начать игру, выйти из игры, или зайти в меню настройки приложения.
2. Система должна предоставлять польователю возможность поставить на паузу во время игры.
3. Система должна предоставлять пользователю возможность выйти в главное меню в любое время.
4. Система должна предоставлять пользователю возможность управлять персонажем.
5. Система должна во время игры выводить на экран количество очков, заработанных игроком.
6. Система должна во время игры выводить на экран количество жизней игрока.
7. Система должна выводить тип бонуса и время длительности этого бонуса при его подборе.

8. Система должна во время игры выводить на экран количество оставшихся у игрока патронов.

9. Система должна предоставлять пользователю возможность стрелять.

10. Система должна предоставлять пользователю возможность подбирать различные усиления.

11. Система должна предоставлять пользователю возможность перезапустить игру после смерти персонажа.

12. Система должна предоставлять пользователю возможность менять настройки игры.

2.3. Нефункциональные требования

Разрабатываемое приложение должно удовлетворять следующим нефункциональным требованиям.

1. Приложение должно быть доступно на платформе ОС MS Windows.

2. Приложение должно быть реализовано на платформе Unity при помощи языка программирования C#.

Вывод

В ходе анализа требований к платформе были выявлены функциональные и нефункциональные требования.

3. ПРОЕКТИРОВАНИЕ

3.1. Варианты использования системы

При проектировании и реализации разрабатываемого приложения была реализована диаграмма деятельности [13], представленная на рисунке 1.

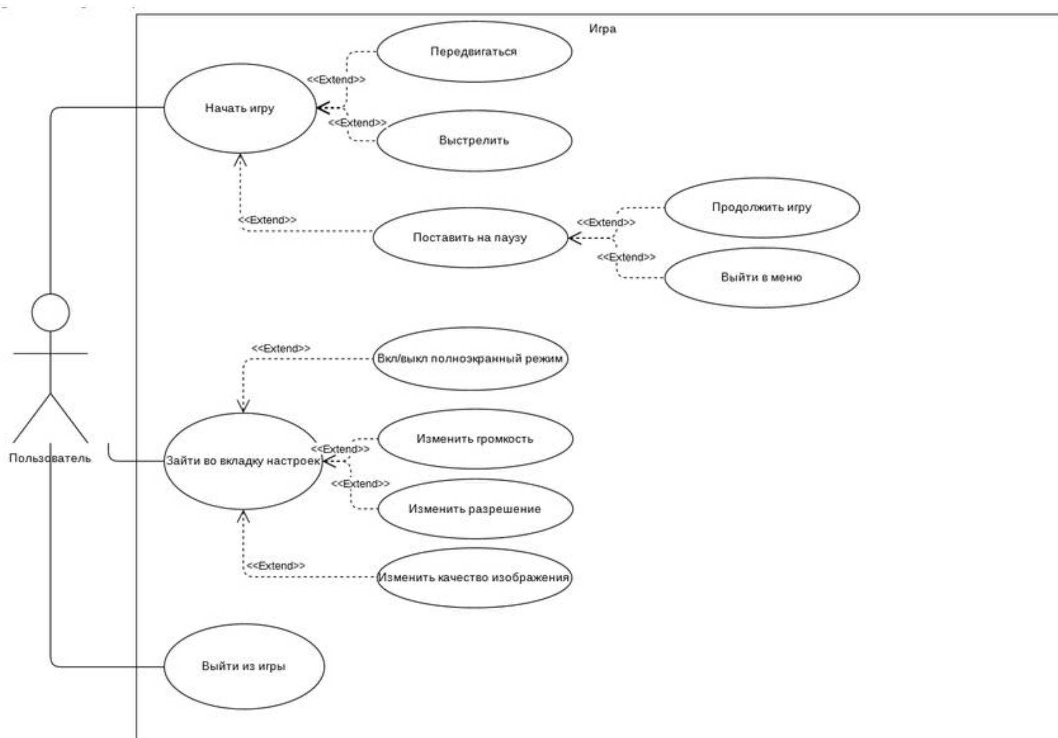

Рис. 1. Диаграмма вариантов использования

Для данной диаграммы основным актером, взаимодействующим с системой, является Пользователь. Для данного актера были определены следующие варианты использования.

1. *Начать игру* – запустить игру при нажатии кнопки «Play» в главном меню программы.

2. *Передвигаться* – перемещать камеру персонажа при нажатии определенных кнопок во время игры.

3. *Выстрелить* – спавн летящего вперед игрового объекта при нажатии на левую кнопку мыши во время игры.

4. *Поставить на паузу* – остановка игрового времени и вывод на экран кнопок «Return» и «Menu.

5. *Продолжить игру* – возможность убрать паузу и вернуться в игру.

6. *Выйти в меню* – возможность выйти в главное меню во время игры.

7. *Зайти во вкадку настроек* – в главном меню программы нажать кнопку «Settings», после перехода на экране будут выведены доступые для изменения настройки.

8. *Вкл/выкл полноэкранный режим* – изменить значение переключателя «Fullscreen» во вкладке «Settings», тем самым меняя режим игры между полноэкранным и оконным.

9. *Изменить громкость* – изменить значение на слайдере «Volume» во вкладке «Settings», тем самым меняя громкость игры.

10. *Изменить разрешение* – выбрать значение во вложенном списке «Resolution» во вкладке «Settings», тем самым меняя разрешение в игре.

11. *Изменить качество изображения* – выбрать значение во вложенном списке «Quality» во вкладке «Settings», тем самым меняя качество изображения в игре.

12. *Выйти из игры* – возможность выйти из игры при нажатии на кнопку «Exit» в главном меню.

3.2. Файловая структура приложения

Файловая структура приложения состоит из каталогов, содержащих файлы, используемые в программе. Файловая структура приложения представлена на рисунке 2.

