СОДЕРЖАНИЕ

АННОТАЦИЯ 4

ВВЕДЕНИЕ 7

1. АНАЛИТИЧЕСКАЯ ЧАСТЬ 8
   1. Постановка задачи 8
   2. Анализ предметной области (концептуальное моделирование) 8
2. РАЗРАБОТКА ЛОГИЧЕСКОЙ МОДЕЛИ БД 9
   1. Построение диаграммы «сущность-связь» в нотации П.Чена 9
   2. Построение модели основанной на ключах (Key Based model, KB) 9
   3. Построение полной атрибутивной модели в нотации IDEF1X 12
3. РАЗРАБОТКА ФИЗИЧЕСКОЙ МОДЕЛИ БАЗЫ ДАННЫХ 19
   1. Выбор аппаратной и программной платформы для реализации БД 19
   2. Реализация базы данных 19
   3. Тестирование базы данных (создание и реализация запросов) 22
   4. Разграничение прав доступа 24
   5. Расчет информационных параметров базы данных 34
4. РАЗРАБОТКА КЛИЕНТСКОГО ПРИЛОЖЕНИЯ 19
   1. Обоснование выбора языка программирования 19
   2. Разработка интерфейса пользователя 19
   3. Алгоритм работы каждого из модулей 22
   4. Тестирование работы приложения 24

ВЫВОДЫ 35

ПЕРЕЧЕНЬ ССЫЛОК 36

ПРИЛОЖЕНИЕ А 37

ВВЕДЕНИЕ

Концепция организации информации в виде баз данных является значительным фактором при создании различных систем автоматизированной обработки информации. Проектирование подобных программно-технических компонентов информационных систем является комплексной задачей, включающей широкий спектр вопросов, начиная от адекватного моделирования предметной области, до выбора необходимых технических и программных средств, написания эргономических интерфейсов и т.д.

В данной курсовой работе предлагается спроектировать базу данных, начиная от ее логического проектирования на бумаге, и до момента создания физической модели базы данных, которая сможет полностью реализовать принципы работы логической модели. Для достижения поставленной цели потребуется применить знания, полученные в области программирования, проектирования трансляторов и администрирования баз данных.

1 АНАЛИТИЧЕСКАЯ ЧАСТЬ

* 1. Постановка задачи

Целью курсового проекта является проектирование и создание базы данных турфирмы, а также приложения для управления созданной БД.

* 1. Анализ предметной области

Для реализации проекта необходимо выделить сущности. Для построения модели БД турфирмы можно выделить такие сущности как:

* Тур
* Отель
* Скидки
* Клиент
* Сотрудник
* Возврат
* Сводка продаж

Сущность Тур имеет атрибуты Вид тура, Стоимость, Дата отправления, Длительность; Отель характеризуется атрибутами Название, Адрес, Класс; Скидки – Размер скидки; Клиент содержит данные о количестве персон и детей, паспорт и телефон; Для обеспечения возврата должны быть заполнены Дата возврата, Сумма возврата и Причина возврата; Возврат и Продажи осуществляет сотрудник, характеризующийся атрибутами ФИО и Телефон; Совершенные продажи заносятся в Сводку продаж, имеющую атрибут Количество продаж.

2 РАЗРАБОТКА ЛОГИЧЕСКОЙ МОДЕЛИ БД

2.1 Построение диаграммы «сущность-связь» в нотации П.Чена

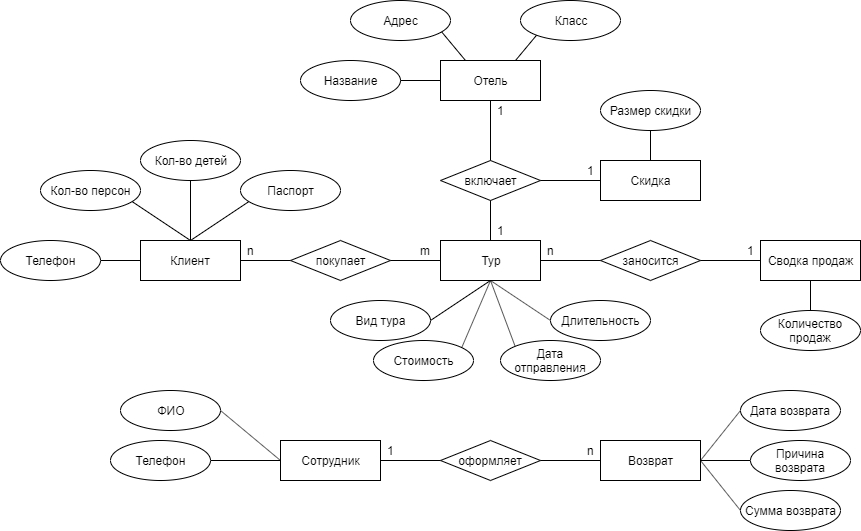


Рисунок 1 – Диаграмма “Сущность-связь” в нотации Чена

2.2 Построение модели основанной на ключах

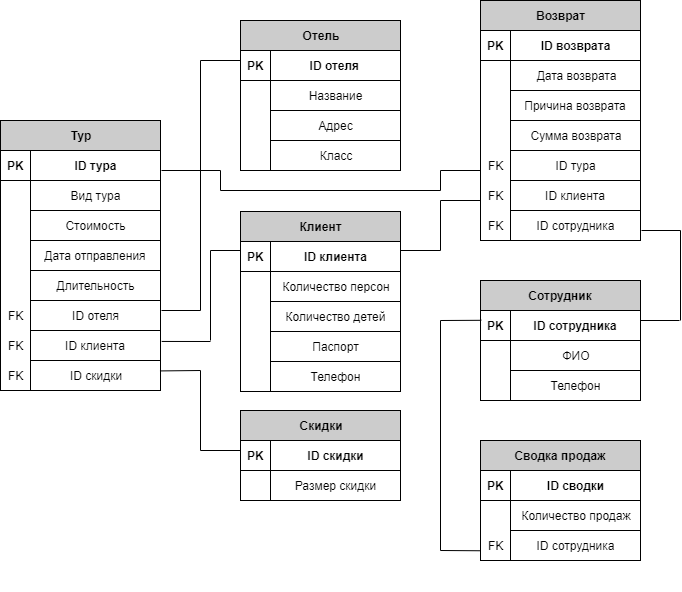


Рисунок 2 – Key Based диаграмма

2.3 Построение информационной модели

На рисунке 3 изображены объекты, располагающиеся на сцене, а также их взаимосвязь.

Рисунок 3 – Информационная модель

Дроп – склянки здоровья или монеты, выпадающие с врагов или разбросанные по карте.

Противник – может быть динамическим объектом, который при обнаружении в зоне видимости главного персонажа бежит на него и атакует. Отнимает определенное количество здоровья за определенный промежуток времени. Также на карте генерируются шипы – опасность для главного героя, при столкновении с ними герой теряет здоровье.

Главный герой – основной персонаж, объект одного из трёх классов, которым управляет игрок.

Сцена – генерируемая случайным образом карта уровня, по которой передвигается главный персонаж и где находятся основные объекты, такие как противники, препятствия и монеты.

UI – элементы интерфейса, позволяющие контролировать состояние персонажа, его расположение на карте и использование уникальных способностей.

Камера – объект, движущийся за персонажем и отображающий игровое поле в определенном радиусе.

2.4 Описание жизненного цикла программы и одного из объектов

MonoBehaviour – базовый класс Unity, от которого наследуются все созданные классы, содержит методы и переменные, необходимые для взаимодействия между объектами и компонентами других классов. Основные вещи, достойные внимания, это две функции, определенные внутри класса.

Функция **Update** – это место для размещения кода, который будет обрабатывать обновление кадра для игрового объекта. Это может быть движение, срабатывание действий и ответная реакция на ввод пользователя, в основном всё, что должно быть обработано с течением времени во игровом процессе.

Функция **LateUpdate** вызывается раз в кадр, после завершения Update. Любые вычисления, произведённые в Update будут уже выполнены на момент начала LateUpdate. Часто LateUpdate используют для преследующей камеры от третьего лица. Если вы перемещаете и поворачиваете персонажа в Update, вы можете выполнить все вычисления перемещения и вращения камеры в LateUpdate. Это обеспечит то, что персонаж будет двигаться до того, как камера отследит его позицию.

Функция **Start** будет вызвана Unity до начала игрового процесса (т.е. до первого вызова функции Update), и это идеальное место для выполнения инициализации.

Функция **OnApplicationQuit** вызывается для всех игровых объектов перед тем, как приложение закрывается.

Жизненный цикл класса MonoBehaviour представлен на рисунке 4.

