

Содержание

Содержание.....	3
Введение	5
1 Постановка задачи	6
2 Выбор решения	8
2.1 Проектирование структуры приложения	8
2.2 Определение необходимых модулей программы	11
3 Разработка скриптов игры.....	12
4 Описание разработки приложения	13
4.1 Разработка спрайтов.....	13
4.2 Разработка Сцен для уровней и менб.....	15
4.3 Разработка Таймера.....	17
4.4 Ввод имени	18
4.5 Музыкальное сопровождение.....	19
4.6 Разработка анимации.....	20
4.7 Разработка скриптов	22
4.8 Схема классов.....	23
5 Тестирование приложения	24
6 Сборка приложения.....	27
7 Технические и программные средства	29
Заключение	30
Список использованных источников.....	31
Приложение А	31
Скрипт CameraController	31
Скрипт EndMenu	31
Скрипт Finish	32
Скрипт ItemCollector	32
Скрипт ItemCollector1	33
Скрипт PlayerLife	34
Скрипт PlayerMove	35
Скрипт RandomSpawn	36

Скрипт Rotate.....	36
Скрипт StartMenu	36
Скрипт StickyPlatforms.....	36
Скрипт WaypointFollower	37

Введение

Индустрия компьютерных игр появилась относительно недавно, около 30 лет назад, но уже смогла развиваться в огромную отрасль с колоссальными доходами в несколько миллиардов долларов год. Понять подобный внезапный рост популярности виртуальных развлечений очень просто: все это благодаря широкому распространению компьютерных технологий, в том числе появлению сети Интернет. Благодаря этому, в отличие от других видов развлечений, компьютерные игры более доступны для конечного пользователя.

Для того что бы просто поиграть игроку нужно иметь только компьютер или игровую приставку и копию самой игры, а с широким распространением сети Интернет ради получения копии игры не нужно выходить из дома. Более того, потребителю не требуется иметь особых знаний для того что бы выбрать подходящую для него игру, в то время как для большинства других видов развлечений необходимо разбираться как минимум в необходимой экипировке. Так же стоит принять во внимание, что компьютерные игры в последнее время перестали позиционироваться как программы только для отдыха и развлечений. К примеру, сегодня, благодаря использованию игровых технологий, создаются специальные комплексы по симуляции, которые служат для обучения специалистов в различных областях: от лесорубов до пилотов реактивных самолетов.

В России же, к сожалению, все несколько иначе. Игровая индустрия у нас, так же как и в большинстве других стран постсоветского пространства, развита крайне слабо. Связано это с тем, что культура компьютерных развлечений пришла к нам слишком поздно и практически не развивалась. Из-за этого, даже при достаточно высоком спросе, мы имеем слишком малое количество компаний-разработчиков, которые могут составить конкуренцию зарубежным компаниям.

1 Постановка задачи

В настоящей курсовой работе необходимо разработать компьютерную игру в 2D стиле.

Основной жанр игры был выбран платформер, стилистика уровней-Скандинавия.

Идея игры состоит в том, что игроку на 1 уровне необходимо найти 6 ключей и затем пройти в выход ведущий на 2 уровень. На втором уровне игрок попадает в зону в которой на него сверху падают коллекционные предметы которое ему необходимо собрать за оставшееся время, после того, как таймер игрока станет равен 0, его перебросить на финальный экран на котором он может увидеть количество заработанных очков. Перед началом игры, на стартовом экране игрок может вписать имя, которое затем увидит на финальном экране.

В данной работе необходимо добавить физику, поддержку 2 уровней, стартового и завершающего экрана. Игрок может как победить (набрав какое-то количество очков более 0), так и проиграть (Набрав 0 очков или истратив таймер на 1 уровне)

Исходя из вышесказанного, необходимо проанализировать предметную область, спроектировать классы и их методы для соответствующих игровых объектов.

Немаловажным этапом является разработка логики (физики) игры, перемещение и взаимодействие игровых объектов.

Для повышения качества разрабатываемой игры следует предусмотреть аудиовизуальные эффекты: разработать игровые спрайты, анимацию и звуковое сопровождение игры.

Язык программирования: C#,

Среда программирования: Microsoft Visual Studio 2019,Unity.

Операционная система: Windows 10.

Важно отметить, что C# является языком высокого уровня, и поддерживает технологии ООП. Так же, он является языком для написания скриптов для UNITY, что будет продемонстрировано в следующий разделах.

2 Выбор решения

2.1 Проектирование структуры приложения

Структура приложения зависит от используемой технологии разработки. В данной работе использован язык программирования C# совместно с межплатформенной средой разработки компьютерных игр UNITY.

Он облегчает разработку и управление компьютерными играми. стремится освободить разработку игр от написания «повторяющегося шаблонного кода» и объединить различные аспекты разработки игр в одной системе.

Как было отмечено, приложение построено на использовании различных сцен приложения. Структура приложения представлена на рисунке 1.

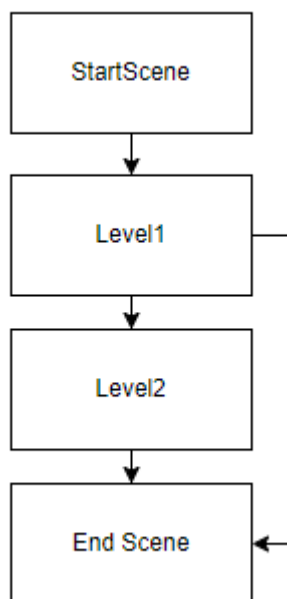


Рисунок 1 – Структура приложения.

UNITY включает инструменты для дизайна уровней и большое количество готовых ассетов, которые можно скачать с их сайта, так же есть инструменты помогающие в настройке анимаций и спрайтов. Помимо этого в ней предоставлено большое количество готовых компонентов, которые могут обеспечить физику объектов, музыкальное сопровождение.

Основа каркаса – это класс MonoBehavior, который является базовым классом от которого наследуются все скрипты. При создании сцен мы можем пользоваться большим набором инструментов от создания UI объектов до TilePallet который позволяет “Рисовать уровень”.

Минимальный проект состоит из 1 сцены и камеры которую игрок может закрепить за любым объектом которые он хочет создать, при разработке были написаны скрипты которые отвечают за движения персонажей, появления предметов, переход между сценами, звуковое сопровождение, взаимодействия между персонажем и предметами и т.д

Рассмотрим имеющиеся скрипты подробнее и отметим, что в части из них реализована функция музыкально сопровождения:

В скрипте CameraController камера привязывается к персонажу и следует за ним.

Скрипт EndMenu предназначен для загрузки переданных ресурсов, и форматировании надписей о победе или поражении игрока.

В скрипте Finish реализуется игровая логика перехода между уровнями

Скрипт ItemCollector служит для взаимодействию с собираемыми предметами и реализует механику поражения и звука подбирания предмета .

Скрипт ItemCollector1 служит для взаимодействию с собираемыми предметами и подсчетами очков для последующей передачей набранных очков в финальный экран звука подбирания предмета.

Скрипт PlayerLife реализует логику смерти персонажа и перезапуск уровня и звук смерти персонажа.

Скрипт PlayerMovement отвечает за перемещение персонажа и звуки прыжков.

Скрипт RandomSpawn реализует случайных спав предметов на 2 уровне.

Скрипт Rotate отвечает за переворот объекта ловушек по своей оси.

Скрипт StartMenu реализует корректную обрисовку стартового меню и ввод имени игрока и передачи её на финальный экран и.

Скрипт StikyPlatform реализует движение персонажа по поверхности платформы

Скрипт WaypointFollower отвечает за перемещение ловушек по заданной оси

2.2 Определение необходимых модулей программы

При разработке программы использовалась концепция ООП. Предметную область необходимо разделить на классы. Классам следует определить методы, т.е. действия, которые будет выполнять определенная сущность, а так же ее свойства и атрибуты.