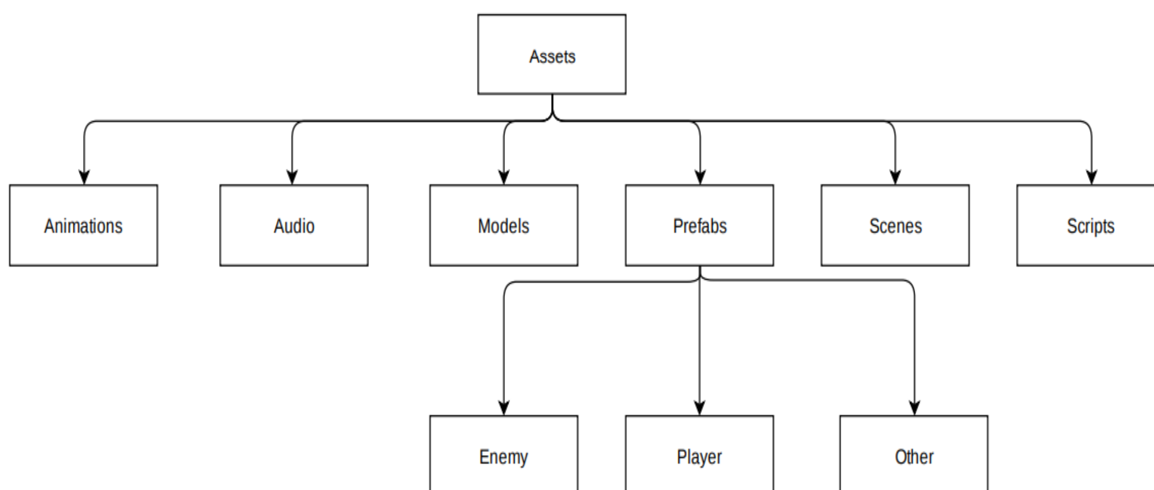

Рис. 2. Файловая структура приложения

Директория *Animations* содержит в себе анимации игровых объектов.

Директория *Audio* содержит файлы музыкального сопровождения.

Директория *Models* содержит в себе 3D модели игровых объектов.

Директория *Prefabs* содержит готовые шаблоны игровых объектов.

Директория *Scenes* содержит игровые сцены.

Директория *Scripts* содержит в себе скрипты, используемые в игровом приложении.

3.3. Диаграммы деятельности

На основе требований к системе были разработаны диаграммы деятельности (activity diagram) [14]. Эти диаграммы показывают, как происходит процесс взаимодействия пользователя с системой.

Диаграмма деятельности перехода в меню настроек и взаимодействия с ним представлена на рисунке 3.

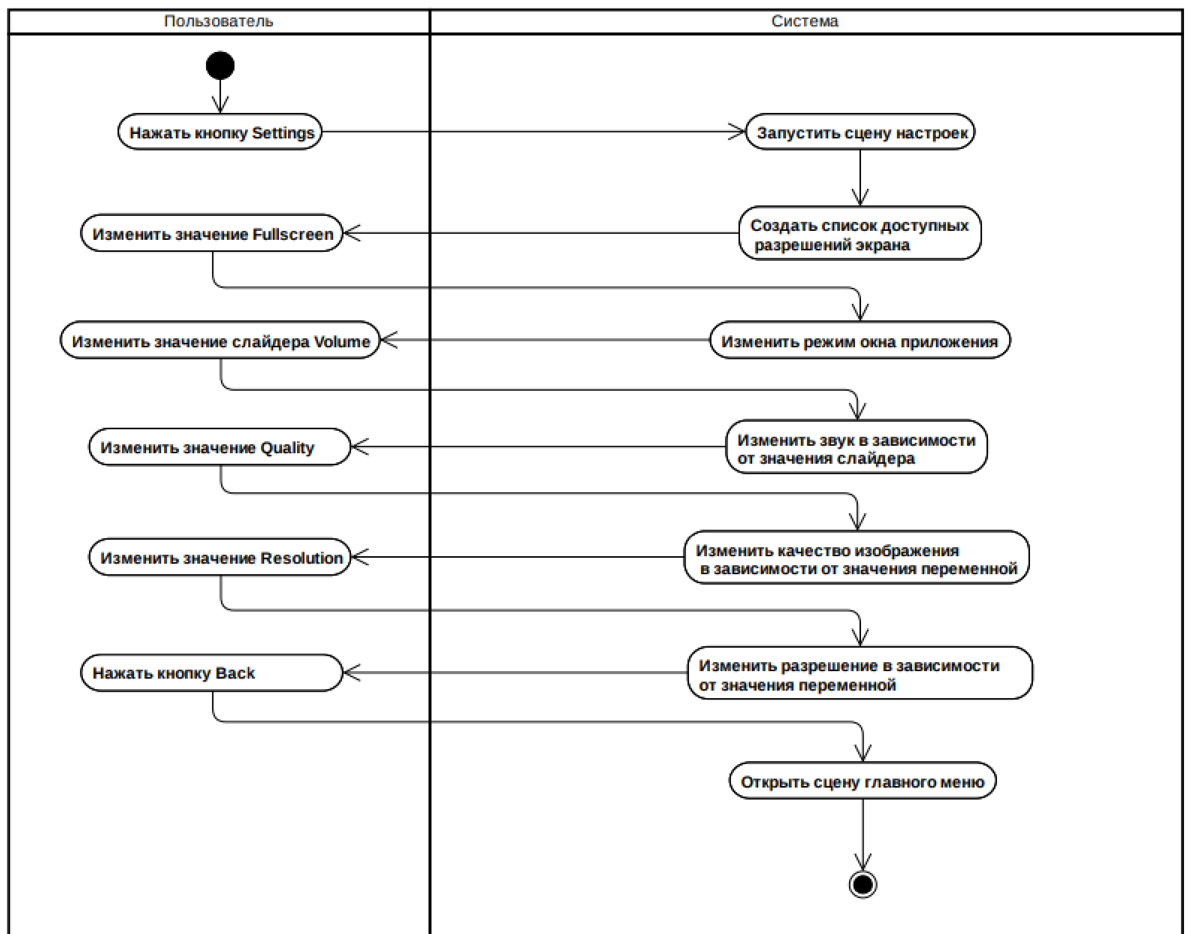

Рис.3. Диаграмма деятельности перехода на страницу настроек и взаимодействие с ней

Диаграмма деятельности поднятия бонуса представлена на рисунке 4.