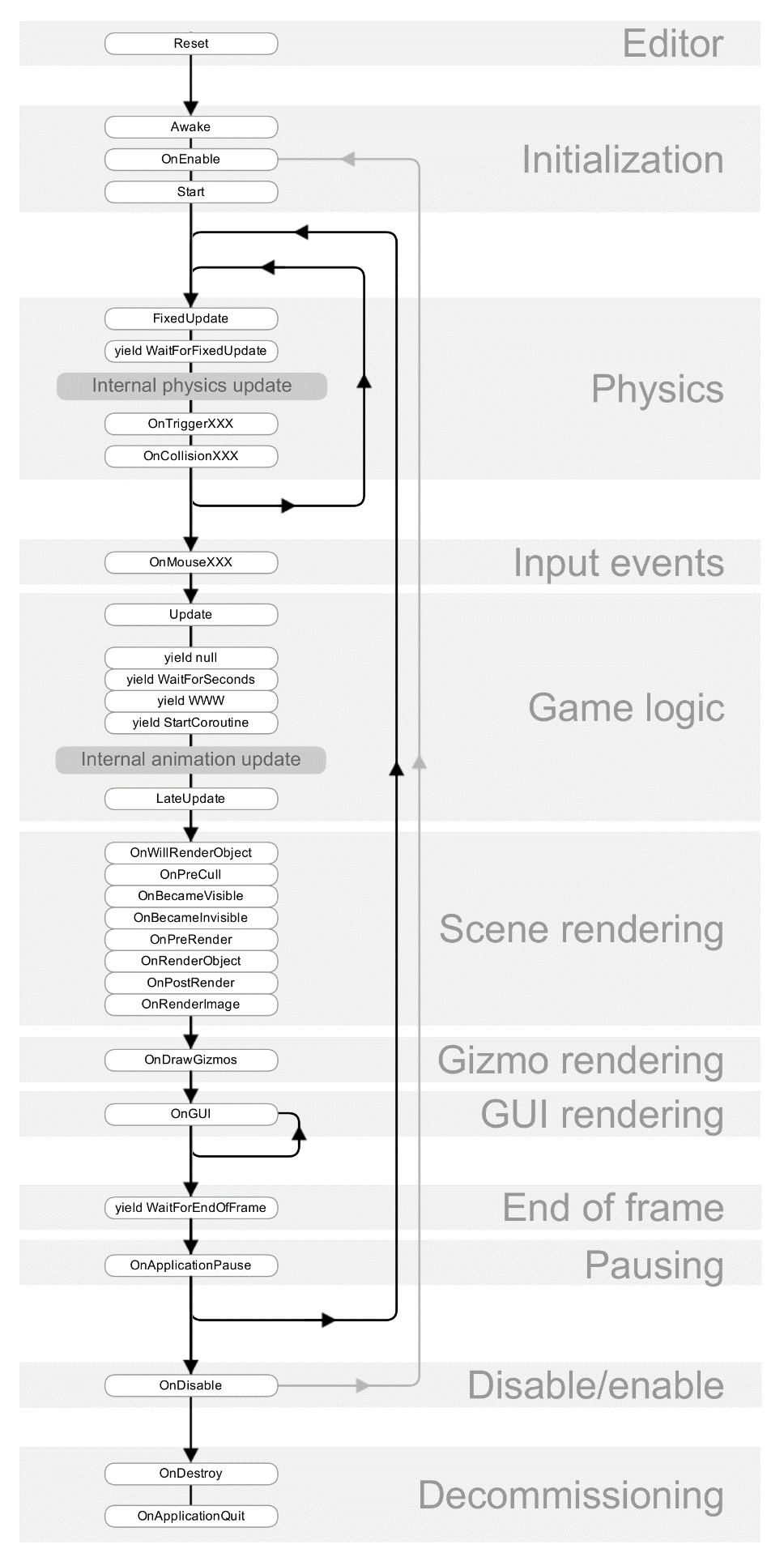


Рисунок 4 – Жизненный цикл класса MonoBehaviour

После запуска игры активируется объект, которым управляет игрок. Он может двигаться, драться с противниками, что вызовет действие проверки здоровья. Если запас здоровья равен нулю, то персонаж уничтожается, и игра начинается заново. Данный цикл изображён на рисунке 5.

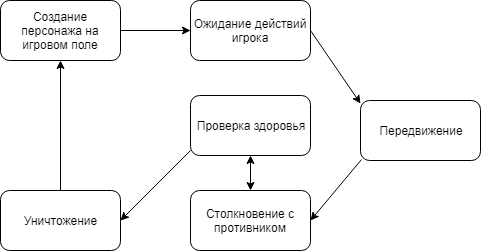


Рисунок 5 – Диаграмма жизненного цикла игрока

2.5 Диаграмма переходов состояний

Состояние *Выбор персонажа* – первое состояние игрового приложения. При переходе в него становится доступно меню выбора персонажа. После выбора персонажа происходит переход в состояние *Игровой процесс*. Диаграмма состояний выбора персонажа представлена на рисунке 6.

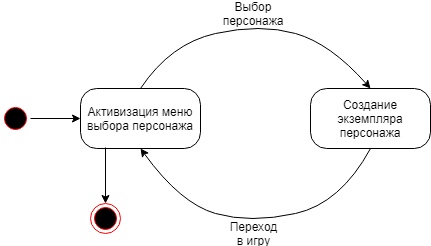


Рисунок 6 – Диаграмма состояний выбора персонажа

Состояние *Игровой процесс* – в этом состоянии игроку доступно передвижение персонажа, битва с врагом и поиск перехода на следующий уровень. Диаграмма состояний игрового процесса представлена на рисунке 7.

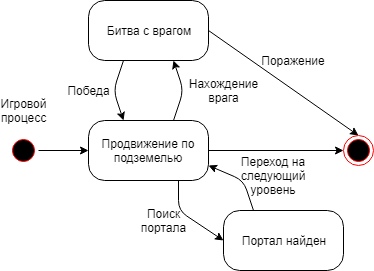


Рисунок 7 – Диаграмма состояний игрового процесса

Состояние *Битва с врагом* – в это состояние игрок переходит, когда встречается с врагом. Персонаж и враг обменивается уроном и каждый кадр проверяется их здоровье. Диаграмма состояний битвы с врагом представлена на рисунке 8.

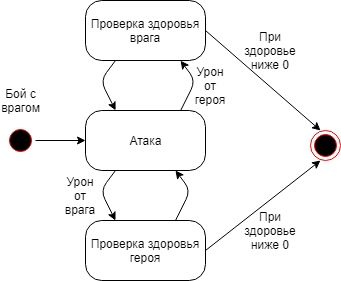


Рисунок 8 – Диаграмма состояний битвы с врагом

Состояние *Движение врага* – если враг не видит персонажа, он двигается по случайным координатам, иначе он получает координаты персонажа и двигается к нему. Диаграмма состояний движения врага представлена на рисунке 9.

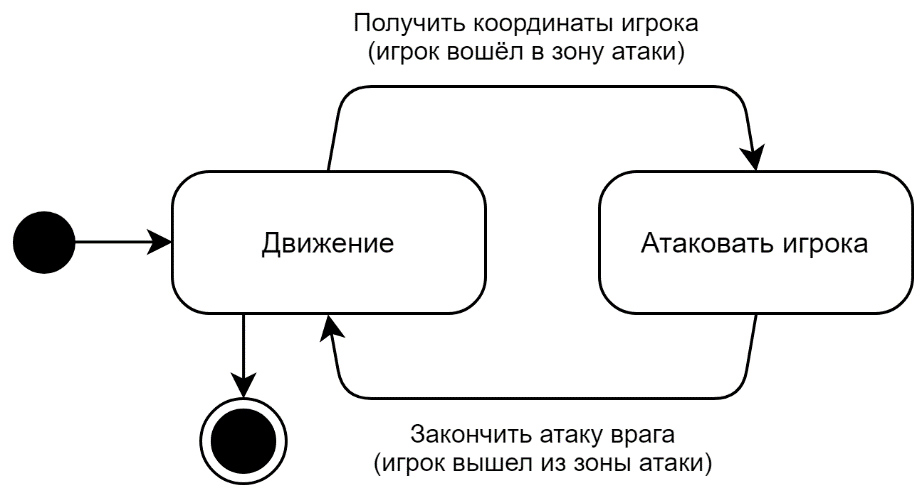


Рисунок 9 – Диаграмма состояний движения врага

Состояние *Появления врага* – система создает точки появления врагов в комнате. Диаграмма состояний появления врага представлена на рисунке 10.

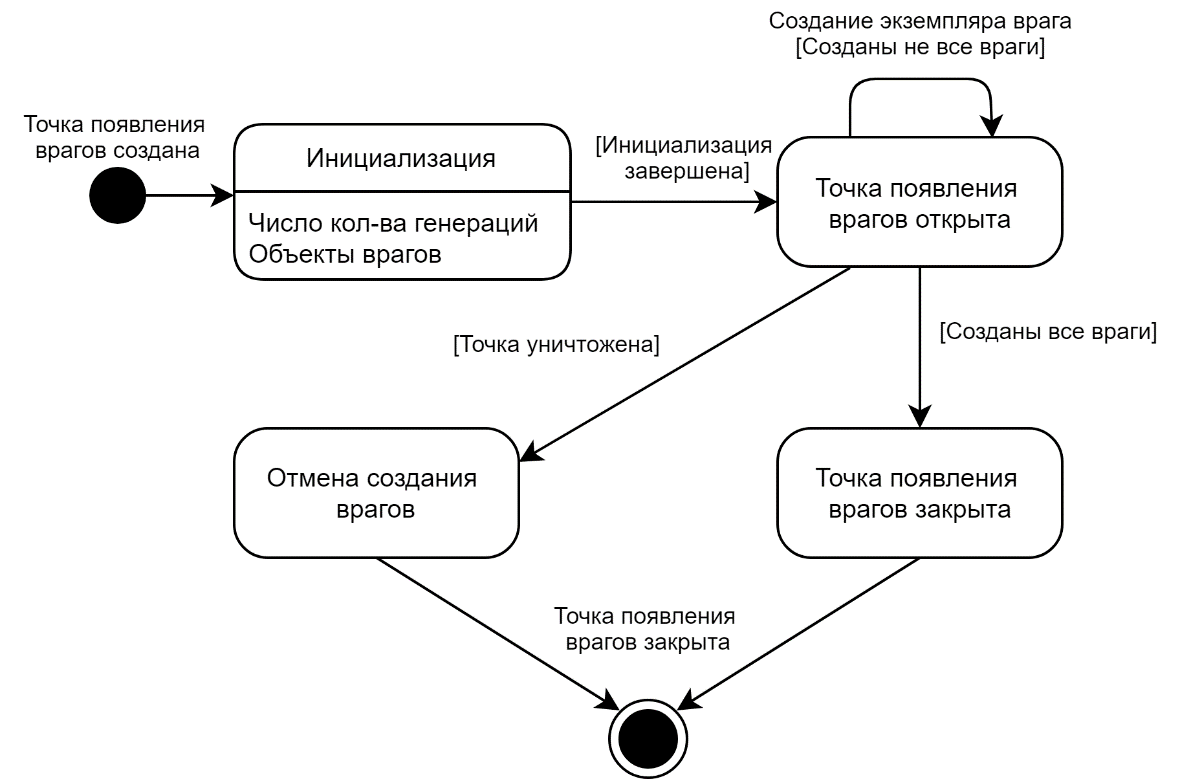


Рисунок 10 – Диаграмма состояний появления врага

Состояние *Босса* – система создает экземпляр класса босс, в этом состоянии объект создает снаряды, летящие в персонажа. Каждый кадр происходит проверка здоровья и если оно равно 0, то объект уничтожается. Диаграмма состояний Босса представлена на рисунке 11.