Такой подход называют восходящим программированием. Программа разделяется на небольшие части (подпрограммы), которые разрабатываются в первую очередь, а затем модифицируются, при необходимости, и используются в дальнейшей разработке.

Анализируя предметную область, можно отметить следующее. Разрабатываемая игра должна иметь два уровня. На первом уровне есть ловушки которые могут убить персонажа они двигаются по разным осям, но используют 1 скрипт, что реализует Использование 1 класса несколькими объектами.

Каждый написанный скрипт использует `MonoBehavior` в качестве базового класса, но вещи за которые эти скрипты отвечают отличаются, чтобы избежать утечки данных были предусмотрены `private` поля у всех необходимых скриптов.

Помимо этого, потребуется реализовать в отдельных классах определенный функционал, связанный с таймингом в игре, работой со списком результатов и другими возможностями в игре.

3 Разработка скриптов игры

Основные функции игры и механики реализованы при помощи компонентов встроенных в UNITY или написанных разработчиком .

Скрипты отвечают за движения персонажа, сбор предметов, музыку, смерть, перемещение объектов,спавн объектов, обрисовку меню, логику победы или поражения, таймер и т.д.

В случае столкновения с собираемым предметом или ловушкой проверяется коллизия между объектами и их теги, при определенных тегах предмет исчезает и счетчик предметом увеличивается или же персонаж умирает и уровень перезапускается.

Спаун объектов создает подбираемые предметы только в определенный области, и отвечает за скорость падения этих предметом, схожая логика работы у скрипта перемещения платформ который двигает платформу между 2 точками.

Результат игрока высвечивается на финальном экране вместе с именем персонажа. Попасть на это меню можно потратив время таймера, однако в зависимости от уровня на котором таймер закончился и количестве очков вид финального экрана может отличаться.

На стартовом экране игрок может ввести имя, которое затем появится в конце.

Ниже приведена блок-схема скрипта itemcollector (рисунок 2).

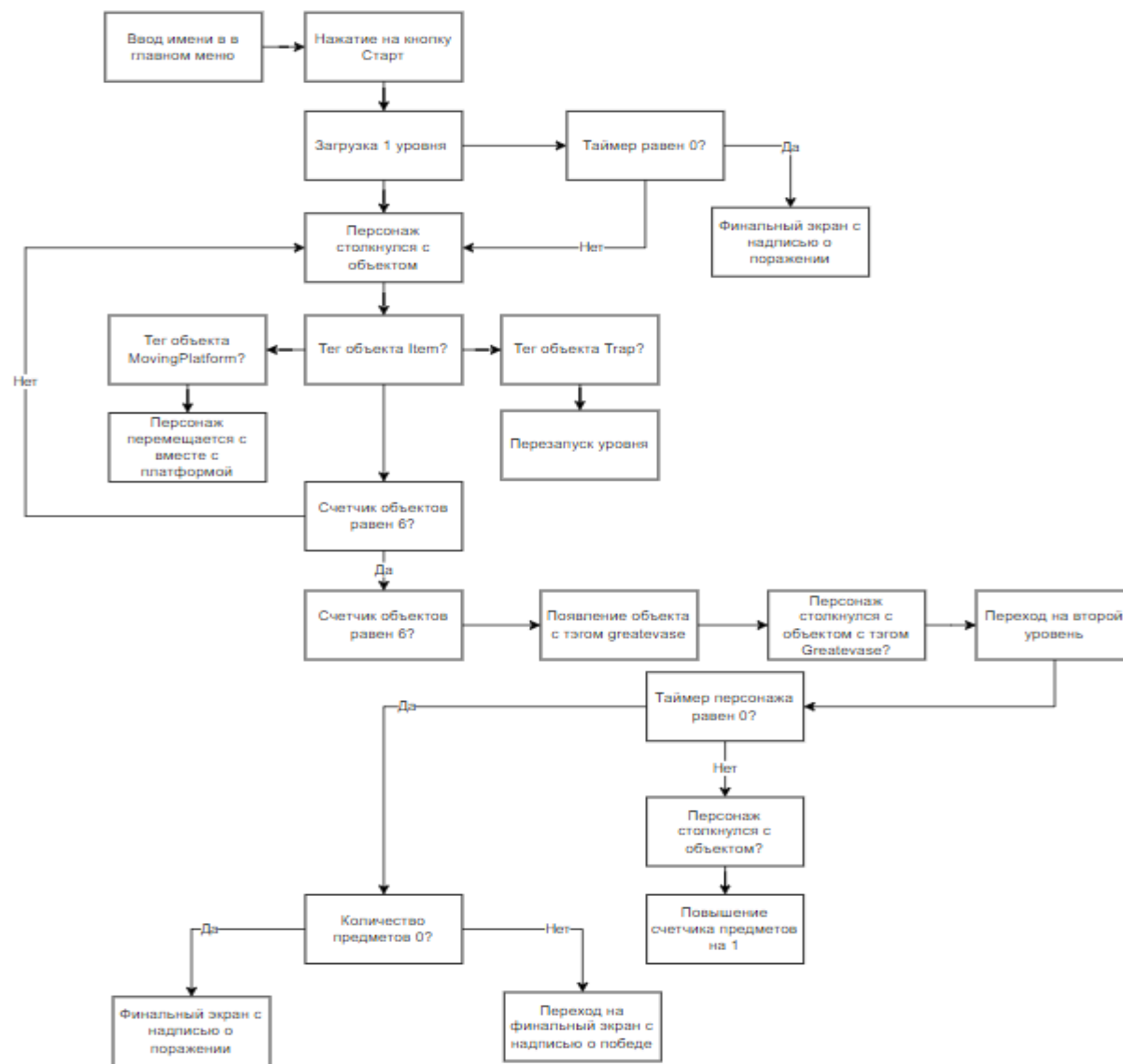


Рисунок 2 – Блок-схема скрипта
ItemCollector.

4.1 Разработка спрайтов

В данной работе были использованы спрайты для персонажа(рисунок 3), коллекционных предметов(рисунок 6), ловушек(рисунки 4-5), платформ(рисунок 7), входа на 2 уровень(рисунок 8)



Рисунок 3 – Спрайт Персонажа.



Рисунок 4 – Спрайт ловушки.



Рисунок 5 – Спрайт противника второго уровня.



Рисунок 6 – Коллекционный предмет



Рисунок 7 – Платформа



Рисунок 8 – Увеличенный в размере спрайт для перехода на 2 уровень .

Подробная информация об использованных спрайтах представлена в таблице 1.

Таблица 1 – Список спрайтов и их названий.

Название спрайта	Название файла
Idle_0	Idle.png
Off	Off.png
Idle	Idle.png
Input	Input.jpg
Items_light_off	Items_light_off.jpg
Items_light_off2	Items_light_off2.png
BrownOn	BrownOn.png

Для возможности загрузки спрайтов использовались компоненты UNITY.

4.2 Разработка Сцен для уровней и менб

Как было сказано ранее, UNITY имеет большой набор инструментов для дизайна локаций .

Одним из инструментов является TilePallet который и был использован .