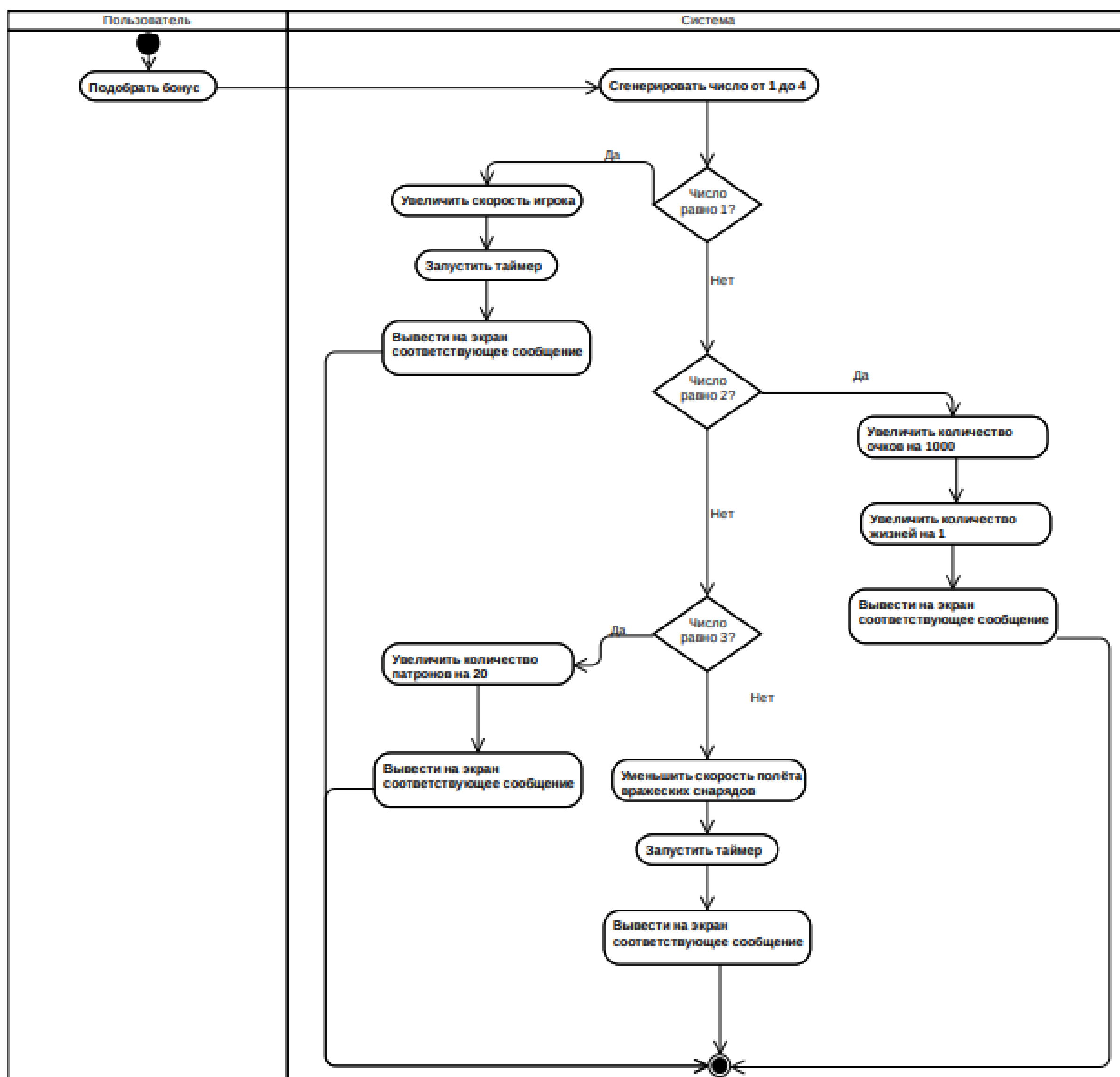

Рис.4. Диаграмма деятельности поднятия бонуса

Вывод

В главе проектирование с учетом назначения системы и обзора готовых решений для ее реализации, проведенных в первой главе, а так же с учетом функциональных и нефункциональных требований, приведенных во второй главе, представлены диаграмма вариантов использования, а также файловая структура приложения.

4. ПРОГРАММНАЯ РЕАЛИЗАЦИЯ

4.1. Программные средства реализации

Для разработки игры была выбрана платформа Unity. Для программной реализации был использован язык программирования C#. Разработка велась в среде Microsoft Visual Studio 2019 [15].

Для разработки моделей были использованы программы Blender [9] и Adobe Fuse CC [10]. Также был использован пакет моделей FireFireWeapon [11].

4.2. Реализация главной страницы

Для реализации главной страницы приложения была создана сцена *Menu*. На данной сцене был создан компонент Canvas, на котором были созданы 3 кнопки: для начала игры, для перехода в меню настроек и для выхода из игры. Также, для управления этими кнопками был создзан класс *MenuControl.* В данном классе были реализованы два метода: *PlayPressed* и *ExitPressed*. Метод *PlayPressed* предназначен для перехода на сцену с игрой по нажатию кнопки. А метод *ExitPressed* предназначен для выхода на рабочий стол при нажатии кнопки.

Интерфейс главного меню приведен на рисунке 5.

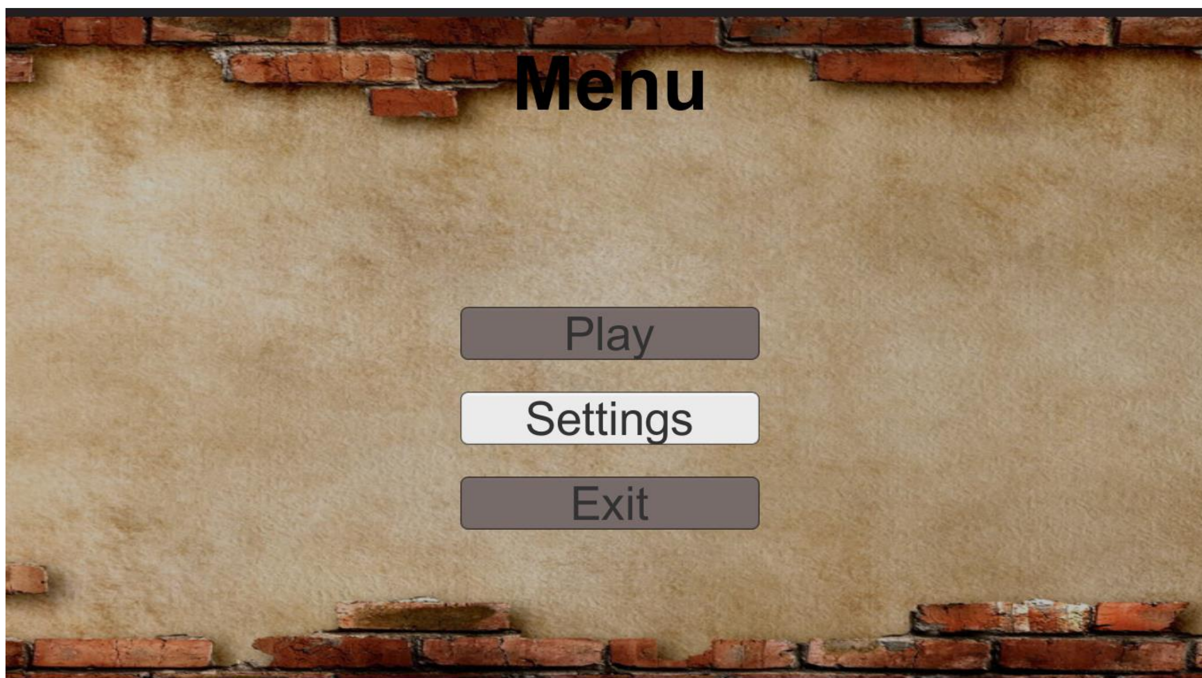

Рис. 5. Интерфейс главного меню

Реализация класса *MenuControl* приведен на рисунке 6.

```
public class MenuControl : MonoBehaviour
{
    public void PlayPressed()
    {
        Cursor.visible = false;
        SceneManager.LoadScene("Test");
    }
    public void ExitPressed()
    {
        Application.Quit();
    }

}
```