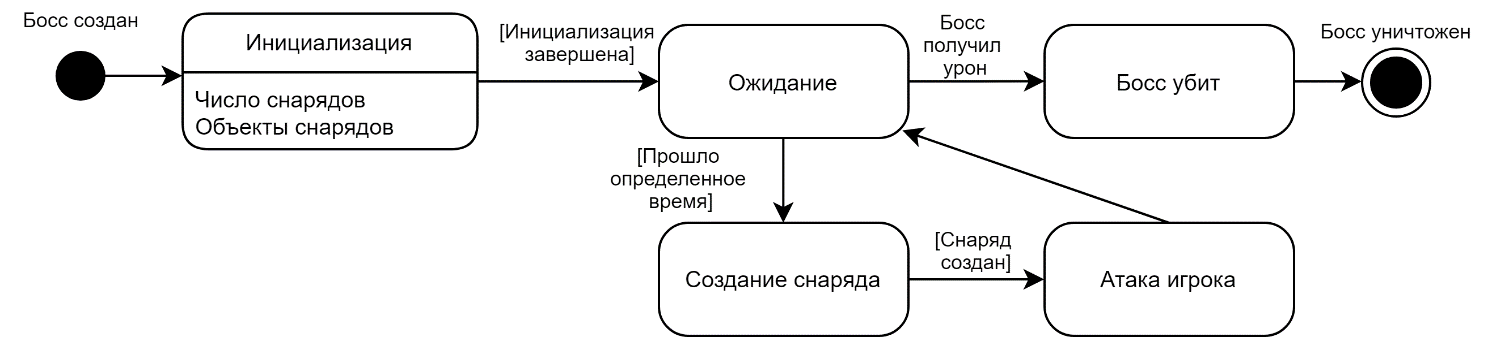


Рисунок 11 – Диаграмма состояний босса

2.6 Диаграмма потоков данных и действий

На рисунке 12 проиллюстрировано какие потоки данных влияют на состояние ожидания и какие данные влияют на другие состояния.



Рисунок 12 – Диаграмма потоков данных для состояния ожидания

Входящими данными являются текущий уровень здоровья и координаты врага. Исходящими данными являются введенные клавиши, которые влияют на дальнейшие действия персонажа.

3 ПРОГРАММНАЯ РЕАЛИЗАЦИЯ

3.1 Обоснование выбора языка (системы) программирования

Основным средством разработки данного проекта является игровой движок Unity. Unity – это мультиплатформенный инструмент для разработки двух- и трёхмерных приложений и игр, работающий под операционными системами Windows и OS X. Созданные с помощью Unity приложения работают под операционными системами Windows, OS X, Windows Phone, Android, Apple. Есть возможность создавать интернет-приложения с помощью специального подключаемого модуля к браузеру Unity. Приложения, созданные с помощью Unity, поддерживают DirectX и OpenGL.

Разработка велась на языке C#, а в качестве редактора кода был выбран Microsoft Visual Studio. Это линейка продуктов компании Microsoft, включающих интегрированную среду разработки программного обеспечения и ряд других инструментальных средств. Данные продукты позволяют разрабатывать как консольные приложения, так и приложения с графическим интерфейсом, в том числе с поддержкой технологии Windows Forms, а также веб-сайты, веб-приложения, веб-службы как в родном, так и в управляемом кодах для всех платформ, поддерживаемых Windows, Windows Mobile, Windows CE, .NET Framework, Xbox, Windows Phone .NET Compact Framework и Silverlight.

3.2 Общее описание взаимодействия программных модулей

Для упрощения поиска и редактирования файлов были созданы папки для разного рода файлов (рис. 13):

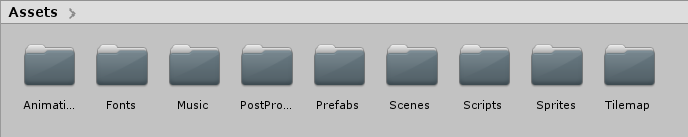


Рисунок 13 – Структура файлов

Animations — файлы анимаций, Fonts — используемые шрифты, Music — музыка и звуковые эффекты, PostProcessing — плагин постобработки изображения, Prefabs — заготовки объектов, Scenes – сцены проекта, Scripts — файлы скриптов (классов), Sprites — изображения игровых объектов и интерфейса, Tilemap – плагин анимации тайлов.

Особенностью Unity является его система управления объектами: каждому объекту можно присвоить какое угодно количество скриптов (имя скрипта соответствует имени класса и по сути это одно и тоже) и управление объектом совершается через эти скрипты (рис. 14).

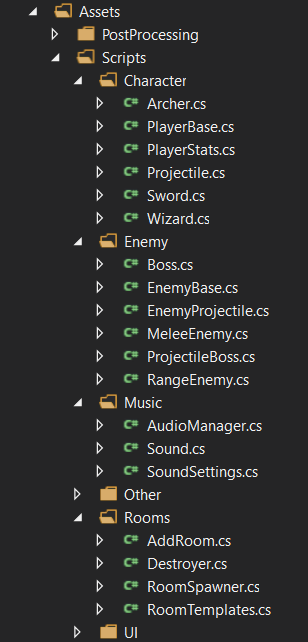


Рисунок 14 – Структура скриптов

В папке Scripts есть несколько типов управляющих скриптов, предназначенных для конкретных объектов. Character — скрипты персонажей, Enemy — скрипты врагов, Music — система управления звуками и музыкой, Other — скрипты побочных игровых объектов, Rooms – скрипты для генерации комнат, UI – скрипты для управления пользовательским интерфейсом.

В папке Prefabs находятся различные шаблоны используемых в проекте объектов (рис. 15):



Рисунок 15 – Шаблоны объектов

После загрузки моделей производятся различные действия с ними: применение скрипта, изменение свойств объекта, редактирование полей скрипта. Для того что бы не производить это вручную или внутри управляющего скрипта можно использовать интерфейс игрового движка Unity (рис. 16).

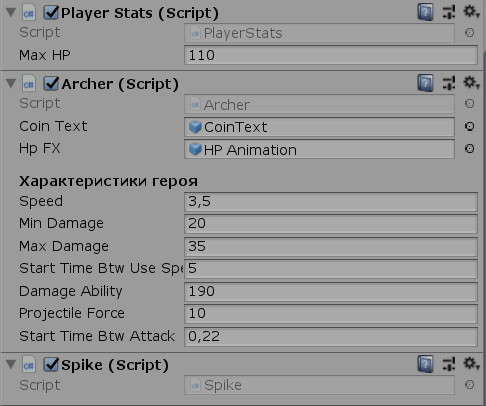


Рисунок 16 – Настройки скрипта персонажа Archer

3.3 Описание реализации основных классов и их методы

1) Класс PlayerBase содержит методы:

* protected virtual void TakeInput() – виртуальная функция, реализующая ввода клавиш управления.
* protected void Move() – функция отвечает за движение персонажа.
* private void AnimationMove(Vector 2 direction) – функция отвечает за анимацию персонажа.
* private void OnTriggerEnter(Colider2D other) – функция отвечает за взаимодействие с монетами.
* protected abstract void Attack() – абстрактная функция атаки, переопределяемая в наследуемых классах.

2) Sword, Archer и Wizard наследуются от базового класса PlayerBase и переопределяют функции TakeInput и Attack.

Класс Sword содержит уникальные методы Energy и Armor и метод отображения атаки HitEffect.

Класс Archer содержит метод ArrowAbility, который описывает уникальную способность выпустить усиленный снаряд.

Класс Wizard содержит уникальный метод Dash, который описывает уникальную способность рывка в направлении движения.

3) Класс EnemyBase содержит методы:

* protected abstract void Following() – абстрактный метод, отвечающий за движение за игроком.
* protected abstract void Attack() – абстрактный метод, отвечающий за атаку.
* protected void FlipX(float other) – метод, отвечающий за поворот спрайта врага.
* protected void TakeDamage(int damage) – метод предназначен для получения урона от игрока.
* protected void ShowDamage(int damage) – метод, предназначенный для отображения получаемого урона.
* protected void CheckDeath() – функция, проверяющая здоровья врага.

4) MeleeEnemy, RangeEnemy и Boss наследуются от базового класса EnemyBase и переопределяют функции Following и Attack.

Класс MeleeEnemy и RangeEnemy содержит разные методы определения координат персонажа и следования за ним.

Класс Boss переопределяет базовую функцию Attack с использованием Coroutine.

5) Классы AddRoom, Destroyer, RoomSpawner отвечают за генерацию игрового уровня, используя массив шаблонов комнат из класса RoomTemplates.

6) Следующие классы не содержат сложных функций и привязаны к простым объектам:

* Camera – отвечает за движение камеры за персонажем.
* Coins – предназначена для учета выпадающих монет.
* Portal – содержит функцию, отвечающая за переход на следующие уровни.
* BottleHP – содержит функцию для использования зелий.
* ArmorAbility – класс, реализующий методы для отображения защиты персонажа.
* Jar – класс отвечает за интерактивные предметы в игровом мире.
* Spike – класс реализует столкновения персонажа с шипами, наносящими урон.

7) Классы, реализующие GUI:

* BarHP – класс, отвечающий за отображения здоровье персонажа.
* Character – реализует выбор персонажа.
* Menu – содержит функции, реализующие главное меню.

8) Классы, отвечающие за звуковые эффекты:

* AudioManager – содержит массив звуков и функции их воспроизведения.
* SoundSettings – реализует меню настройки звука.