Для разработки использовались готовые ассеты из магазина UNITY которые подверглись небольшой обработке чтобы соответствовать скандинавскому стилю. Экран TilePalletе и спрайты которые использовались для дизайна уровней представлены на рисунке 9

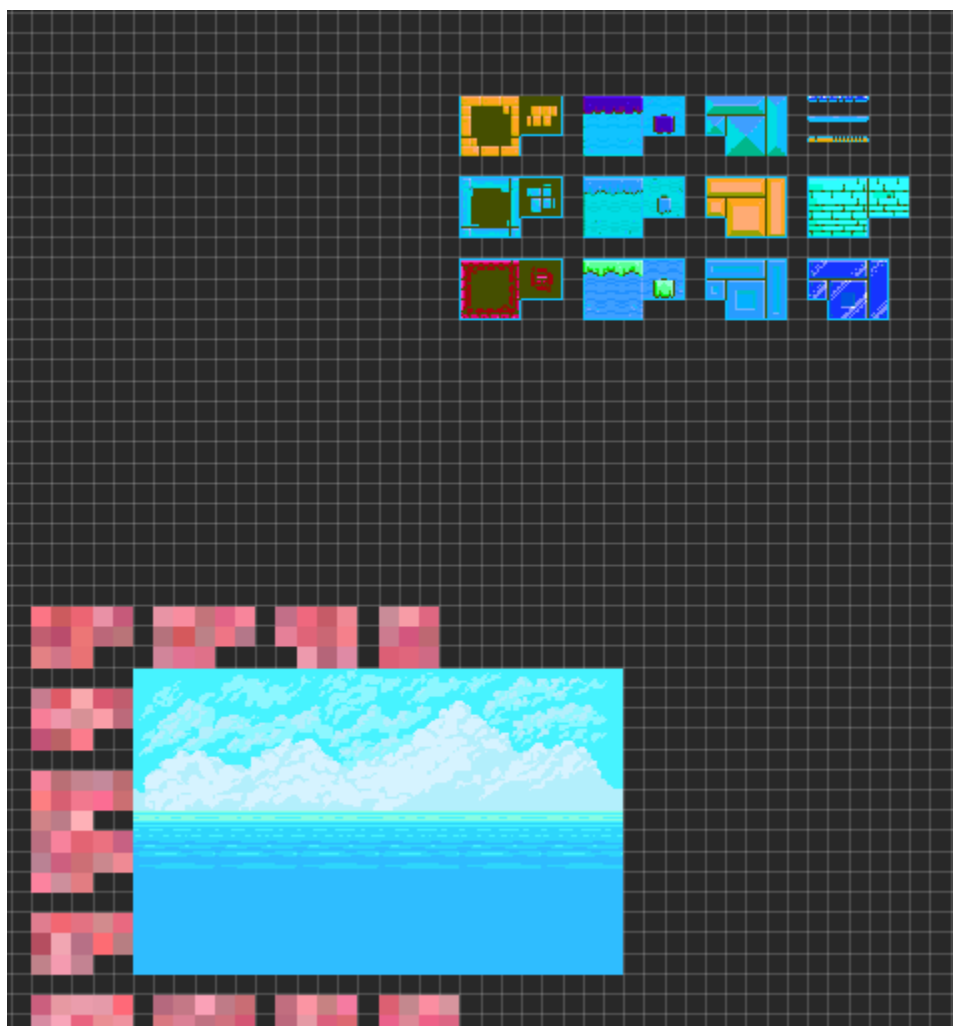


Рисунок 9-Экран TilePallet с спрайтами для дизайна уровней

Для разработки меню были использованы картинки, которые использовали canvas на котором в дальнейшем располагались основные поля меню

Вид стартового и финального экрана представлен на рисунках 10-11



Рисунок 10-Стартовое меню

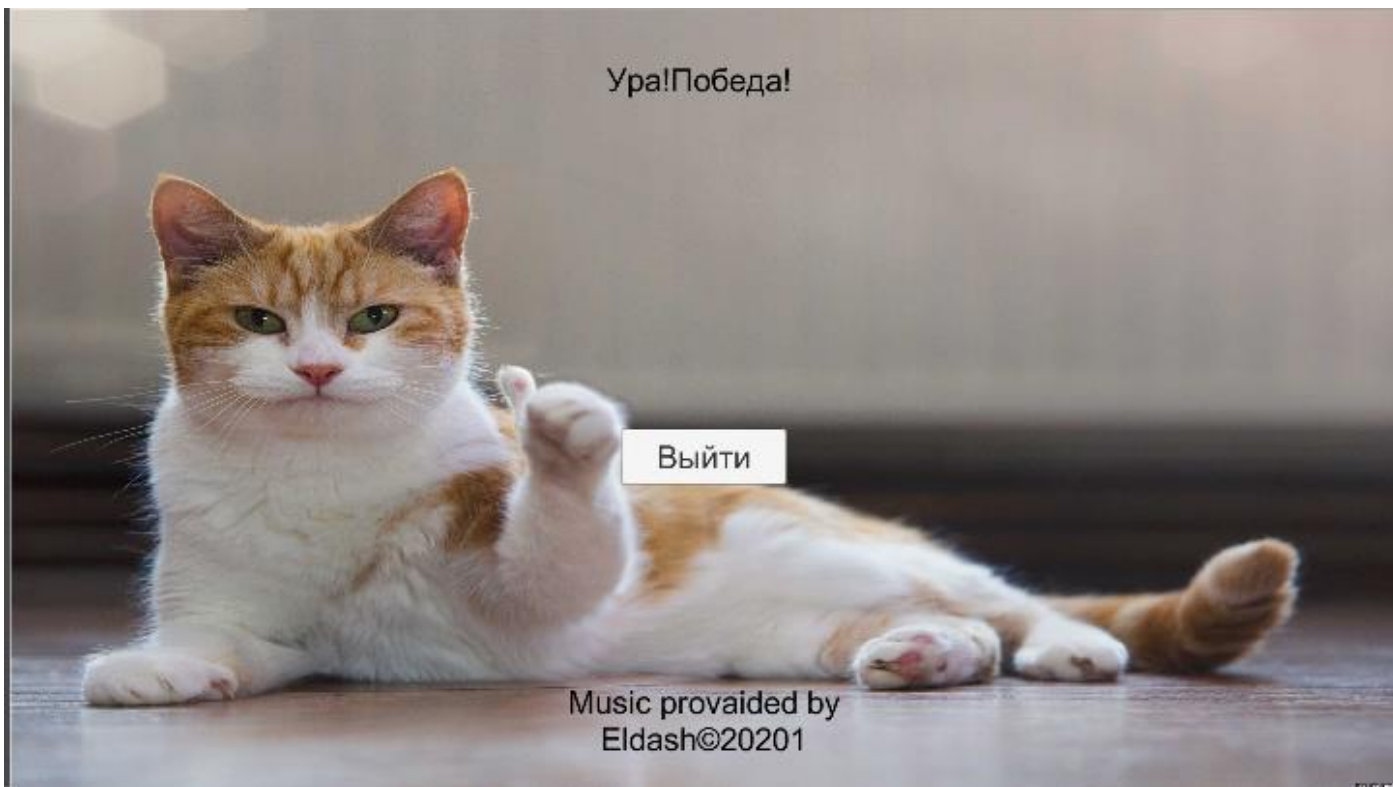


Рисунок 11-Финальное меню

4.3 Разработка Таймера

Таймер был реализован за счет нескольких скриптов, у которых другая основная функция, например подсчет времени.

```
timertext.text ="Time:"+(timer- Convert.ToInt32(Time.realtimeSinceStartup));
```

По сути таймер реализован за счет заранее установленного времени из которого постоянно вычитается время с запуска программы, при достижении таймером 0 игрок переходит на финальный экран

```
if(timer - Convert.ToInt32(Time.realtimeSinceStartup)<=0)

{

    CompliteLevel();

}.
```


4.4 Ввод имени

В данной игре ввод имени осуществляется путем заполнения специального поля и нажатии на кнопку которая также запускает 1 уровень

Вид кнопки и поля ввода представлен на рисунке 12

Код всех скриптов за счет которых осуществляется игровые механики и функции представлен в приложении А