Рис. 6. Реализация класса *MenuControl*

4.3. Реализация сраницы настроек

Для реализации страницы настроек был создан компонент типа Canvas на сцене Menu, который изначально отключен. При нажатии на кнопку «Settings» этот компонент включается. На данном компоненте были добавлены переключатель для изменения режима окна, слайдер для управления громкостью звука, 2 скрытых списка для смены качества изображения и разрешения приложения и кнопка для возврата в главное

меню. Для управления компонентами был создан класс *Settings*. В данном классе были реализованы шесть методов: *Awake*, *FullScrToggle*, *AudioVolume*, *Quality*, *Resolution*, *OnEnable*. Метод *Awake* это стандартный метод Unity, который выполняется при загрузке скрипта. В данном классе он отвечает за создание списка поддерживаемых подключенным монитором разрешений.

Интерфейс страницы настроек приведен на рисунке 7.

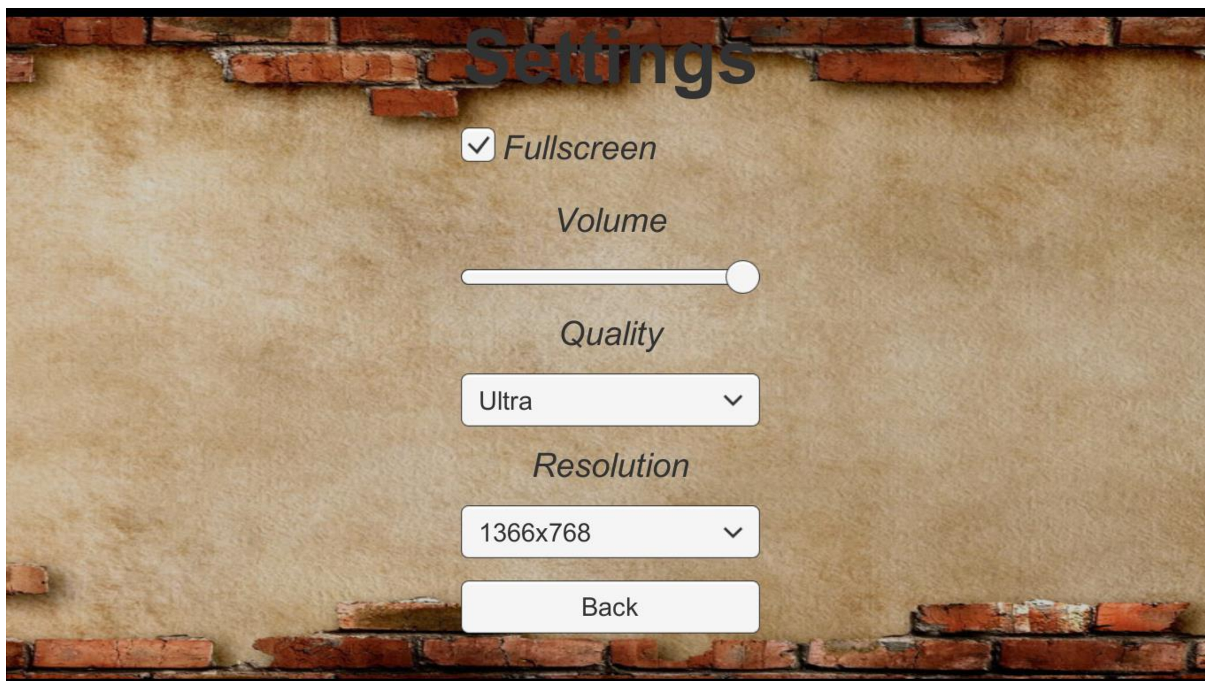

Рис. 7. Интерфейс страницы настроек

Реализация метода *Awake* приведен на рисунке 8.

```
public void Awake()
    {
        resolutions = new List<string>();
        rsl = Screen.resolutions;
        foreach (var i in rsl)
        {
            resolutions.Add(i.width + "x" + i.height);
        }
        dropdown.ClearOptions();
        dropdown.AddOptions(resolutions);
    }
```

Рис. 8. Реализация метода *Awake*

Метод FullScrToggle, который вызывается при смене значения переключателя, меняет значение встроенной логической переменной Unity, отвечающей за полноэкранный режим.

Реализация метода *FullScrToggle* представлен на рисунке 9.

```
public void FullScreenToggle()
{

    isFullScreen = !isFullScreen;
    Screen.fullScreen = isFullScreen;
}
```

Рис. 9. Реализация метода *FullScrToggle*

Метод *AudioVolume* при смене значения слайдера меняет громкость игры.

Реализация метода *AudioVolume* представлен на рисунке 10.

```
public void AudioVolume(float sliderValue)
{
    audio.SetFloat("masterVolume", sliderValue);
}
```

Рис. 10. Реализация метода *AudioVolume*

Методы *Quality* и *Resolution* получают номер строки, которую выбрал игрок в соответствующем выпадающем списке, и меняют значение качества изображения и разрешения на соответствующие.

Реализация методов *Quality* и *Resolution* представлены на рисунке 11.

```
public void Quality(int q)
{
    QualitySettings.SetQualityLevel(q);
}
public void Resolution(int r)
{
    Screen.SetResolution(rsl[r].width, rsl[r].height, isFullScreen);
}
```

Рис. 11. Реализация методов *Quality* и *Resolution*

Стандартный метод Unity *OnEnable,* который вызывается вызывается каждый раз, когда компонент становится активным, отвечает за значение переключателя полноэкранного режима.

Реализация метода *OnEnable* представлены на рисунке 12.

```
public void Quality(int q)
{
    QualitySettings.SetQualityLevel(q);
}
public void Resolution(int r)
{
    Screen.SetResolution(rsl[r].width, rsl[r].height, isFullScreen);
}
```

Рис. 12. Реализация метода *OnEnable*

4.4. Реализация передвижения игрока

Для реализации передвижения игрока был создан класс *PController,* в котором были реализованы 4 метода: *Start*, *OnCollisionStay*, *OnCollisionExit*, *FixedUpdate*. Метод старт вызывается перед первым вызовом метода *Update* и отвеачет за начальное направление игрока.