3.4 Интерфейс пользователя

Программа имеет собственный установщик, представленный на рисунке 17.

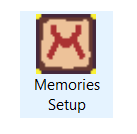


Рисунок 17 – Установщик игры

Далее проиллюстрирован процесс установки (рис. 18-20).

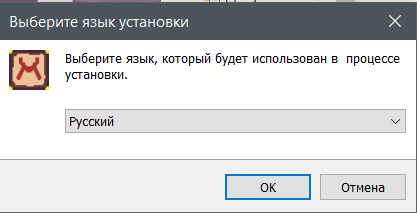


Рисунок 18 – Выбор языка установщика

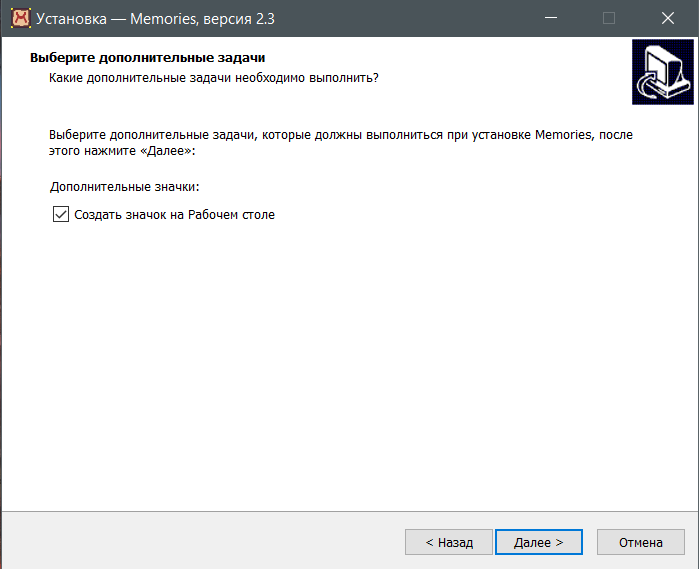


Рисунок 19 – Запрос на создание ярлыка игры

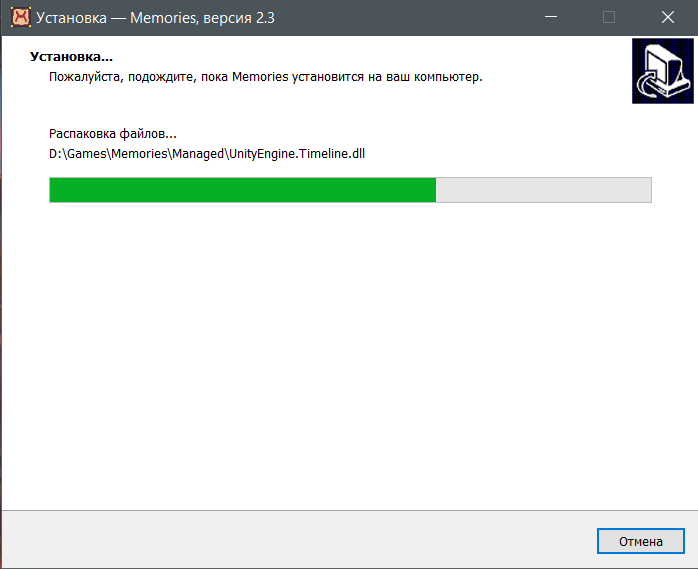


Рисунок 20 – Прогресс установки игры

Файл запускающий игру находится в папке, представленной на рисунке 21. При открытии исполнительного файла Memories.exe запускается игра (рис. 22).

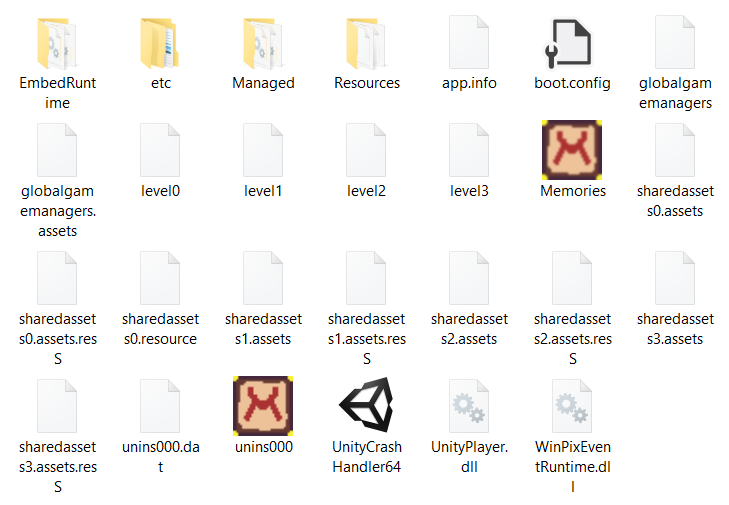


Рисунок 21 – Содержание папки с игрой

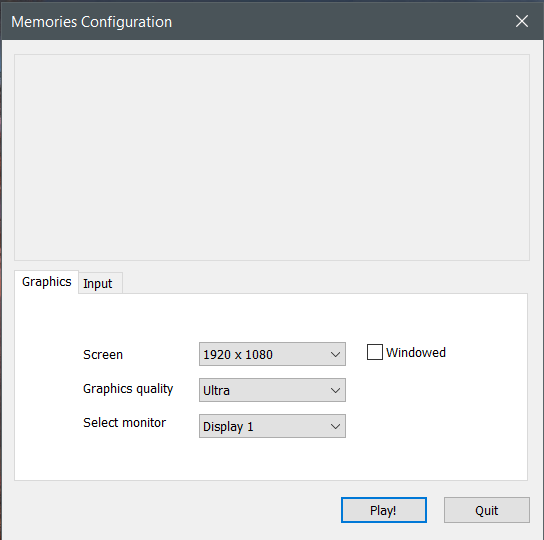


Рисунок 22 – Выбор настроек при запуске игры

Первое что видит игрок – главное меню, представленное на рисунке 23 и содержащее пункты: play, options, quit.



Рисунок 23 – Главное меню

В настройках игрок имеет возможность изменить уровень громкости музыки и звуков (рис. 24).

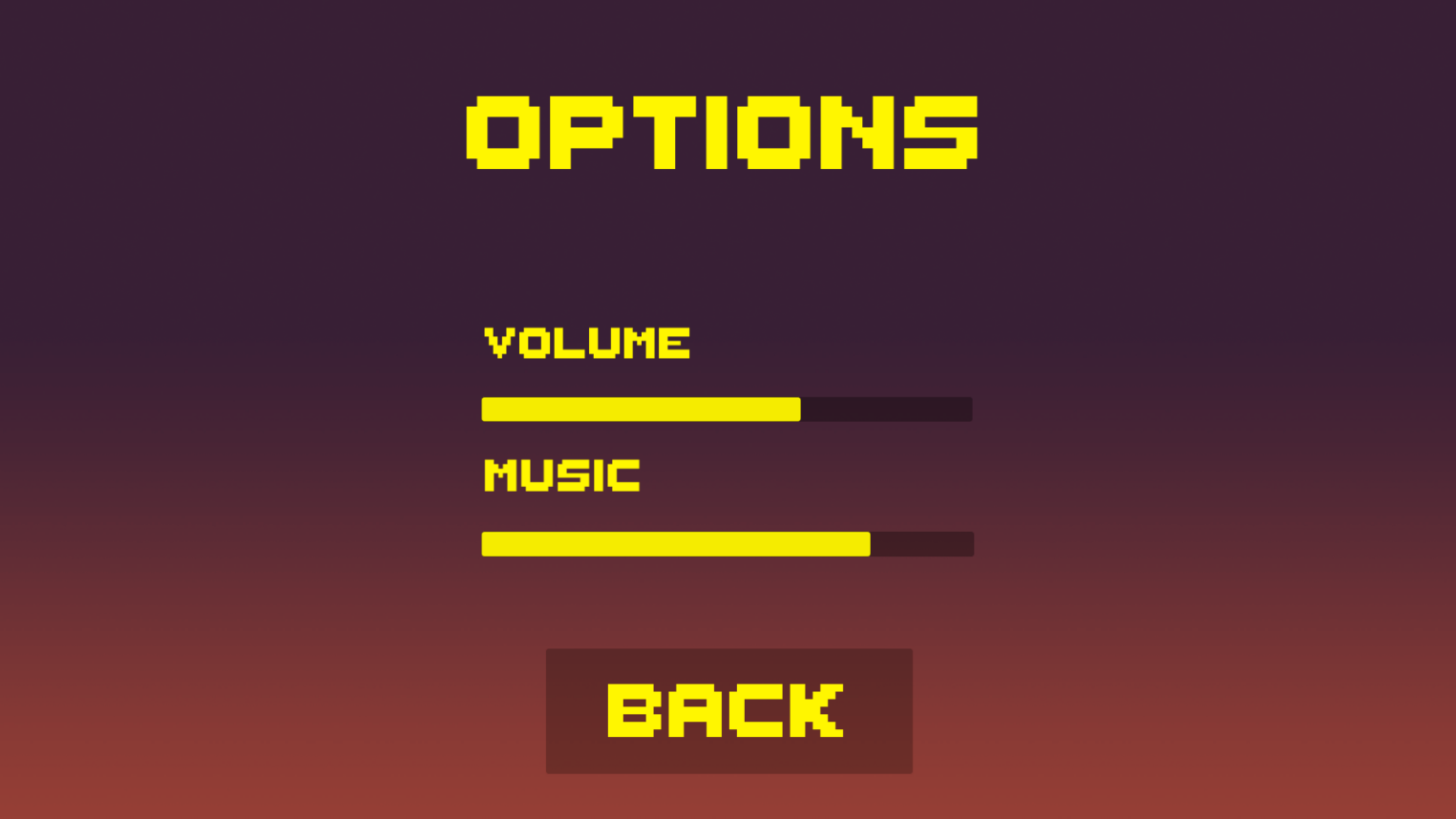


Рисунок 24 – Меню настроек

При нажатии на кнопку “Play” игроку становится доступно меню выбора персонажа (рис. 25). Персонажи имею свои уникальные параметры и умения.