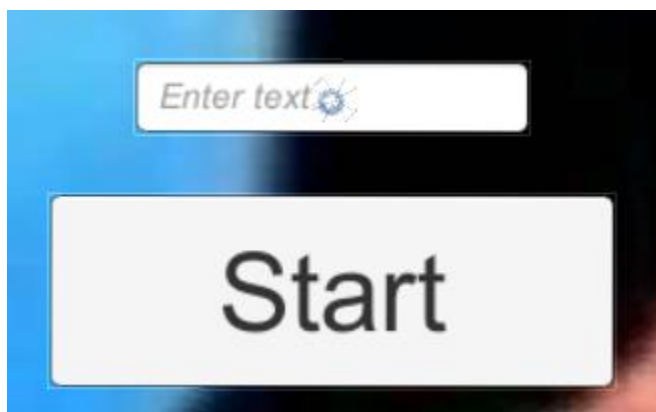


Рисунок 12-Поле ввода имени и кнопка старта

4.5 Музыкальное сопровождение

Музыкальное сопровождение реализованное компонентами UNITY в которых мы указываем нужны источник звука который мы можем настроить, затем в других скриптах реализовано срабатывание специальных триггеров которые воспроизводят звук. Пример настройки компонента AudioSource представлен на рисунке 13

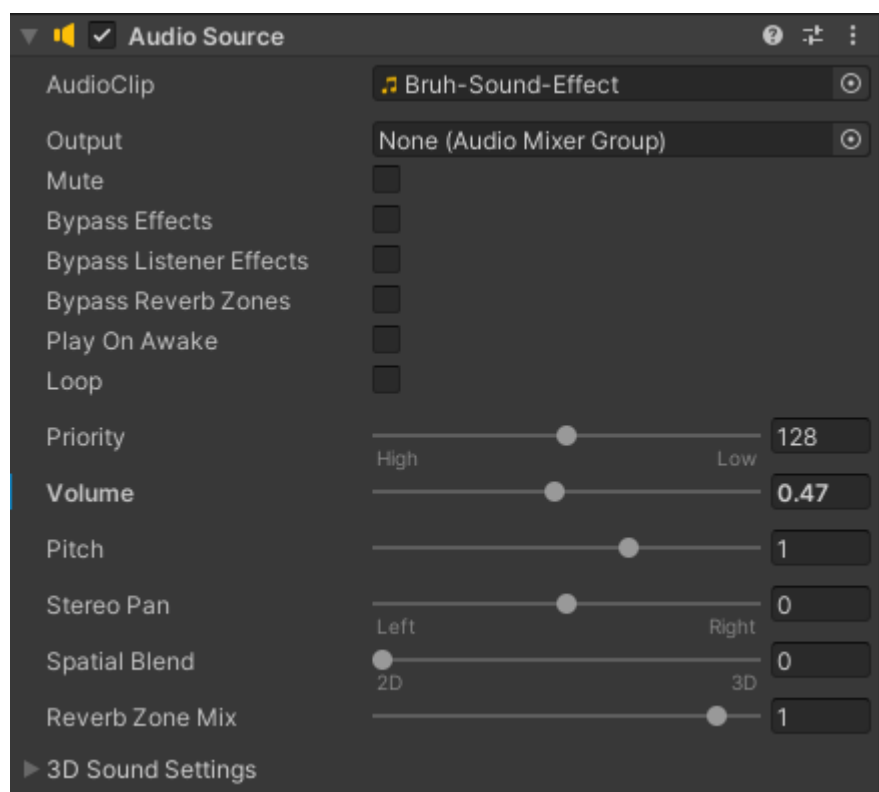


Рисунок 13 – Компонент AudioSource с.

Код всех скриптов за счет которых осуществляется игровые механики и функции представлен в приложении А.

4.6 Разработка анимации

В игре используется больше количество анимаций в основном анимаций персонажа, инструменты UNITY позволяют легко настроить анимации и обеспечить их реакцию на действие персонажа чтобы каждому действию соответствовала своя анимация. Всего в игре есть 5 анимаций их исходные файлы представлены картинками которые последовательно заменяют друг друга

Окно Animator отвечает за смену анимации в зависимости от действий пользователя(рисунок 14)

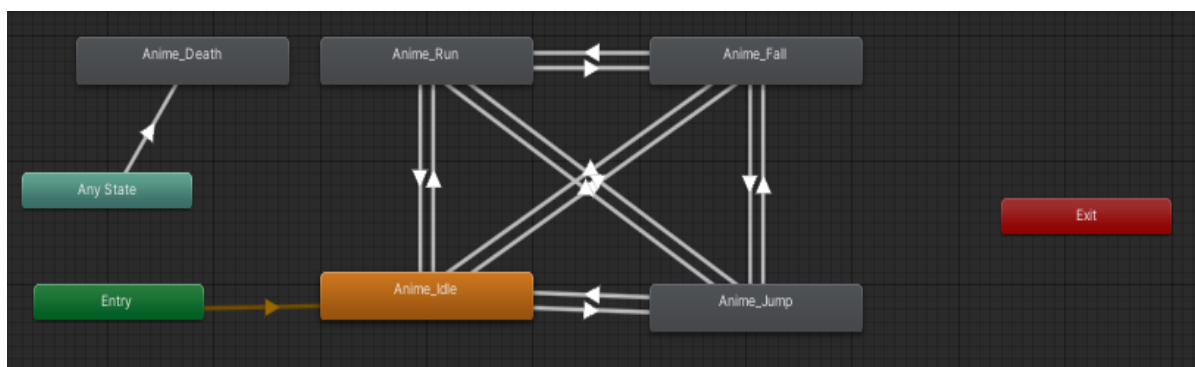


Рисунок 14-Окно Animator

Кроме окна Animator UNITY имеет возможность загружать свои анимации и настраивать их в окне Animation(рисунок 15)

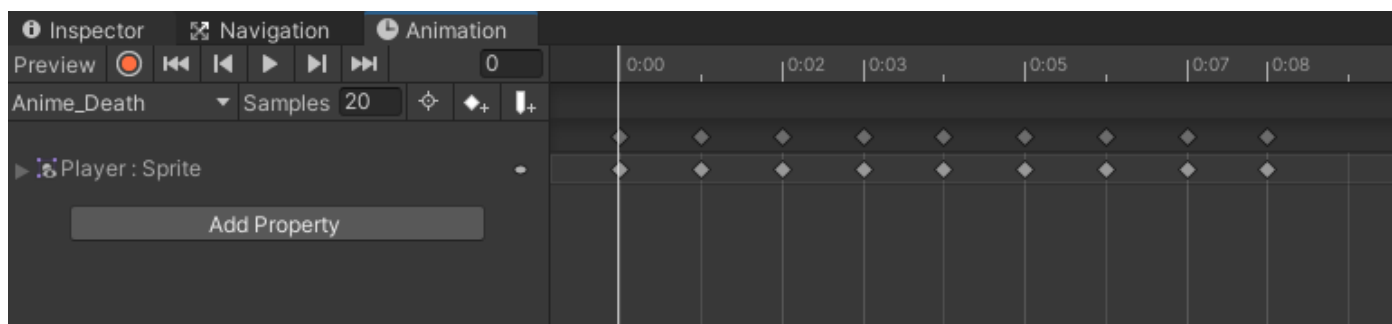


Рисунок 15-Окно Animation

Описание всех анимаций представлено в таблице 2

Таблица 2-Описания анимаций

Название анимаций	Исходный Файл	Описание
Anime_Jump	Jump.Png	Анимация прыжка персонажа
Anime_death	Death.png	Анимация смерти персонажа
Anime_Run	Run.png	Анимация бега
Anime_Fall	Fall.png	Анимация падения
Anime_idle	Idle.png	Анимация стойки

4.7 Разработка скриптов

Unity позволяет удобно использовать готовые компоненты а также писать собственные скрипты которые затем легко привязываются к нужным объектам в окне Inspector(Рисунок 16)