Реализация метода *Start* представлены на рисунке 13.

```
void Start()
{
    StartingRotation = transform.rotation;
}
```

Рис. 13. Реализация метода *Start*

Методы *OnCollisionStay* и *OnCollisionExit* отвечают за проверку того, находится ли игрок на земле.

Реализация методов *OnCollisionStay* и *OnCollisionExit* представлены на рисунке 14.

```
void OnCollisionStay(Collision other)
{
    if (other.gameObject.tag == "ground")
    {
        isGround = true;
    }
}
void OnCollisionExit(Collision other)
{
    if (other.gameObject.tag == "ground")
    {
        isGround = false;
    }
}
```

Рис. 14. Реализация методов *OnCollisionStay* и *OnCollisionExit*

Метод *FixedUpdate* вызывается с частотой фиксированных кадров и служит для смены угла просмотра камеры при перемещении мышки, передвижения персонажа при нажатии клавиш передвижения, а также для реализации прыжка персонажа по нажатию соответствующей кнопки, если персонаж находится на земле.

Реализация метода *FixedUpdate* представлены на рисунке 15.

```
void FixedUpdate()
{
    RotHor += Input.GetAxis("Mouse X") * mousesensitivity;
    RotVer -= Input.GetAxis("Mouse Y") * mousesensitivity;
    RotVer = Mathf.Clamp(RotVer, -60, 60);
    Quaternion RotX = Quaternion.AngleAxis(RotVer, Vector3.right);
    Quaternion RotY = Quaternion.AngleAxis(RotHor, Vector3.up);
    cam.transform.rotation = StartingRotation * transform.rotation * RotX;
    transform.rotation = StartingRotation * RotY;
    if (isGround)
    {
        Ver = Input.GetAxis("Vertical") * Time.deltaTime * Speed;
        Hor = Input.GetAxis("Horizontal") * Time.deltaTime * Speed;
        Jump = Input.GetAxis("Jump") * Time.deltaTime * JumpSpeed;
        GetComponent<Rigidbody>().AddForce(transform.up * Jump,
ForceMode.Impulse);
    }
    transform.Translate(new Vector3(Hor, 0, Ver));
}
```

Рис. 15. Реализация метода *FixedUpdate*

4.5. Реализация противников

Модель противника была реализована с помощью программы Adobe Fuse CC. Модель противника представлена на рисунке 16.

Рис. 16. Модель противника

Модель оружия противника была взята из пакета FirefireWeapon. Модель оружия противника представлена на рисунке 17.

Рис. 17. Модель оружия противника

Для реализации поведения противника была добавлена анимация ходьбы и добавлен компонент Nav Mesh Agent, отвечающий за передвижение противника. Также был создан объект Ragdoll для реализации

смерти противника и 4 класса, отвечающие за поведение противника: *EnemyDeathScr, EnemyTriggerScr, EnemyWalkScr, LookScr*. Класс *LookScr* отвечает за то, чтобы противник поворачивался к игроку, в зависимости от местоположения игрока, и включает в себя один метод *LateUpdate.* Этот метод вызывается каждый кадр, если объект включен.

Реализация класса *LookScr* представлена на рисунке 18.

```
public class LookScr : MonoBehaviour
{
    public Transform target;
    void LateUpdate()
    {
        transform.LookAt(target.transform.position);
    }
}
```

Рис. 18. Реализация класса *LookScr*

Класс *EnemyWalkScr* предназначен для задания противнику точки, к которой он будет двигаться и состоит из двух методов: *Start, Update.* В методе *Start* мы задаем переменной значение компонена Nav Mesh Agent, а в методе *Update* передается позиция объекта, за которым будет следовать противник.

Реализация класса *EnemyWalkScr* представлена на рисунке 19.

```
public class EnemyWalkScr : MonoBehaviour
{
    public Transform player;
    NavMeshAgent agent;
    void Start()
    {
        agent = GetComponent<NavMeshAgent>();
    }
    void Update()
    {
        agent.SetDestination(player.position);
    }
}
```

Рис. 19. Реализация класса *EnemyWalkScr*

Класс *EnemyTriggerScr* отвечает за то, как себя будет вести противник, если рядом будет игрок, а также создает объект пули возле модели оружия, и придает этому объекту скорость полета и воспроизводит звук выстрела.

Также при появлении рядом игрока, этот класс отключает компонент Nav Mesh Agent и отключает анимацию ходьбы. Класс включает в себя 3 метода: *OnTriggerEnter, OnTriggerStay, OnTriggerExit.* Методы *OnTriggeEnter* и *OnTriggerExit* отвечают за включение и отключение перемещения противника и анимации ходьбы.

Реализация методов *OnTriggerEnter* и *OnTriggerExit* представлена на рисунке 20.

```
void OnTriggerEnter(Collider other)
{
    if (other.tag == "Player")
    {
        Enemy.GetComponent<Animator>().enabled = false;
        Enemy.GetComponent<NavMeshAgent>().enabled = false;
    }
}
void OnTriggerExit(Collider other)
{
    if (other.tag == "Player")
    {
        Enemy.GetComponent<NavMeshAgent>().enabled = true;
        Enemy.GetComponent<Animator>().enabled = true;
    }
}
```

Рис. 20. Реализация методов *OnTriggerEnter* и *OnTriggerExit*

Метод *OnTriggerStay,* если рядом с противником есть игрок, с каким – то промежутком спавнит объект пули и придает ему ускорение.

Реализация метода *OnTriggerStay* представлена на рисунке 21.

```
void OnTriggerStay(Collider other)
{
    if (other.tag =="Player")
    {
        bullettimer -= Time.deltaTime;
        if (bullettimer <= 0)
        {
            bullettimer = newbullettimer;
            Transform BulletInstance = (Transform)Instantiate(Bullet,
GameObject.Find("EnemyMuzzle").transform.position, Quaternion.identity);

BulletInstance.GetComponent<Rigidbody>().AddForce(transform.forward *
BulletSpeed);
            GetComponent<AudioSource>().PlayOneShot(Shot);
        }
    }
}
```

Рис. 21. Реализация метода *OnTriggerStay*

Класс *EnemyDeathScr* отвечает за смерть противника и состоит из одного метода: *OnTriggerEnter*. Этот метод, при попадании в противника объекта с тэгом «Bullet» уменьшает количество жизней противника на 1, и если у противника после этого не осталось жизней, то метод уничтожает противника, и на его месте появляется ragdoll, который служит анимацией смерти, а также уменьшает счетчик количества противников на 1 и увеличивает количество очков игрока на 100, и если количество становится кратно 1000, то увеличивает количество жизней игрока на 1.