Рисунок 25 – Меню выбора персонажа

После выбора персонажа начинается игровой процесс (рис. 26).



Рисунок 26 – Начало игрового процесса

Нажав клавишу “Esc” игрок попадает в меню паузы, представленную на рисунке 27.



Рисунок 27 – Меню паузы

Для каждого персонажа определена клавиша использования уникальной способности.

Для героя-мечника уникальная способность — это создание защитной сферы, которая снижает весь получаемый урон в 2 раза, действие данной способности представлено на рисунке 28.

Для героя-лучника уникальная способность позволяет запустить золотую стрелу, которая нанесет большой урон, тем самым мгновенно убив любого врага кроме босса.

Герой-маг имеет способность телепортации на короткие расстояния, что позволяет ему избегать вражеские атаки и быстрее преодолевать расстояние (рис. 29).



Рисунок 28 – Использования защиты Мечника



Рисунок 29 – Использование рывка Мага

Игровой процесс подразумевает битву с врагами (рис. 30) и в процессе игрок может получить урон от них.



Рисунок 30 – Битва с врагами

Игроку предоставлена возможность восстановить потерянное здоровье (рис. 31) при помощи склянок здоровья, выпадающих с врагов.



Рисунок 31 – Использование склянок здоровья

Если игрок не успеет восстановить потерянное здоровье его ждет смерть. (рис. 32).

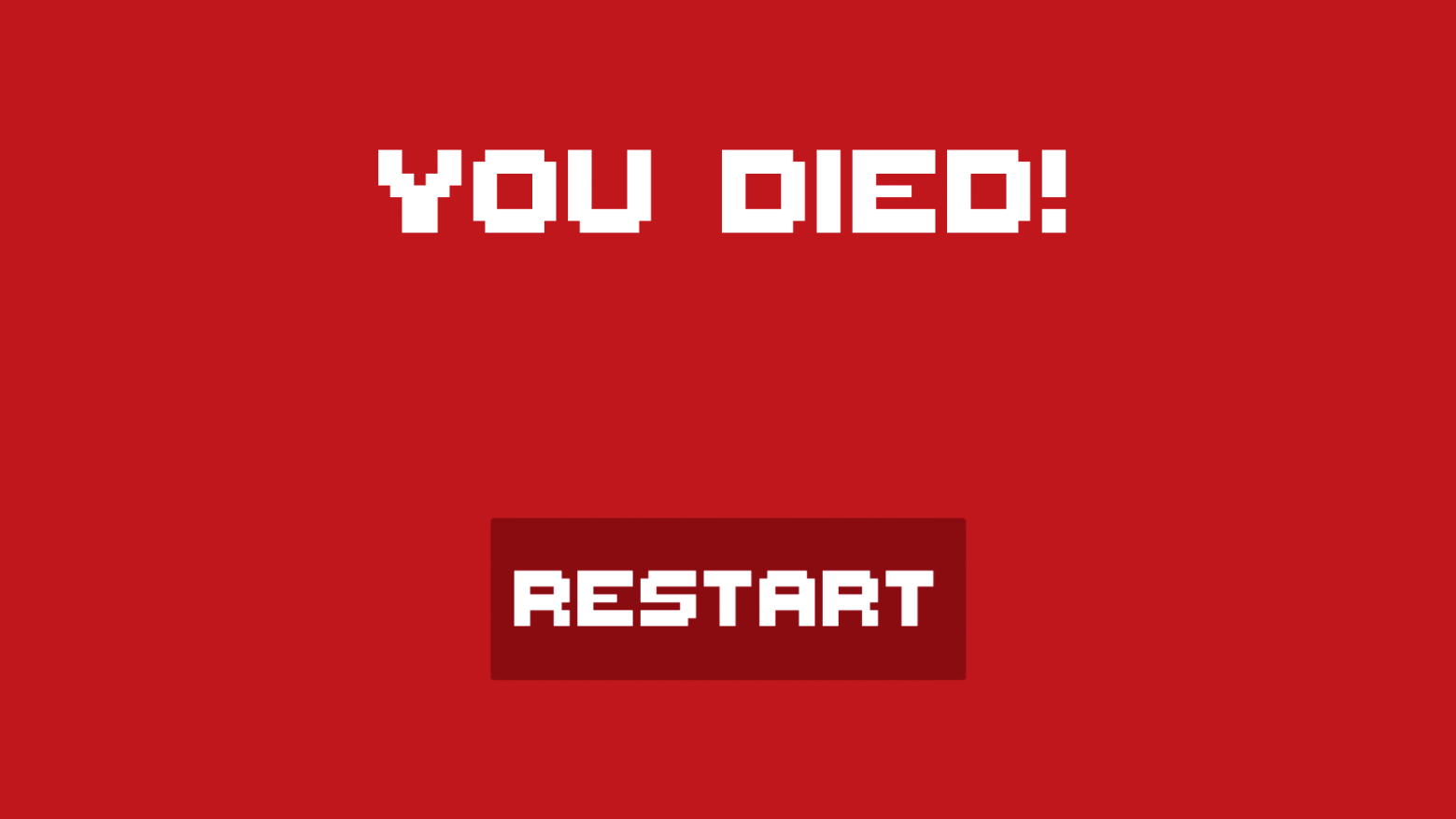


Рисунок 32 – Смерть персонажа

Помимо выпадения с врагов, монеты можно найти на уровне, данный процесс представлен на рисунке 33.



Рисунок 33 – Процесс нахождения монет

Целью игры является поиск портала, который переводит врага на следующий уровень (рис. 34).



Рисунок 34 – Портал

На финальном уровне подземелья игрока ждет битва с боссом, представленная на рисунке 35. После смерти босса закончится игра (рис. 36).



Рисунок 35 – Битва с боссом

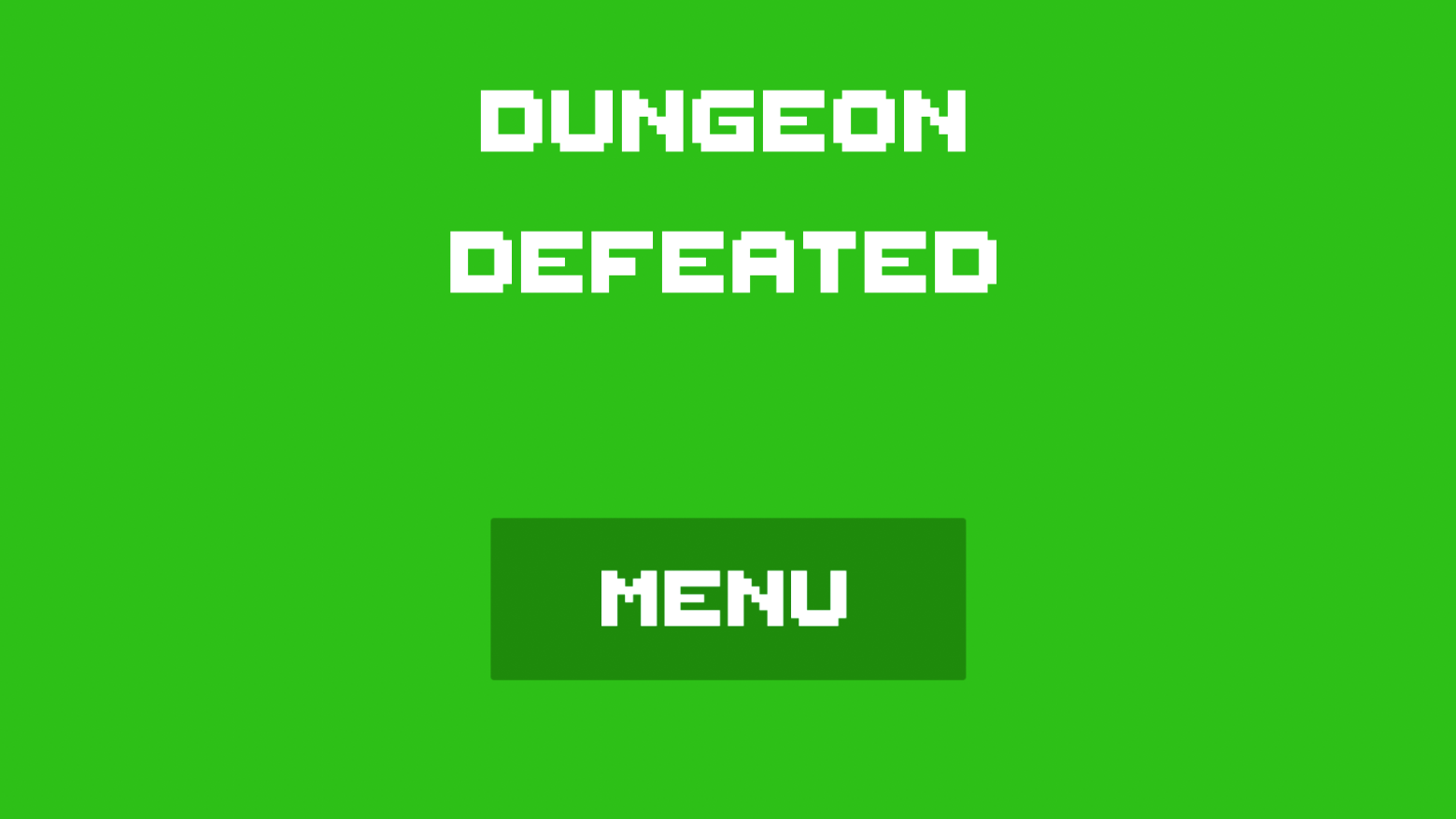


Рисунок 36 – Конец игры

3.5 Критерии качества программной системы

1. Портативность — так как данная программа реализована в среде Unity, несложно портировать игру на Android, iOS, Windows 10, Linux, Xbox, WebGL.
2. Легкость освоения — так как игра представляет собой классического представителя жанра Roguelike, управление не является сложным и в целом игра проста в освоении.
3. Производительность — данное приложение использует 2D графику в стиле Pixel Art, что положительно сказывается на производительности системы, поэтому системные требования минимальны.
4. Удобство сопровождения — данное приложение можно без особых усилий модифицировать, так как подсистемы реализованы в виде отдельных модулей.
5. Надёжность — программа полностью выполняет возложенные на неё функции и безотказно функционирует при любых характеристиках системы.