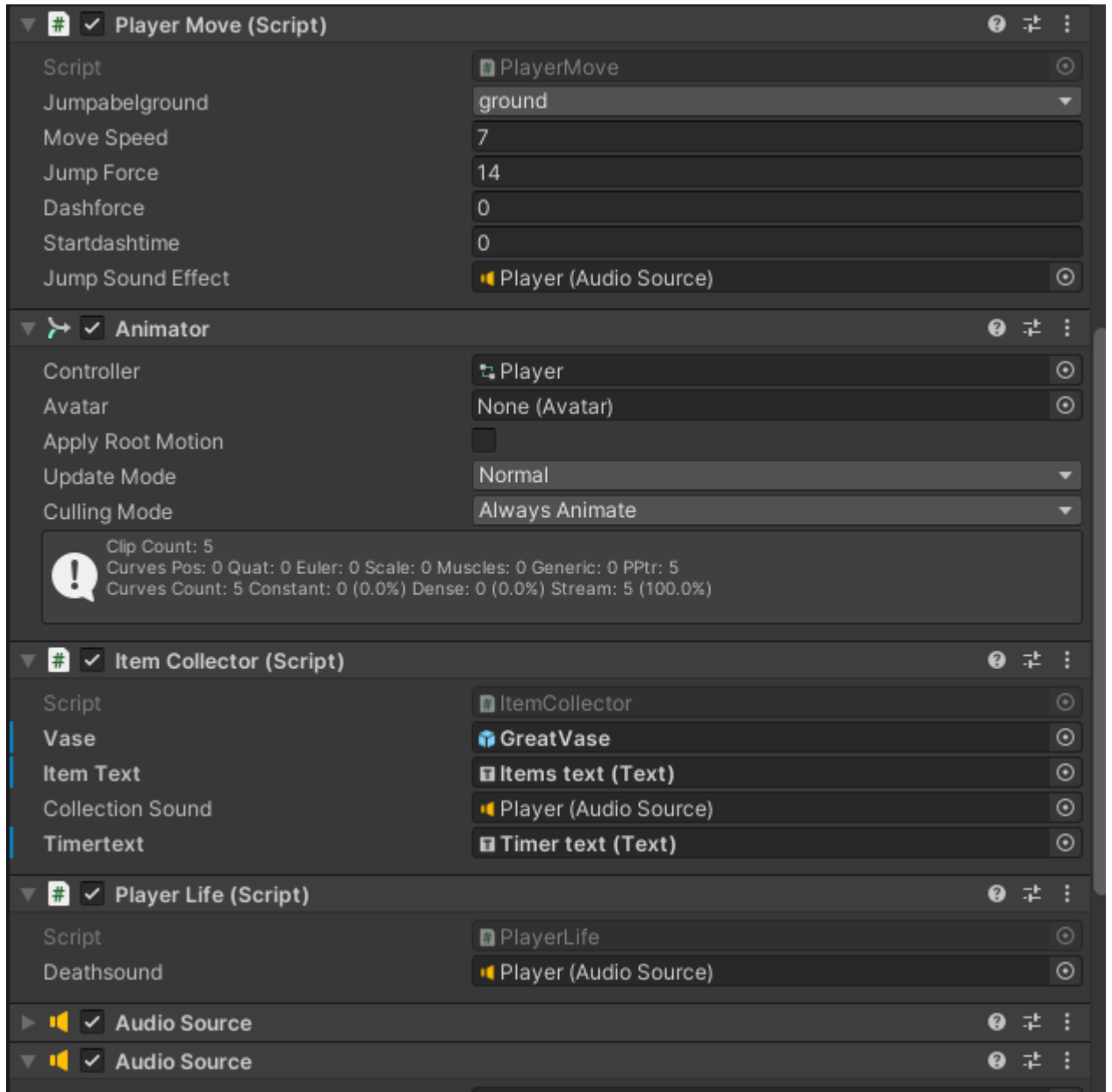


Рисунок 16 – Пример применения написанных скриптов к объекту игрока .

Код всех скриптов представлен в приложении А

4.8 Схема классов

Схема классов представлена на рисунке 17

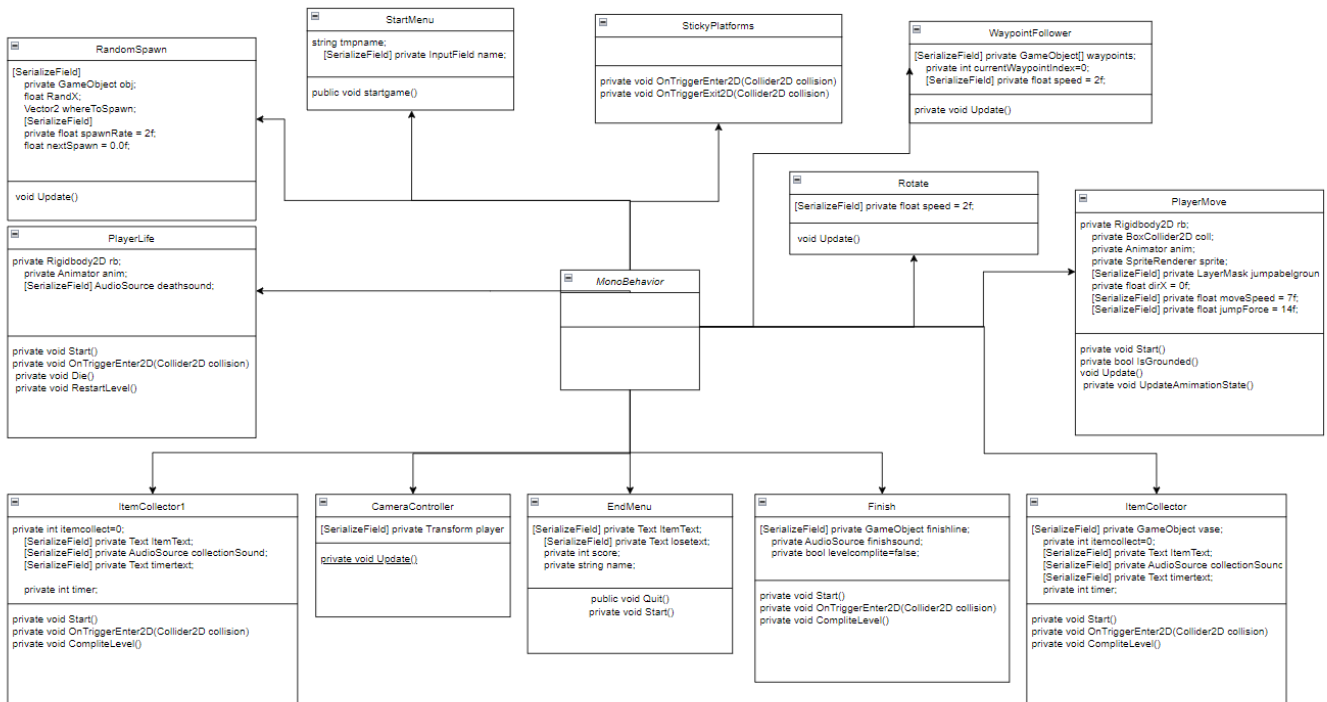


Рисунок 17-Схема классов

На рисунке 17 приведена схема классов разработанной программы. Как было сказано ранее, использование данного подхода позволяет решить задачу методом восходящего программирования, т.е. постепенно переходить от частного к общему. Следовательно, каждый класс создан для определенной задачи. Класс MonoBehavior является базовым классом от которого остальные наследуют основные методы например Start, Update и т.д

Ранее было описано что все механизмы в курсовой работе выполняются на основе скриптов, каждый скрипт реализован в формате класса с собственными методами, таким образом описание работы каждого класса можно посмотреть в пункте 2.1 Код же самих скриптов и реализованных к ним классам можно посмотреть в приложении А

5 Тестирование приложения

После выбора стартовой сцены мы можем запустить её.

Появилось стартовое окно которые можно увидеть на рисунке 18.



Рисунок 18 – Стартовая сцена.

Далее был осуществлен ввод имени и нажата кнопка Start. Открылась сцена первого уровня – рисунок 19.