Реализация класса *EnemyDeathScr* представлена на рисунке 22.

```
public class EnemyDeathScr : MonoBehaviour
{
    public GameObject Enemy;
    public GameObject Ragdoll;
    public int Lives = 1;
    private void OnTriggerEnter(Collider other)
    {
        if (other.tag == "Bullet")
        {
            Lives -= 1;
            if (Lives < 1)
            {
                Destroy(Enemy);
                Ragdoll.SetActive(true);
                Instantiate(Ragdoll, transform.position,
transform.rotation);
                SpawnScr.Count -= 1;
                Pointsscr.curScore += 100;
                if (Pointsscr.curScore % 1000 == 0)
                {
                    Pointsscr.curLives += 1;
                }
            }
        }
    }

}
```

Рис. 22. Реализация класса *EnemyDeathScr*

4.5. Реализация паузы

Для реализации паузы был создан класс *PauseScr*, который при нажатии на кнопку паузы останавливает время игры и выводит на экран 2 кнопки: для возврата в игру и для выхода в главное меню. В данном классе 2 метода: *Update* и *OnGUI*. Метод *Update* проверяет была ли нажата кнопка

открытия паузы, и если была, то стоит ли игра на паузе в данный момент. Если игра стоит на паузе и нажимается нопка паузы, то пауза откючается.

Реализация метода *Update* представлена на рисунке 23.

```
void Update()
{
    Time.timeScale = timer;

    if (Input.GetKeyDown(KeyCode.Escape)  && ispause == false)
    {
        ispause = true;
    }
    else if (Input.GetKeyDown(KeyCode.Escape) && ispause == true)
    {
        ispause = false;
    }
    if (ispause == true)
    {
        timer = 0;
        guipause = true;
    }
    else if (ispause == false)
    {
        timer = 1f;
        guipause = false;
    }
}
```

Рис. 23. Реализация метода *Update*

Метод *OnGUI*, если стоит пауза выводит на экран 2 кнопки, первая из которых снимает паузу, если на нее нажать, а вторая загружает сцену главного меню.

Реализация метода *OnGUI* представлена на рисунке 24.

```
public void OnGUI()
{
    if (guipause == true)

    {
        Cursor.visible = true;
        if (GUI.Button(new Rect((float)(Screen.width / 2) - 100,
(float)(Screen.height / 2) - 100f, 150f, 45f), "Return"))
        {
            ispause = false;
            timer = 0;
            Cursor.visible = false;
        }

        if (GUI.Button(new Rect((float)(Screen.width / 2) - 100,
(float)(Screen.height / 2) + 100f, 150f, 45f), "Menu"))
        {
            ispause = false;
            timer = 0;
            SceneManager.LoadScene("Menu");
        }
    }
}
```

Рис. 24. Реализация метода *OnGUI*

4.6. Реализация игрового интерфейса

Для реализации игрового интерфейса был создан класс *PointsScr,* в котором реализован один метод *OnGUI,* который выводит на экран количество жизней, очков, патронов и подобранные бонусы.

Реализация метода *OnGUI* представлена на рисунке 25.

```
void OnGUI()
{
    if (PickupScr.i == 1 && PickupScr.timer > 0)
    {
        GUI.Label(new Rect((float)(Screen.width) - 100, 10, 100, 100),
"Speed Bonus: " + PickupScr.timer);
    }
    if (PickupScr.i == 2 && PickupScr.timer > 0)
    {
        GUI.Label(new Rect((float)(Screen.width) - 100, 10, 100, 100),
"+1000 Points");
    }
    if (PickupScr.i == 3 && PickupScr.timer > 0)
    {
        GUI.Label(new Rect((float)(Screen.width) - 100, 10, 100, 100),
"+20 Bullets");
    }
    if (PickupScr.i == 4 && PickupScr.timer > 0)
    {
        GUI.Label(new Rect((float)(Screen.width) - 100, 10, 100, 100),
"Slow Bonus: " + PickupScr.timer);
    }
    GUI.Label(new Rect(10, 10, 100, 100), "Lives: " + curLives);
    GUI.Label(new Rect(10, (float)(Screen.height) - 20, 100, 100), "Ammo:
" + WeaponScr.BulletAmount + "/10");
    GUI.Label(new Rect((float)(Screen.width) - 100, 10, 100, 100), "Score:
" + curScore);
}
```

Рис. 25. Реализация метода *OnGUI*

4.7. Реализация бонусов

Для реализации бонусов был создан класс *PickupScr*, в котором были реализованы 2 метода: *OnTriggerEnter, Update.* Метод *OnTriggerEnter* проверяет, является ли объект, который подошел к модели бонуса игроком, и если является, то уничтожается объект бонуса, генерируется случайное число от 1 до 4, и в зависимости от сгенерированного числа игроку дается бонус. Если сгенерировалось число 1, то увеличивается скорость персонажа на определенное количество времени, если 2, количество очков игрока увеличивается на тысячу и количество жизней увеличивается на 1, если сгенерироавалось 3, то количество патронов игрока увеличивается на 20, и если генерируется 4, то скорость полета снарядов противника замедляется на определенное время.

Реализация метода *OnTriggerEnter* представлена на рисунке 26.

```
private void OnTriggerEnter(Collider other)
{
    if (other.tag == "Player")
    {
        Destroy(gameObject);
        i = Random.Range(1, 5);
        if (i == 1)
        {
            timer = 10;
            PController.Speed = 15;
        }
        else if (i == 2)
        {
            timer = 3;
            Pointsscr.curScore += 1000;
            Pointsscr.curLives += 1;
        }
        else if (i == 3)
        {
            timer = 3;
            WeaponScr.BulletAmount += 20;
        }
        else
        {
            timer = 20;
            EnemyTriggerScr.BulletSpeed = 300;
        }
    }
}
```

Рис. 26 Реализация метода *OnTriggerEnter*

Метод *Update* служит для того, чтобы по окончании таймера забирались временные бонусы, и если бонус долго не подбирался игроком, то он уничтожается.