ВЫВОДЫ

В ходе выполнения курсов работы был глубже изучен современный подход к программированию на основе объектно-ориентированной технологии, были приобретены навыки написания приложения. Для достижения цели на разных этапах проектирования были решены следующие задачи:

1. Разработаны программные документы, в соответствии с действующими стандартами — оформлен отчет по работе, содержащий диаграммы, рисунки и описание работы.

2. Произведено тестирование и отладка программы — приложение не закрывается без команды пользователя, процесс игры предсказуем, скрипты не содержат синтаксических ошибок.

3. Написан текст разработанных классов на выбранном языке;

4. Абстрагированы, выделены классы и их иерархии — получены классы и иерархии, позволяющие реализовать проект согласно ПЗ.

5. Выбран вариант задания, язык реализации, детализирована постановка задачи — выбран вариант игрового проекта, среда разработки Unity и язык программирования C#.

Результатом работы стала игра в жанре Roguelike, подразумевающая процедурную генерацию уровней. На выбор игроку предоставляется три персонажа с различными способностями и параметрами. Игровой процесс заключается в уничтожении врагов, сражении с боссом и поиске перехода на следующий уровень. Игрок может подбирать предметы с врагов, такие как: монеты и склянки с зельем, восстанавливающим здоровье. HUD содержит шкалу здоровья, количество монет, мини-карту и информацию о способностях. Реализовано меню паузы с возможностью перезапуска уровня и выхода в главное в меню. Главное меню содержит следующие пункты: меню выбора персонажа, меню настроек и кнопку выхода из игры.

ПЕРЕЧЕНЬ ССЫЛОК

1. Hocking, J. Unity in Action: Multiplatform game development in C# / J. Hocking. – New York, United States : Manning Publications, 2018. – 400 с.
2. Richter, J. CLR via C# / J. Richter. – Redmond, United States : Microsoft Press, U.S., 2013. – 896 с.
3. Торн, А. Искусство создания сценариев в Unity / А. Торн. – Москва : ДМК-Пресс, 2016. – 360 с.
4. Мартин, Р. Чистый код. Создание, анализ и рефакторинг / Р. Мартин. – Санкт-Петербург : Питер, 2008. – 464 с.
5. Макконнелл, С. Совершенный код. Мастер-класс / С. Макконнелл. – Москва : Русская редакция, 2017. – 896 с.
6. Schildt, H. C# 4.0 The Complete Reference / H. Schildt. – New York, United States : McGraw-Hill Education, 2010. – 976 с.
7. Полное руководство по языку программирования С# 7.0 и платформе .NET 4.7 [Электронный ресурс]. – Режим доступа: https://metanit.com/sharp/tutorial/. – Дата доступа: 24.05.2019.

ПРИЛОЖЕНИЕ А

Код программы

Код класса PlayerBase:

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

using UnityEngine.UI;

public abstract class PlayerBase : MonoBehaviour

{

public GameObject coinText;

public GameObject hpFX;

[Header("Характеристики героя")]

public float Speed;

protected int damage;

public int minDamage;

public int maxDamage;

protected Vector2 direction;

    private Animator animator;

private Rigidbody2D rb;

private Text txt;

protected float timeBtwUseSpell;

public float startTimeBtwUseSpell;

protected Image imageCooldown;

protected bool isCooldown;

void Start()

{

animator = GetComponent<Animator>();

rb = GetComponent<Rigidbody2D>();

imageCooldown = GameObject.FindGameObjectWithTag("Cooldown").GetComponent<Image>();

txt = GameObject.FindGameObjectWithTag("NumberCoins").GetComponent<Text>();

Time.timeScale = 1;

PlayerStats.armor = 1f;

}

protected bool NeedDiagonalAdjustment()

{

return (Input.GetKey(KeyCode.W) && Input.GetKey(KeyCode.A)) ||

(Input.GetKey(KeyCode.W) && Input.GetKey(KeyCode.D)) ||

(Input.GetKey(KeyCode.S) && Input.GetKey(KeyCode.D)) ||

(Input.GetKey(KeyCode.S) && Input.GetKey(KeyCode.A));

}

protected virtual void TakeInput()

    {

direction = Vector2.zero;

if (Input.GetKey (KeyCode.W)) direction += Vector2.up\*0.9f;

if (Input.GetKey (KeyCode.A)) direction += Vector2.left;

if (Input.GetKey (KeyCode.S)) direction += Vector2.down\*0.9f;

if (Input.GetKey (KeyCode.D)) direction += Vector2.right;

if (NeedDiagonalAdjustment()) direction \*= 0.8f;

AddHP();

}

protected void Move()

    {

        transform.Translate(direction \* Speed \* Time.deltaTime);

if (direction.x != 0 || direction.y != 0) {

            AnimationMove (direction);

        } else {

            animator.SetLayerWeight (1, 0);

        }

    }

    void AnimationMove(Vector2 direction)

    {

        animator.SetLayerWeight (1, 1);

        animator.SetFloat ("x", direction.x);

        animator.SetFloat ("y", direction.y);

    }

void OnTriggerEnter2D(Collider2D other)

{

if (other.tag == "Coin")

{

FindObjectOfType<AudioManager>().Play("Coin");

int money = other.GetComponent<Coins>().money;

PlayerStats.coins += money;

coinText.GetComponent<TextMesh>().text = "+" + money.ToString();

Instantiate(coinText, other.transform.position, Quaternion.identity);

txt.text = PlayerStats.coins.ToString();

Destroy(other.gameObject);

}

}

protected void AddHP()

{

if ((Input.GetKeyDown(KeyCode.Alpha1) || Input.GetKeyDown(KeyCode.Keypad1)) && PlayerStats.numberOfBottle > 0 && Time.deltaTime != 0)

{

PlayerStats.numberOfBottle--; GameObject.FindGameObjectWithTag("MainCanvas").GetComponent<StartGame>().UpdateBottleNumber();

Vector2 hpPosition = new Vector2(transform.position.x, transform.position.y + 0.25f);

Destroy(Instantiate(hpFX, hpPosition, Quaternion.identity), 2f);

PlayerStats.HP += 10;

}

}

protected abstract void Attack();

protected void StartCooldown(float cooldown)

{

imageCooldown.fillAmount += 1 / cooldown \* Time.deltaTime;

if (imageCooldown.fillAmount >= 1)

{

imageCooldown.fillAmount = 0;

isCooldown = false;

}

}

}

Код класса PlayerStats:

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

public class PlayerStats : MonoBehaviour

{

public static PlayerStats playerStats;

[Header("Характеристики героя")]

public static float HP;

public static float armor;

public static int numberOfBottle;

public float maxHP;

public static int coins;

public static bool death;

void Awake()

{

if (playerStats !=null) {

Destroy(playerStats);

} else {

playerStats = this;

}

DontDestroyOnLoad(this);

}

void Start()

{

HP = maxHP;

}

void Update()

{

if (HP > maxHP)

HP = maxHP;

}

}

Код класса Sword:

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

public class Sword : PlayerBase

{

private float timeBtwAttack;

public float startTimeBtwAttack;

public Transform attackPos;

public LayerMask whatIsEnemies;

[Range(0,1)]

public float attackRange;

public GameObject hit;

public GameObject armorFX;

public float energy;

void Update ()

    {

Move ();

        TakeInput ();

Attack();

Energy();

Armor();

}

protected override void TakeInput()

{

direction = Vector2.zero;

if (Input.GetKey (KeyCode.W)) {

attackPos.position = new Vector2 (transform.position.x, transform.position.y+0.3f);

direction += Vector2.up\*0.9f;

}

if (Input.GetKey (KeyCode.A)) {

attackPos.position = new Vector2 (transform.position.x-0.3f, transform.position.y);

direction += Vector2.left;

}

if (Input.GetKey (KeyCode.S)) {

attackPos.position = new Vector2 (transform.position.x, transform.position.y-0.3f);

direction += Vector2.down\*0.9f;

}

if (Input.GetKey (KeyCode.D)) {

attackPos.position = new Vector2 (transform.position.x+0.3f, transform.position.y);

direction += Vector2.right;

}

if (Input.GetKey(KeyCode.W) && Input.GetKey(KeyCode.A))

attackPos.position = new Vector2 (transform.position.x-0.3f, transform.position.y+0.3f);

if (Input.GetKey(KeyCode.W) && Input.GetKey(KeyCode.D))

attackPos.position = new Vector2 (transform.position.x+0.3f, transform.position.y+0.3f);

if (Input.GetKey(KeyCode.S) && Input.GetKey(KeyCode.A))

attackPos.position = new Vector2 (transform.position.x-0.3f, transform.position.y-0.3f);

if (Input.GetKey(KeyCode.S) && Input.GetKey(KeyCode.D))

attackPos.position = new Vector2 (transform.position.x+0.3f, transform.position.y-0.3f);

if (NeedDiagonalAdjustment()) direction \*= 0.8f;

AddHP();

}

protected override void Attack()

{

if (timeBtwAttack <= 0)

{

if (Input.GetKeyDown(KeyCode.Mouse0))

{

damage = Random.Range(minDamage, maxDamage);

Collider2D[] enemiesToDamage = Physics2D.OverlapCircleAll(attackPos.position, attackRange, whatIsEnemies);

for (int i = 0; i < enemiesToDamage.Length; i++)

{

hitEffect();

if (enemiesToDamage[i].tag == "Jar")

{

if (enemiesToDamage[i].GetComponent<Jar>())

enemiesToDamage[i].GetComponent<Jar>().crash = true;