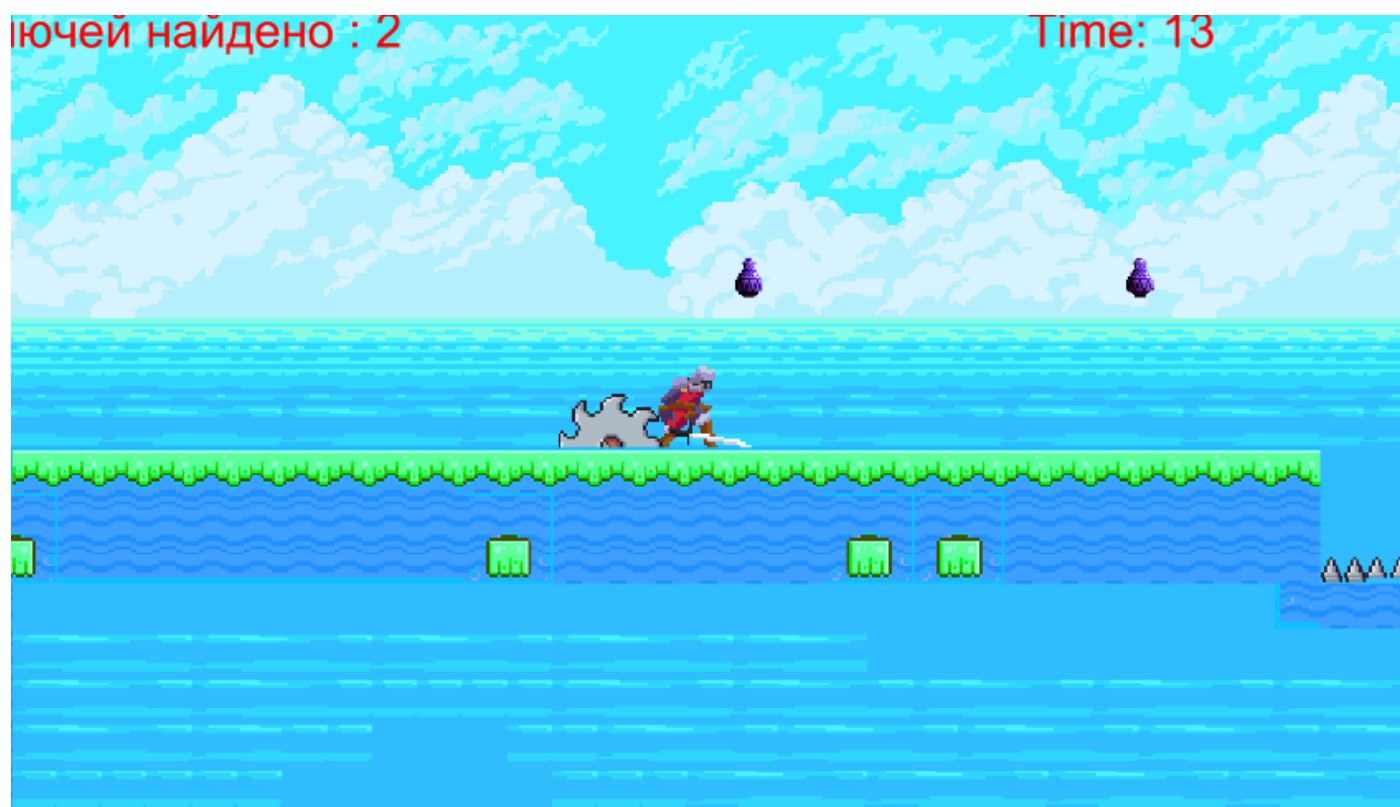


Рисунок 19 – Первый уровень игры

Затем был осуществлен переход на 2 уровень рисунок 20.

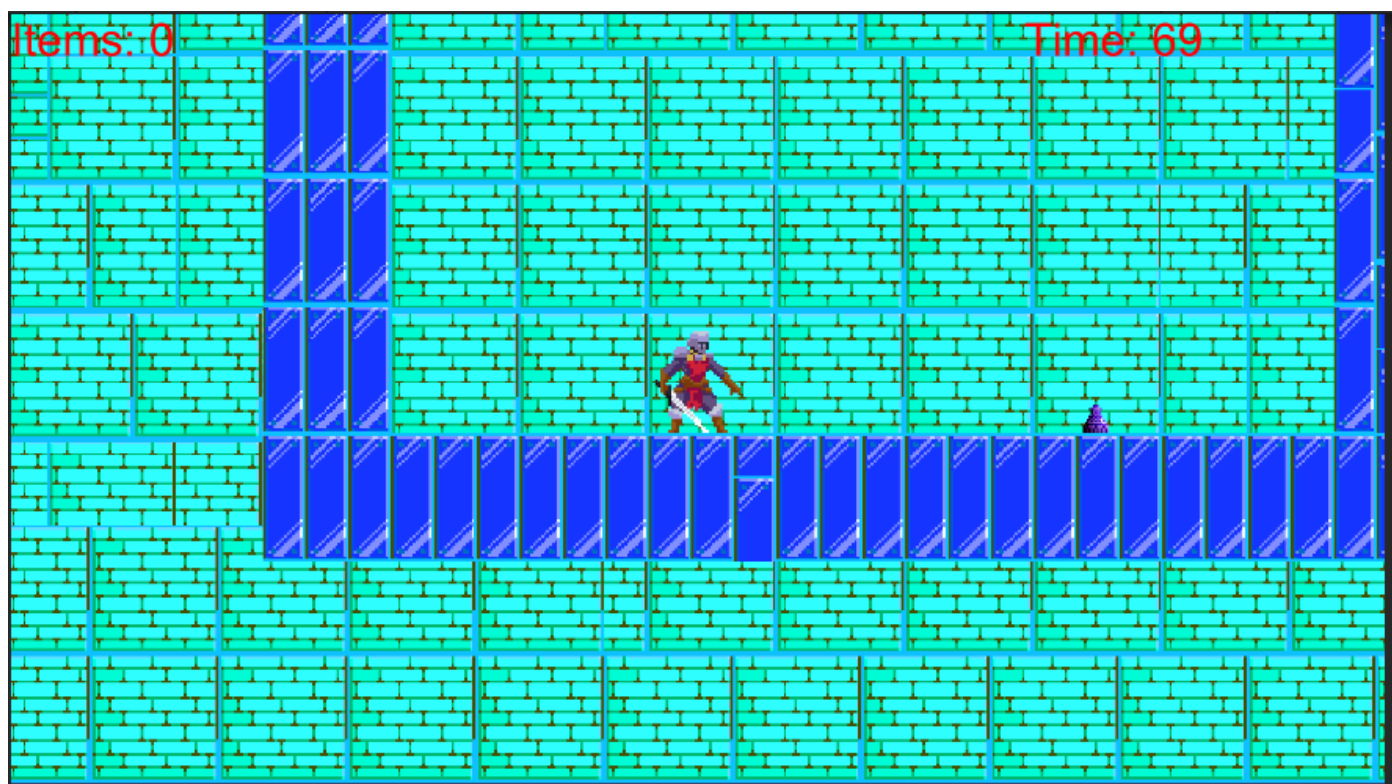


Рисунок 20 – 2 уровень .

Последняя сцена это финальное окно с результатами рисунок -21



Рисунок 21 – Заключительная сцена.

Таким образом видно, что все сцена исправно работают и взаимодействуют друг с другом, полное технодемо игры представлено преподавателю.

6 Сборка приложения

Сборка приложения осуществляется средствами UNITY путем выбора сцен которые будут включены в готовый проект.