Реализация метода *Update* представлена на рисунке 27.

```
void Update()
{
    if (destime > 0)
    {
        destime -= Time.deltaTime;
    }
    if (destime <=0)
    {
        Destroy(gameObject);
    }
    if (timer > 0)
    {
        timer -= Time.deltaTime;
    }
    if (timer <= 0 && i != 0)
    {
        i = 0;
        EnemyTriggerScr.BulletSpeed = 750;
        PController.Speed = 10;
    }
}
```

Рис. 27 Реализация метода *Update*

4.8. Реализация проигрыша игрока

Для реализации проигрыша персонажа игрока была создана сцена экрана проигрыша с двумя кнопками и 2 класса: *DeathScr* и *DeathSceneScr.* В классе *DeathScr* был реализован один метод *OnTriggerEnter*, в котором проверяется, с каким объектом столкнулся игрок, и если у этого объекта тэг «Kill», то количество жизней уменьшается на 1. Когда количество жизней опускается до 0, то персонаж игрока отключается, и загружается сцена экрана проигрыша.

Реализация метода *OnTriggerEnter* представлена на рисунке 28.

```
private void OnTriggerEnter(Collider other)
{
    if (other.tag == "Kill")
    {
        Pointsscr.curLives -= 1;
        if (Pointsscr.curLives < 1)
        {
            Enemy.SetActive(false);
            SceneManager.LoadScene("Death");
            Cursor.visible = true;
        }
    }
}
```

Рис. 28 Реализация метода *OnTriggerEnter*

Интерфейс экрана проигрыша приведен на рисунке 29.

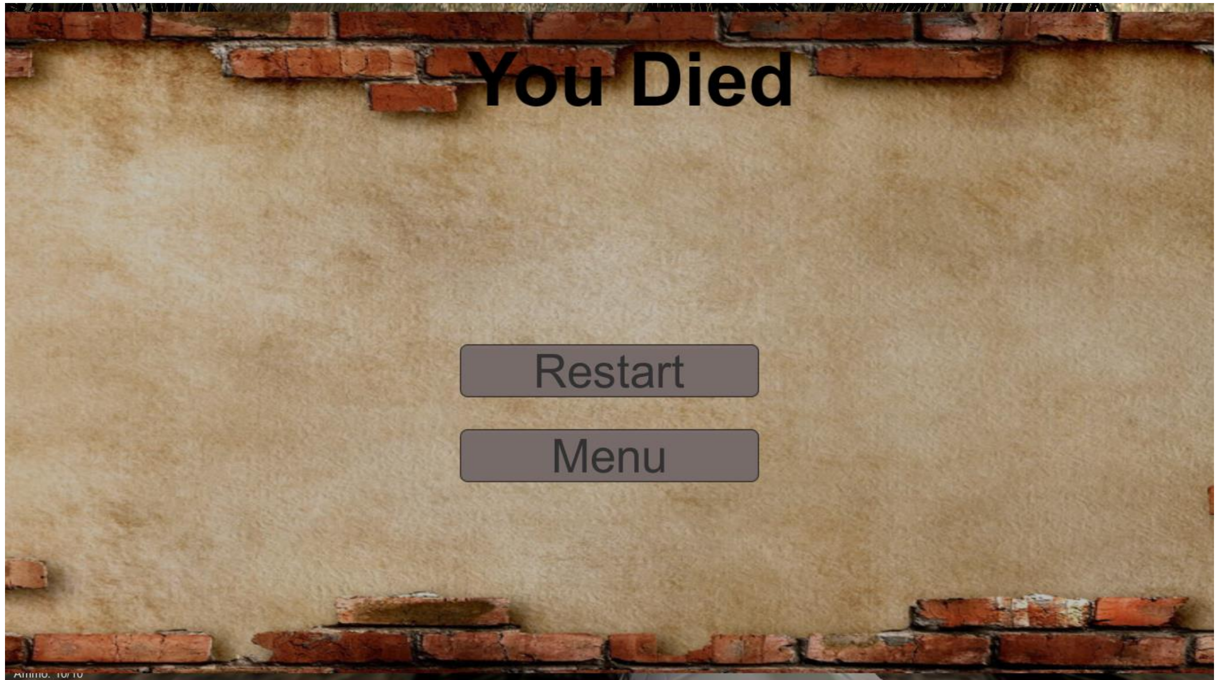

Рис. 29. Интерфейс экрана проигрыша

Класс *DeathSceneScr* отвечает за то, что будут делать кнопки на экране проигрыша. В данном классе реализованы 2 метода: *PlayPressed* и *ExitPressed.* Метод *PlayPressed* при нажатии на кнопку загружает сцену уровня заново, а метод *ExitPressed* при нажатии на кнопку загружает сцену главного меню.

Реализация класса *DeathSceneScr* представлена на рисунке 30.

```
public class DeathSceneScr : MonoBehaviour
{
    public void PlayPressed()
    {
        Cursor.visible = false;
        SceneManager.LoadScene("Test");
    }
    public void ExitPressed()
    {
        SceneManager.LoadScene("Menu");
    }
}
```

Рис. 30 Реализация класса *DeathSceneScr*

4.9. Реализация случайной генерации объектов

Для реализации случайной генерации объектов был создан класс *SpawnScr.* Также в программе Blender были сделаны модели ящиков, чтобы генерировать препятствия, и создан пустой объект, вокруг которого генерировались объекты. В классе *SpawnScr* были реализованы два метода: *Start* и *Update.* Метод *Start* отвечает за генерацию препятсвий при загрузке уровня. Этот метод берет 2 случайных значения, прибавляет их к координатам пустого объекта и генерирует на полученных координатах нужный объект.

Реализация метода *Start* представлена на рисунке 31.

```
void Start()
{
    SpawnTimer = time;
    timerbuff = bufftimer;
    for (int i = 1; i < 16; i++)
    {
        numb = Random.Range(1, 4);
        if (numb == 1)
        {
            int addXPos = Random.Range(-80, 80);
            int addZPos = Random.Range(-80, 80);
            Vector3 spawnPos = transform.position + new Vector3(addXPos,
0, addZPos);
            Instantiate(Crate1, spawnPos, Quaternion.identity);
        }
        if (numb == 2)
        {
            int addXPos = Random.Range(-80, 80);
            int addZPos = Random.Range(-80, 80);
            Vector3 spawnPos = transform.position + new Vector3(addXPos,
0, addZPos);
            Instantiate(Crate2, spawnPos, Quaternion.identity);
        }
        if (numb == 3)
        {
            int addXPos = Random.Range(-80, 80);
            int addZPos = Random.Range(-80, 80);
            Vector3 spawnPos = transform.position + new Vector3(addXPos,
0, addZPos);
            Instantiate(Crate3, spawnPos, Quaternion.identity);
        }
    }
}
```

Рис. 31 Реализация класса *Start*

Метод *Update* отвечает за генерацию противников и бонусов после определенного промежутка времени.