} else {

enemiesToDamage[i].GetComponent<EnemyBase>().ShowDamage(damage);

enemiesToDamage[i].GetComponent<EnemyBase>().TakeDamage(damage);

}

}

timeBtwAttack = startTimeBtwAttack;

}

} else {

timeBtwAttack -= Time.deltaTime;

}

}

void Energy()

{

if (energy > 10 && Input.GetKey(KeyCode.Space)) {

energy -= 0.5f;

Speed = 3.7f;

} else {

energy += 0.5f;

Speed = 2.5f;

}

if (energy >= 100) energy = 100;

if ((energy == 10 && Input.GetKey(KeyCode.Space)) || energy <= 0) energy = 0;

}

void Armor()

{

if (timeBtwUseSpell <= 0)

{

if (Input.GetKeyDown(KeyCode.F) && Time.timeScale == 1)

{

isCooldown = true;

PlayerStats.armor = 0.5f;

Instantiate(armorFX, transform.position, Quaternion.identity);

timeBtwUseSpell = startTimeBtwUseSpell;

}

} else {

timeBtwUseSpell -= Time.deltaTime;

}

if (isCooldown)

{

StartCooldown(10);

}

}

void hitEffect()

{

Destroy(Instantiate(hit, attackPos.position, Quaternion.identity), 0.4f);

FindObjectOfType<AudioManager>().Play("AttackSword1");

}

void OnDrawGizmosSelected()

{

Gizmos.color = Color.red;

Gizmos.DrawWireSphere(attackPos.position, attackRange);

}

}

Код класса Archer:

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

public class Archer : PlayerBase

{

private GameObject projectile;

public int damageAbility;

public float projectileForce;

private float timeBtwAttack;

public float startTimeBtwAttack;

void Update ()

    {

        TakeInput ();

        Move ();

Attack();

ArrowAbility();

}

protected override void Attack()

{

if (timeBtwAttack <= 0)

{

if (Input.GetMouseButton(0) && Time.timeScale == 1)

{

if (!GameObject.FindGameObjectWithTag("GOManager")) return;

projectile = GameObject.FindGameObjectWithTag("GOManager").GetComponent<GOManager>().bullet[3];

Vector2 mousePos = UnityEngine.Camera.main.ScreenToWorldPoint(Input.mousePosition);

Vector2 myPos = transform.position;

Vector2 direction = (mousePos - myPos).normalized;

GameObject arrow = Instantiate(projectile, transform.position, Quaternion.identity);

arrow.GetComponent<Rigidbody2D>().velocity = direction \* projectileForce;

arrow.GetComponent<Projectile>().damage = Random.Range(minDamage, maxDamage);

timeBtwAttack = startTimeBtwAttack;

}

} else {

timeBtwAttack -= Time.deltaTime;

}

}

void ArrowAbility()

{

if (timeBtwUseSpell <= 0)

{

if (Input.GetKeyDown(KeyCode.F) && Time.timeScale == 1)

{

isCooldown = true;

projectile = GameObject.FindGameObjectWithTag("GOManager").GetComponent<GOManager>().bullet[4];

Vector2 mousePos = UnityEngine.Camera.main.ScreenToWorldPoint(Input.mousePosition);

Vector2 myPos = transform.position;

Vector2 direction = (mousePos - myPos).normalized;

GameObject arrowAbility = Instantiate(projectile, transform.position, Quaternion.identity);

arrowAbility.GetComponent<Rigidbody2D>().velocity = direction \* projectileForce;

arrowAbility.GetComponent<Projectile>().damage = damageAbility;

timeBtwUseSpell = startTimeBtwUseSpell;

}

} else {

timeBtwUseSpell -= Time.deltaTime;

}

if (isCooldown)

{

StartCooldown(5);

}

}

}

Код класса EnemyBase:

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

using UnityEngine.UI;

public abstract class EnemyBase : MonoBehaviour

{

private SpriteRenderer sr;

protected Animator anim;

protected Transform target;

public GameObject damageText;

public GameObject explosion;

public Slider sl;

private Shake shake;

[Header("Характеристики врага")]

public float HP;

public float speed;

public float damage;

public int maxDrop;

protected virtual void Start()

{

shake = GameObject.FindGameObjectWithTag("ShakeManager").GetComponent<Shake>();

anim = GetComponent<Animator>();

sr = GetComponent<SpriteRenderer>();

target = GameObject.FindGameObjectWithTag("Player").GetComponent<Transform>();

sl.maxValue = HP;

sl.value = HP;

}

protected abstract void Following();

protected abstract void Attack();

protected void FlipX(float other)

{

if (other < transform.position.x) {

sr.flipX=true;

} else {

sr.flipX=false;

}

}

public void TakeDamage(int damage)

{

HP -= damage;

sl.value = HP;

}

public void ShowDamage(int damage)

{

damageText.GetComponent<TextMesh>().text = "-" + damage;

Instantiate(damageText, transform.position, Quaternion.identity);

}

protected void CheckDeath()

{

if (HP<=0) {

shake.CamShake();

FindObjectOfType<AudioManager>().Play("EnemyDeath");

Destroy(Instantiate(explosion, transform.position, Quaternion.identity), 0.45f);

int numberDrop = Random.Range(0, maxDrop);

GameObject drop = GameObject.FindGameObjectWithTag("GOManager").GetComponent<GOManager>().drop[numberDrop];

Instantiate(drop, transform.position, Quaternion.identity);

Destroy(gameObject);

}

}

}

Код класса EnemyProjectile:

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

public class EnemyProjectile : MonoBehaviour

{

[HideInInspector]

public float damage;

public float speed;

private Transform player;

private Vector2 target;

void Start()

{

if (!GameObject.FindGameObjectWithTag("Player")) return;

player = GameObject.FindGameObjectWithTag("Player").transform;

target = new Vector2(player.position.x, player.position.y);

}

void Update()

{

transform.position = Vector2.MoveTowards(transform.position, target, speed \* Time.deltaTime);

if (transform.position.x == target.x && transform.position.y == target.y)

Destroy(gameObject);

}

void OnTriggerEnter2D(Collider2D other)

{

if (other.tag == "Wall")

{

Destroy(gameObject);

}

if (other.tag == "Player")

{

PlayerStats.HP -= damage;

Destroy(gameObject);

}

}

}

Код класса MeleeEnemy:

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

public class MeleeEnemy : EnemyBase

{

[Range(0,5)]

public float seeDistance;

[Range(0,5)]

public float attackDistance;

private float waitTime, waitTime2;

public float startWaitTime, startWaitTime2;

protected bool runToRandom = false;

protected float rndX, rndY;

void Update()

{

Following();

CheckDeath();

}

protected override void Following()

{

if (Vector2.Distance (transform.position, target.transform.position) < seeDistance)

{

anim.SetBool("Running", true);

runToRandom=false;

waitTime=1;

speed = 1.8f;

FlipX(target.position.x);

Attack();

} else {

anim.SetBool("Running", false);

speed = 1.2f;

if (waitTime <= 0) {

rndX= Random.Range(transform.position.x-5f, transform.position.x+5f);

rndY= Random.Range(transform.position.y-5f, transform.position.y+5f);

FlipX(rndX);

runToRandom=true;

waitTime=startWaitTime;

} else {

waitTime -= Time.deltaTime;

}

}

if (runToRandom)

{

transform.position = Vector2.MoveTowards(transform.position, new Vector2 (rndX, rndY), speed \* Time.deltaTime);

anim.SetBool("Running", true);

if (waitTime2 <= 0) {

anim.SetBool("Running", false);

runToRandom=false;

waitTime2=startWaitTime2;

} else {

waitTime2 -= Time.deltaTime;

}

}

}

protected override void Attack()

{

if (Time.timeScale != 0) {

if (Vector2.Distance (transform.position, target.transform.position) > attackDistance)

{

transform.position = Vector2.MoveTowards(transform.position, target.position, speed \* Time.deltaTime);

} else {

PlayerStats.HP -= damage \* PlayerStats.armor;

}

}

}

void OnDrawGizmosSelected()

{

Gizmos.color = Color.green;

Gizmos.DrawWireSphere(transform.position, seeDistance);

Gizmos.color = Color.red;

Gizmos.DrawWireSphere(transform.position, attackDistance);

}

}

Код класса RangeEnemy:

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

public class RangeEnemy : EnemyBase

{

[Range(0,7)]

public float seeDistance;

[Range(0,5)]

public float stopDistance;

[Range(0,3)]

public float retreatDistance;

public GameObject projectile;

private float timeBtwAttack;

public float startTimeBtwAttack;

void Update()

{

Following();

CheckDeath();

}

protected override void Following()

{

if (Vector2.Distance (transform.position, target.transform.position) < seeDistance)

{

FlipX(target.position.x);

Attack();

if (Vector2.Distance(transform.position, target.transform.position) > stopDistance) {

anim.SetBool("Running", true);

transform.position = Vector2.MoveTowards(transform.position, target.transform.position, speed \* Time.deltaTime);

}

else if (Vector2.Distance(transform.position, target.transform.position) < stopDistance &&

Vector2.Distance(transform.position, target.transform.position) > retreatDistance) {

anim.SetBool("Running", false);

transform.position = this.transform.position;

}

else if (Vector2.Distance(transform.position, target.transform.position) < retreatDistance) {

anim.SetBool("Running", true);

transform.position = Vector2.MoveTowards(transform.position, target.transform.position, -speed \* Time.deltaTime);

}

} else {

anim.SetBool("Running", false);

}

}

protected override void Attack()

{

if (timeBtwAttack <= 0) {

Instantiate(projectile, transform.position, Quaternion.identity);

projectile.GetComponent<EnemyProjectile>().damage = damage \* PlayerStats.armor;

timeBtwAttack = startTimeBtwAttack;

} else {

timeBtwAttack -= Time.deltaTime;

}

}

void OnDrawGizmosSelected()

{

Gizmos.color = Color.green;

Gizmos.DrawWireSphere(transform.position, retreatDistance);