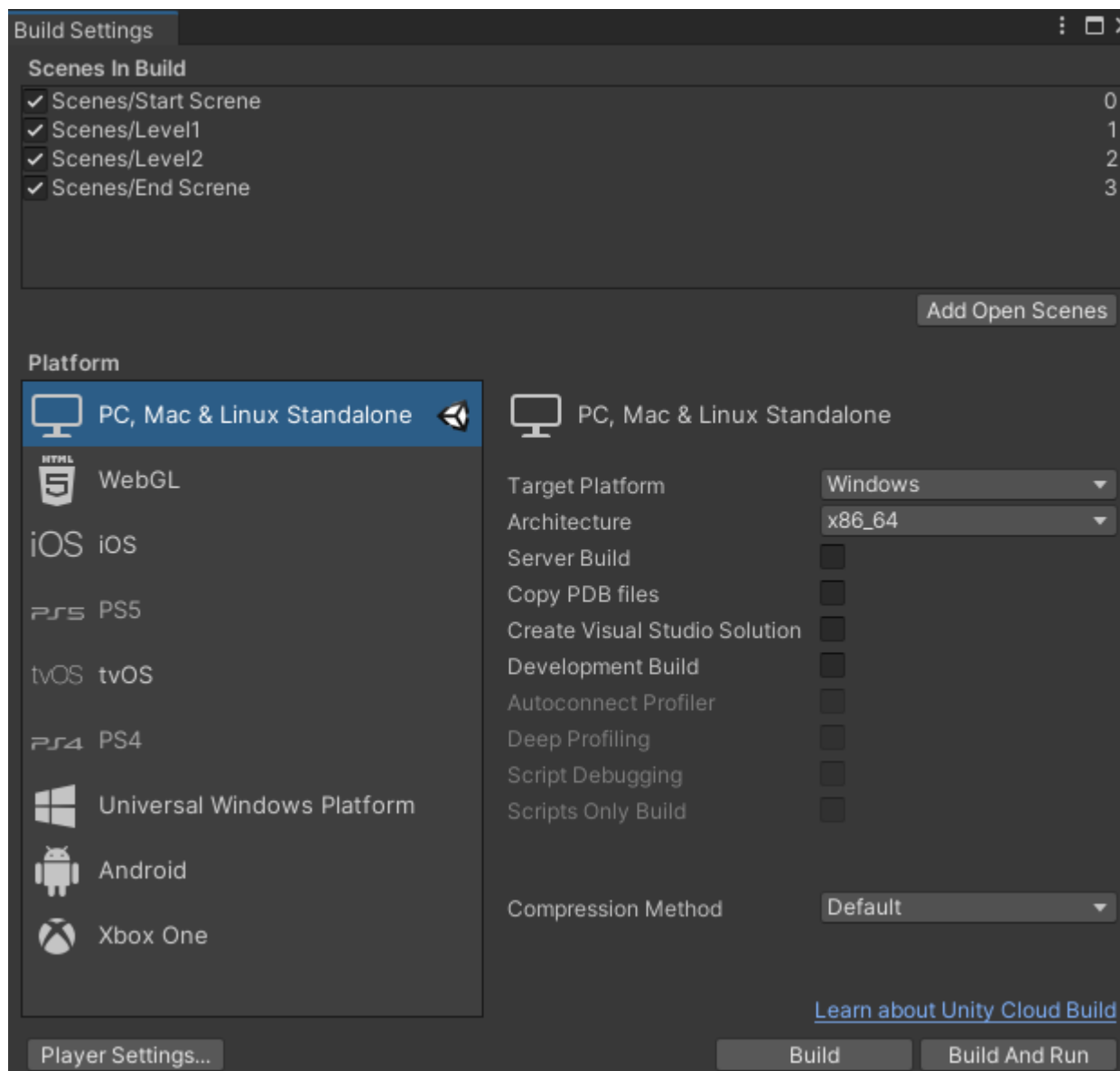


Рисунок 22 – Окно сборки проекта.

Скомпилированная папка выглядит следующим образом (рисунок 23).






.альный диск (E:) > test >				
Имя	Дата изменения	Тип	Размер	
 MonoBleedingEdge	20.11.2021 12:02	Папка с файлами		
 RealOne_Data	27.12.2021 0:40	Папка с файлами		
 RealOne.exe	13.10.2021 12:51	Приложение	639 КБ	
 UnityCrashHandler64.exe	13.10.2021 12:52	Приложение	1 204 КБ	
 UnityPlayer.dll	13.10.2021 12:52	Расширение при...	27 490 КБ	

Рисунок 23 – Собранный проект .

7 Технические и программные средства

Аппаратные требования:

- 32-разрядный (x86) или 64-разрядный (x64) процессор с тактовой частотой 1 ГГц или выше.
- 1 ГБ (для 32-разрядного процессора) или 2 ГБ (для 64-разрядного процессора) ОЗУ.

Программные требования:

- .NET Framework 4.5
- Требования памяти:
- Размер собранной папки : 90,03 МБ.

Заключение

Таким образом, была разработана игра в формате 2D в скандинавском стиле включающая 2 уровня, созданы спрайты, анимации, логика смерти персонажа, сбор коллекционных предметов, случайный спавн предметов, стартовое и финальное меню с отображением счета и введенным именем игрока, звуковое сопровождение, реализована победа и поражение игрока посредством таймера, реализована физика и хождение персонажа

Программа написана на языке C# методом восходящего программирования с использованием возможностей UNITY.

Список использованных источников

- 1) Бертран М. Объектно-ориентированное конструирование программных систем / М. Бертран.-Москва: Русская Редакция, 2005 – 1204с.
- 2) Томас Д.,Хант Э.Программист-програмтик, / Диалектика,2020-368с
- 3) Хокинг Д., Unity в действии. Мультиплатформенная разработка на С#. 2-е межд. Издание/ Издательский дом "Питер", 2018 г – 352с
- 4) Потапова Л.Е. Объектно-ориентированное программирование на языке с# / Л.Е. Потапова. – Витебск, 2012 – 121с.

Приложение А

Скрипт CameraController

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class CameraController : MonoBehaviour
{
    [SerializeField] private Transform player;

    private void Update()
    {
        transform.position = new Vector3(player.position.x,player.position.y,transform.position.z);
    }
}
```

Скрипт EndMenu

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.UI;
using System;
using UnityEngine.SceneManagement;
public class EndMenu : MonoBehaviour
{
    [SerializeField] private Text ItemText;
    [SerializeField] private Text losetText;
    private int score;
    private string name;
    public void Quit()
    {
        Application.Quit();
    }
    private void Start()
    {
        name = PlayerPrefs.GetString("name");
        score= PlayerPrefs.GetInt("itemscore");
        if (score == 0)
        {
            ItemText.text = name + ", к сожалению вы не успели набрать очки";
            losetText.text = "Досадное поражение";
        }
    }
}
```

```

        else {
            ItemText.text = name + ", ВАШ СЧЕТ: " + score;
        }
    }
}

```

Скрипт Finish

```

using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.SceneManagement;

public class Finish : MonoBehaviour
{
    [SerializeField] private GameObject finishline;
    private AudioSource finishsound;
    private bool levelcomplite=false;
    private void Start()
    {
        finishsound = GetComponent<AudioSource>();
    }

    private void OnTriggerEnter2D(Collider2D collision)
    {
        if (collision.gameObject.name == "Player" && !levelcomplite)
        {
            finishline.transform.position =new Vector2(12, 11);
            finishsound.Play();
            levelcomplite = true;
            Invoke("CompliteLevel", 1.5f);
        }
    }
    private void CompliteLevel()
    {
        SceneManager.LoadScene(SceneManager.GetActiveScene().buildIndex + 1);
    }
}

```

Скрипт ItemCollector

```

using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.UI;
using System;
using UnityEngine.SceneManagement;
public class ItemCollector : MonoBehaviour
{
    [SerializeField] private GameObject vase;
    private int itemcollect=0;
    [SerializeField] private Text ItemText;
    [SerializeField] private AudioSource collectionSound;
    [SerializeField] private Text timertext;
    private int timer;
    private void Start()
    {
        timer =75;
    }
}