Реализация метода *Update* представлена на рисунке 32.

```
void Update()
{
    if (Count < 15)
    {
        SpawnTimer -= Time.deltaTime;
    }
    if (SpawnTimer <= 0)
    {
        int addXPos = Random.Range(-80, 80);
        int addZPos = Random.Range(-80, 80);
        Vector3 spawnPos = transform.position + new Vector3(addXPos, 0,
addZPos);
        Instantiate(Enemy, spawnPos, Quaternion.identity);
        SpawnTimer = time;
        Count += 1;
    }
    timerbuff -= Time.deltaTime;
    if (timerbuff <= 0)
    {
        int addXPos = Random.Range(-50, 50);
        int addZPos = Random.Range(-50, 50);
        Vector3 spawnPos = transform.position + new Vector3(addXPos, 0,
addZPos);
        Instantiate(Buff, spawnPos, Quaternion.identity);
        timerbuff = bufftimer;
    }
}
```

Рис. 32 Реализация класса *Update*

Модели ящиков представлены на рисунке 33.

Рис. 33. Модели ящиков

Вывод

В соответствии с задачами работы была реализована игра для ОС MS Windows. Были реализованы главное меню, страница настроек игры, передвижение персонажа, поведение противников, пауза, игровой интерфейс, бонусы, проигрыш игрока и случайная генерация объектов.

5. ТЕСТИРОВАНИЕ ПРИЛОЖЕНИЯ

Тестирование программного обеспечения [12] – проверка соответствия между реальным и ожидаемым поведением программы, осуществляемая на конечном наборе тестов, выбранном определенным образом. В более широком смысле тестирование – это одна из техник контроля качества, включающая в себя активности по планированию работ, проектированию тестов, выполнению тестирования и анализу полученных результатов.

5.1. Функциональное тестирование

Функциональное тестирование является одним из ключевых видов тестирования, задача которого – установить соответствие разработанного программного обеспечения исходным функциональным требованиям заказчика. Проведение функционального тестирования позволяет проверить способность информационной системы в определенных условиях решать задачи, нужные пользователям. Функциональное тестирование системы представлено в таблице 1.

Табл. 1. Функциональное тестирование системы

№	Действие	Ожидаемый результат	Результат теста
1	Пользователь нажимает кнопу «Play» в главном меню	Игра запускается	Пройден
2	Пользователь нажимает кнопку «Settings» в главном меню	Открывается меню настроек	Пройден
3	Пользователь нажимает кнопку «Exit» в главном меню	Игра закрывается	Пройден
4	Пользователь нажимает кнопку паузы во время игры	Игра ставится на паузу, и открывается меню паузы	Пройден

Окончание табл. 1

№	Действие	Ожидаемый результат	Результат теста
5	Пользователь нажимает кнопку «Продолжить» во время паузы	Игра продолжается	Пройден
6	Пользоватеь нажимает кнопку «В меню» во время паузы	Открывается главное меню	Пройден
7	Пользователь нажимает на кнопки управления персонажем	Персонаж передвигается в зависимости от нажатых кнопок	Пройден
8	Пользователь подбирает бонус	Игроку дается случайный бонус и выводоится соответствующее сообщение	Пройден
9	Пользователь нажимает на кнопку выстрела	Воспроизводится звук выстрела и создается объект пули, которому придается ускорение	Пройден
10	Пользователь меняет показатель значения «Fullscreen» в настройках	Режим окна меняется в зависимости от установленного значения	Пройден
11	Пользователь меняет показатель значения «Volume» в настройках	Громкость звука меняется в зависимости от установленного значения	Пройден
12	Пользователь меняет показатель значения «Quality» в настройках	Настройки графики меняются в зависимости от установленного значения	Пройден
13	Пользователь меняет показатель значения «Resolution» в настройках	Разрешение игры меняется в зависимости от установленного значения	Пройден
14	Пользователь нажимает на кнопку «Back» в разделе настроек	Открывается главное меню	Пройден
15	Пользователь нажимает на кнопку «Restart» в меню проигрыша	Игра начинается сначала	Пройден

ЗАКЛЮЧЕНИЕ

В ходе проделанной работы были решены следующие задачи.

1. Выполнен анализ предметной области и произведен обзор существующих решений.
2. Произведено проектирование приложения.
3. Реализована игра в жанре «Аркада» для ОС MS Windows.
4. Протестировано приложение.

ЛИТЕРАТУРА

1. Статья об истории развития компьютерных игр. [Электронный ресурс] URL: https://stepgames.ru/blog/istoriya-kompyuternyh-igr (Дата обращения: 01.06.2020).

2. Rouse, Richard. Game Design: Theory & Practice, 2.//Los Rios Boulevard, Plano, Texas, USA : Wordware Publishing, 2004. — 698 с.

3. Pacman. Официальный сайт. [Электронный ресурс] URL: https://www.pacman.com/en/ (Дата обращения: 01.06.2020).

4. FlatOut 2. Официальный сайт. [Электронный ресурс] URL: https://bugbeargames.com/ (Дата обращения: 01.06.2020).

5. Quake III Arena. Официальный сайт. [Электронный ресурс] URL: https://quake.bethesda.net/ru (Дата обращения: 01.06.2020).

6. Unreal Engine. Официальный сайт. [Электронный ресурс] URL: https://www.unrealengine.com/ (Дата обращения: 01.06.2020).

7. CryEngine. Официальный сайт. [Электронный ресурс] URL: https://www.cryengine.com (Дата обращения: 01.06.2020).

8. Unity. Официальный сайт. [Электронный ресурс] URL: https://unity.com/ru (Дата обращения: 01.06.2020).

9. Blender. Официальный сайт. [Электронный ресурс] URL: https://www.blender.org/ (Дата обращения: 01.06.2020).

10. Adobe Fuse CC. Официальный сайт. [Электронный ресурс] URL: https://www.adobe.com/ru/products/fuse.html (Дата обращения: 01.06.2020).

11. FirefireWeapon Asset. Страница в Unity Asset Store. [Электронный ресурс] URL: https://assetstore.unity.com/packages/3d/props/weapons/firefireword-light-machine-gun-95750 (Дата обращения: 01.06.2020).

12. Тестирование программного обеспечения - основные понятия и определения. [Электронный ресурс] URL: http://www.protesting.ru/testing/ (Дата обращения: 01.06.2020).

13. Фаулер М. Основы UML. Краткое руководство по стандартному языку объектного моделирования, 3-е изд.// «Символ Плюс», 2005. –126 с.

14. Фаулер М. Основы UML. Краткое руководство по стандартному языку объектного моделирования, 3-е изд.// «Символ Плюс», 2005. – 139 с.

15. Visual Studio 2019. Официальный сайт. [Электронный ресурс] URL: https://visualstudio.microsoft.com/ru/vs/ (Дата обращения: 01.06.2020).