Gizmos.color = Color.red;

Gizmos.DrawWireSphere(transform.position, stopDistance);

Gizmos.color = Color.white;

Gizmos.DrawWireSphere(transform.position, seeDistance);

}

}

Код класса Boss:

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

public class Boss : EnemyBase

{

    public GameObject projectile;

    public float radius;

public float projectileSpeed;

    private int numberOfProjectiles;

    protected override void Start()

    {

        base.Start();

        Attack();

    }

void Update()

{

Following();

CheckDeath();

}

    IEnumerator SpawnProjectiles()

    {

        while(true)

        {

            numberOfProjectiles = Random.Range(5, 12);

            float angleStep = 360f / numberOfProjectiles;

            float angle = 0f;

            for (int i = 0; i <= numberOfProjectiles - 1; i++)

            {

                float projectileDirXposition = transform.position.x + Mathf.Sin ((angle \* Mathf.PI) / 180) \* radius;

                float projectileDirYposition = transform.position.y + Mathf.Cos ((angle \* Mathf.PI) / 180) \* radius;

                Vector2 projectileVector = new Vector2 (projectileDirXposition, projectileDirYposition);

                Vector2 tp = new Vector2 (transform.position.x, transform.position.y);

                Vector2 projectileMoveDirection = (projectileVector - tp).normalized \* projectileSpeed;

                var proj = Instantiate (projectile, transform.position, Quaternion.identity);

                projectile.GetComponent<ProjectileBoss>().damage = damage \* PlayerStats.armor;

                proj.GetComponent<Rigidbody2D> ().velocity = new Vector2 (projectileMoveDirection.x, projectileMoveDirection.y);

                angle += angleStep;

            }

            yield return new WaitForSeconds(3f);

        }

    }

protected override void Attack() {

StartCoroutine(SpawnProjectiles ());

}

protected override void Following() {

//движение босса

}

}

Код класса ProjectileBoss:

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

public class ProjectileBoss : MonoBehaviour

{

[HideInInspector]

public float damage;

public GameObject explosion;

void Start()

{

Destroy(gameObject, 4f);

}

void OnTriggerEnter2D(Collider2D other)

{

if (other.tag == "Wall")

{

Destroy(Instantiate(explosion, transform.position, Quaternion.identity), 1f);

Destroy(gameObject);

}

if (other.tag == "Player")

{

Destroy(Instantiate(explosion, transform.position, Quaternion.identity), 1f);

PlayerStats.HP -= damage;

Destroy(gameObject);

}

}

}

Код класса RoomTemplates:

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

public class RoomTemplates : MonoBehaviour

{

    public GameObject[] bottomRooms;

    public GameObject[] topRooms;

    public GameObject[] leftRooms;

    public GameObject[] rightRooms;

    public GameObject closedRoom;

    public List<GameObject> rooms;

    public float waitTime;

    private bool spawnedBoss;

    public GameObject boss;

    void Update()

    {

        if(waitTime <= 0 && spawnedBoss == false) {

            for (int i = 0; i < rooms.Count; i++)

            {

                if(i == rooms.Count-1) {

                    Instantiate(boss, rooms[i].transform.position, Quaternion.identity);

                    spawnedBoss = true;

                }

            }

        } else {

            waitTime -= Time.deltaTime;

        }

    }

}

Код класса RoomSpawner:

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

public class RoomSpawner : MonoBehaviour

{

public int openingDirection;

private RoomTemplates templates;

private int rand;

private bool spawned = false;

public float waitTime = 4f;

private void Start()

{

Destroy(gameObject, waitTime);

templates = GameObject.FindGameObjectWithTag("Rooms").GetComponent<RoomTemplates>();

Invoke("Spawn", 0.2f);

}

private void Spawn()

{

if (spawned == false)

{

if (openingDirection == 1)

{

rand = Random.Range(0, templates.bottomRooms.Length);

Instantiate(templates.bottomRooms[rand], transform.position, templates.bottomRooms[rand].transform.rotation);

}

else if (openingDirection == 2)

{

rand = Random.Range(0, templates.topRooms.Length);

Instantiate(templates.topRooms[rand], transform.position, templates.topRooms[rand].transform.rotation);

}

else if (openingDirection == 3)

{

rand = Random.Range(0, templates.leftRooms.Length);

Instantiate(templates.leftRooms[rand], transform.position, templates.leftRooms[rand].transform.rotation);

}

else if (openingDirection == 4)

{

rand = Random.Range(0, templates.rightRooms.Length);

Instantiate(templates.rightRooms[rand], transform.position, templates.rightRooms[rand].transform.rotation);

}

spawned = true;

}

}

private void OnTriggerEnter2D(Collider2D other)

{

if (other.CompareTag("SpawnPoint"))

if (other.GetComponent<RoomSpawner>().spawned == false && spawned == false)

{

Instantiate(templates.closedRoom, transform.position, Quaternion.identity);

Destroy(gameObject);

}

spawned = true;

}

}

Код класса Camera:

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

public class Camera : MonoBehaviour

{

private Transform player;

public float smoothing;

public Vector3 offset;

void Start()

{

player = GameObject.FindGameObjectWithTag("Player").GetComponent<Transform>();

}

void FixedUpdate()

{

if (player!=null)

{

Vector3 newPos = Vector3.Lerp(transform.position, player.transform.position + offset, smoothing);

transform.position = newPos;

}

}

}

Код класса TextDamage:

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

public class TextDamage : MonoBehaviour

{

private Vector2 rnd;

void Start()

{

Destroy(gameObject, 1f);

rnd = new Vector2(Random.Range(-0.5f, 0.5f), Random.Range(-0.5f, 0.5f));

}

void Update()

{

transform.Translate(rnd \* Time.deltaTime);

}

}

Код класса SpawnCharacter:

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

public class SpawnCharacter : MonoBehaviour

{

public static SpawnCharacter character;

public Transform spawnHero;

public static int numberChooseHero;

public GameObject sword;

public GameObject archer;

public GameObject wizard;

void Awake()

{

if (character !=null)

{

Destroy(character);

}

else

{

character = this;

}

DontDestroyOnLoad(this);

switch (numberChooseHero)

{

case 1:

Instantiate(sword, spawnHero.transform.position, Quaternion.identity);

break;

case 2:

Instantiate(archer, spawnHero.transform.position, Quaternion.identity);

break;

case 3:

Instantiate(wizard, spawnHero.transform.position, Quaternion.identity);

break;

}

}

}

Код класса StartGame:

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

using UnityEngine.UI;

using UnityEngine.SceneManagement;

public class StartGame : MonoBehaviour

{

public GameObject death;

public GameObject pause;

public GameObject map;

public GameObject abilityPanel;

public Sprite[] img;

public GameObject myImg;

public Text numberBottleText;

public Text numberCoins;

void Awake()

{

PlayerStats.death=false;

}

void Start()

{

switch (Character.numberChooseHero)

{

case 1:

myImg.GetComponent<Image>().sprite = img[0];

break;

case 2:

myImg.GetComponent<Image>().sprite = img[1];

break;

case 3:

myImg.GetComponent<Image>().sprite = img[2];

break;

}

numberBottleText.text = PlayerStats.numberOfBottle.ToString();

numberCoins.text = PlayerStats.coins.ToString();

}

void Update()

{

if (PlayerStats.HP <= 0 && !PlayerStats.death)

{

GameObject.FindGameObjectWithTag("Player").SetActive(false);

FindObjectOfType<AudioManager>().Play("PlayerDeath");

map.SetActive(false);

abilityPanel.SetActive(false);

death.SetActive(true);

PlayerStats.death = true;

}

if (Input.GetKeyDown(KeyCode.Escape))

{

if (Time.timeScale == 0)

{

pause.SetActive(false);

Time.timeScale = 1;

} else if (Time.timeScale == 1) {

pause.SetActive(true);

Time.timeScale = 0;

}

}

}

public void RestartGame()

{

StatsNull();

if (GameObject.FindGameObjectWithTag("Player"))

GameObject.FindGameObjectWithTag("Player").SetActive(false);

SceneManager.LoadScene(SceneManager.GetActiveScene().buildIndex);

}

public void BackToMenu()

{

StatsNull();

SceneManager.LoadScene("Menu");

}

public void ExitTheGame()

{

Application.Quit();

}

public void ClosePanel()

{

pause.SetActive(false);

Time.timeScale = 1;

}

public void UpdateBottleNumber()

{

numberBottleText.text = PlayerStats.numberOfBottle.ToString();

}

void StatsNull()

{

PlayerStats.numberOfBottle = 0;

PlayerStats.coins = 0;

}

}

Код класса Menu:

using System.Collections;

using UnityEngine;

using UnityEngine.UI;

using UnityEngine.SceneManagement;

public class Menu : MonoBehaviour

{

public GameObject loadingScreen;

public GameObject hero;

public Slider slider;

public Text loadingText;

void Awake()

{

Destroy (GameObject.FindGameObjectWithTag("Player"));

Destroy (GameObject.FindGameObjectWithTag("Event"));

}

public void PlayGame()

{

SceneManager.LoadScene(SceneManager.GetActiveScene().buildIndex + 1);

}

public void QuitGame()

{

Application.Quit();

}

public void ChooseSword()

{

Character.numberChooseHero = 1;

}

public void ChooseArcher()

{

Character.numberChooseHero = 2;

}

public void ChooseWizard()

{

Character.numberChooseHero = 3;

}

public void LoadLevel()

{

if(Character.numberChooseHero > 0 && Character.numberChooseHero < 4)

StartCoroutine(LoadAsynchronously());

}

IEnumerator LoadAsynchronously ()

{

AsyncOperation operation = SceneManager.LoadSceneAsync(SceneManager.GetActiveScene().buildIndex + 1);

hero.SetActive(false);

loadingScreen.SetActive(true);

while (!operation.isDone)

{

float progress = Mathf.Clamp01(operation.progress / 0.9f);

slider.value = progress;

loadingText.text = progress \* 100f + "%";

yield return null;

}

}

}