```

```

        vase.SetActive(false);
    }
    private void Update()
    {
        timertext.text = "Time: " + (timer - Convert.ToInt32(Time.realtimeSinceStartup));
        if(timer - Convert.ToInt32(Time.realtimeSinceStartup)<=0)
        {
            CompliteLevel();
        }
    }
    private void OnTriggerEnter2D(Collider2D collision)
    {
        if (collision.gameObject.CompareTag("Items"))
        {
            collectionSound.Play();
            Destroy(collision.gameObject);
            itemcollect++;
            ItemText.text = "Ключей найдено : " + itemcollect;
        }
        if (itemcollect >= 6)
        {
            vase.SetActive(true);
            vase.transform.position = new Vector3(-7, -2, 0);
            ItemText.text = "Найдите выход!";
        }
        if (collision.gameObject.CompareTag("Greatvase") && itemcollect>=1)
        {
            vase.transform.position = new Vector3(11, 12, 0);
        }
    }
}
private void CompliteLevel()
{
    itemcollect = 0;
    PlayerPrefs.SetInt("itemscore", itemcollect);
    timertext.text = "";
    SceneManager.LoadScene(SceneManager.GetActiveScene().buildIndex + 2);
}
}

```

Скрипт ItemCollector1

```

using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.UI;
using System;
using UnityEngine.SceneManagement;
public class ItemCollector1 : MonoBehaviour
{
    private int itemcollect=0;
    [SerializeField] private Text ItemText;
    [SerializeField] private AudioSource collectionSound;
    [SerializeField] private Text timertext;

    private int timer;
    private void Start()
    {
        itemcollect = 0;
        timer = 75;
    }
}

```



```

private void Update()
{
    ItemText.text = "Items: " + itemcollect;
    timertext.text = "Time: " + (timer-Convert.ToInt32(Time.realtimeSinceStartup));
    if ((timer - Convert.ToInt32(Time.realtimeSinceStartup)) <= 0)
    {
        CompliteLevel();
    }
}
private void OnTriggerEnter2D(Collider2D collision)
{
    if (collision.gameObject.CompareTag("Items"))
    {
        collectionSound.Play();
        Destroy(collision.gameObject);
        itemcollect++;
        ItemText.text = "Items: " + itemcollect;
    }
}

}
private void CompliteLevel()
{
    PlayerPrefs.SetInt("itemscore", itemcollect);
    timertext.text = "";
    SceneManager.LoadScene(SceneManager.GetActiveScene().buildIndex + 1);
}
}

```

Скрипт PlayerLife

```

using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.SceneManagement;

public class PlayerLife : MonoBehaviour
{
    private Rigidbody2D rb;
    private Animator anim;
    [SerializeField] AudioSource deathsound;
    // Start is called before the first frame update
    private void Start()
    {
        anim = GetComponent<Animator>();
        rb = GetComponent<Rigidbody2D>();
    }

    // Update is called once per frame
    private void OnCollisionEnter2D(Collision2D collision)
    {
        if (collision.gameObject.CompareTag("Trap"))
        {
            Die();
        }
    }
    private void Die()
    {
        deathsound.Play();
        anim.SetTrigger("death");
        rb.bodyType = RigidbodyType2D.Static;
    }
    private void RestartLevel()
    {
        SceneManager.LoadScene(SceneManager.GetActiveScene().name);
    }
}

```

Скрипт PlayerMove

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class PlayerMove : MonoBehaviour
{
    private Rigidbody2D rb;
    private BoxCollider2D coll;
    private Animator anim;
    private SpriteRenderer sprite;
    [SerializeField] private LayerMask jumpabelground;
    private float dirX = 0f;
    [SerializeField] private float moveSpeed = 7f;
    [SerializeField] private float jumpForce = 14f;

    private enum MovementState {idle,running,jumping,falling}

    [SerializeField] private AudioSource jumpSoundEffect;

    // Start is called before the first frame update
    void Start()
    {
        rb = GetComponent<Rigidbody2D>();
        coll = GetComponent<BoxCollider2D>();
        sprite = GetComponent<SpriteRenderer>();
        anim = GetComponent<Animator>();
    }

    // Update is called once per frame
    void Update()
    {
        dirX = Input.GetAxisRaw("Horizontal");
        rb.velocity = new Vector2(dirX * moveSpeed, rb.velocity.y);

        UpdateAmimationState();
    }
    private void UpdateAmimationState()
    {
        MovementState state;
        if (Input.GetButtonDown("Jump") && IsGrounded() || (Input.GetButtonDown("Fire1") && IsGrounded()))
        {
            jumpSoundEffect.Play();
            rb.velocity = new Vector2(0, jumpForce);
        }

        if (dirX > 0f)
        {
            state = MovementState.running;
            sprite.flipX = false;
        }
        else if (dirX < 0f)
        {
            state = MovementState.running;
            sprite.flipX = true;
        }
        else
        {
            state = MovementState.idle;
        }

        if (rb.velocity.y > .1f)
        {
            state = MovementState.jumping;
        }
        else if (rb.velocity.y < -.1f)
        {
            state = MovementState.falling;
        }

        anim.SetInteger("state", (int)state);
    }
    private bool IsGrounded()
```

```

    {
        return Physics2D.BoxCast(coll.bounds.center, coll.bounds.size, 0f, Vector2.down, .1f, jumpabelground);
    }
}

```

Скрипт RandomSpawn

```

using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class RandomSpawn : MonoBehaviour
{
    [SerializeField]
    private GameObject obj;
    float RandX;
    Vector2 whereToSpawn;
    [SerializeField]
    private float spawnRate = 2f;
    float nextSpawn = 0.0f;
    // Start is called before the first frame update
    void Start()
    {
    }

    // Update is called once per frame
    void Update()
    {
        if(Time.time>nextSpawn)
        {
            nextSpawn = Time.time + spawnRate;
            RandX = Random.Range(-0.87f, 18.47f);
            whereToSpawn = new Vector2(RandX, transform.position.y);
            Instantiate(obj, whereToSpawn, Quaternion.identity);
        }
    }
}

```

Скрипт Rotate

```

using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class Rotate : MonoBehaviour
{
    [SerializeField] private float speed = 2f;

    private void Update()
    {
        transform.Rotate(0, 0, 360 * speed * Time.deltaTime);
    }
}

```

Скрипт StartMenu

```

using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.UI;
using System;
using UnityEngine.SceneManagement;

public class StartMenu : MonoBehaviour
{
    string tmpname;
    [SerializeField] private InputField name;
    public void startgame()
    {
        tmpname = name.text;
        PlayerPrefs.SetString("name",tmpname);
        SceneManager.LoadScene(SceneManager.GetActiveScene().buildIndex + 1);
    }
}

```

Скрипт StickyPlatforms

```

using System.Collections;
using System.Collections.Generic;

```

```

using UnityEngine;

public class StickyPlatforms : MonoBehaviour
{
    private void OnTriggerEnter2D(Collider2D collision)
    {
        if (collision.gameObject.name == "Player")
        {
            collision.gameObject.transform.SetParent(transform);
        }
    }
    private void OnTriggerExit2D(Collider2D collision)
    {
        if (collision.gameObject.name == "Player")
        {
            collision.gameObject.transform.SetParent(null);
        }
    }
}

```

Скрипт WaypointFollower

```

using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class WaypointFollower : MonoBehaviour
{
    [SerializeField] private GameObject[] waypoints;
    private int currentWaypointIndex=0;
    [SerializeField] private float speed = 2f;

    private void Update()
    {
        if (Vector2.Distance(waypoints[currentWaypointIndex].transform.position, transform.position) < .1f)
        {
            currentWaypointIndex++;
            if (currentWaypointIndex >= waypoints.Length)
            {
                currentWaypointIndex = 0;
            }
        }
        transform.position = Vector2.MoveTowards(transform.position,
        waypoints[currentWaypointIndex].transform.position, Time.deltaTime * speed);
    }
}

```