**Содержание**

[**ВВЕДЕНИЕ** 3](#_Toc98702672)

[1 Техническое обоснование проекта 5](#_Toc98702673)

[2 Формирование технического задания 6](#_Toc98702674)

[2.1 Условия применения системы 6](#_Toc98702675)

[3 Создание пользовательского интерфейса 8](#_Toc98702676)

[3.1 Реализация пользовательского интерфейса 8](#_Toc98702677)

[3.2 Создание программного продукта 12](#_Toc98702678)

[4 Тестирование 17](#_Toc98702679)

[**ЗАКЛЮЧЕНИЕ** 24](#_Toc98702680)

[**СПИСОК ИСПЛЬЗОВАНННОЙ ЛИТЕРАТУРЫ** 25](#_Toc98702681)

[**ПРИЛОЖЕНИЕ** 26](#_Toc98702682)

# **ВВЕДЕНИЕ**

Разработка компьютерных игр - процесс создания компьютерных игр. Разработкой видеоигр занимается разработчик, который может быть представлен как одним человеком, так и фирмой. Обычно крупномасштабные коммерческие игры разрабатываются командами разработчиков в пределах компании, специализирующейся на играх для персонального компьютера или консолей. Как правило, разработку финансирует другая, более крупная компания-издатель, которая по окончанию разработки занимается изданием игры и связанными с ним тратами.

Игровой движок – программный компонент, предоставляющий разработчикам инструментарий для создания и запуска игры.

Что бы не трать время на разработку основы для игры, таких как игровой движок, разработчики используют готовые решения.

В этом проекте было проведено исследование и путем анализа был выбран игровой движок Unity.

Платформа Unity обладает широким спектром возможностей, удобным и интуитивно понятны интерфейсом. Большим преимуществом Unity является кроссплатформенная разработка, которая делает легким и быстрым портирование игр под такие платформы как Windows, iOS, Android, Windows Phone 8, а также разрабатывать игры для Xbox, PS, Nintendo, веб-броузеров и даже VR

С точки зрения организации, данные игры с простым содержанием и увлекательным игровым процессом помогают организациям в удержании аудитории на сайте или приложении на любой платформе.

Такие игры могут быть внедрены в любой существующий проект что делает данную тему актуальной.

Все это в совокупности и обусловило определение цели, задач, объекта и предмета данной работы.

Цель: Создание программного продукта «Компьютерная игра на движке Unity»

Задачи:

-исследовать предметную область;

-построить информационно-логическую модель с использованием СУБД;

-создать пользовательский интерфейс;

-разработать формы для внесения информации;

-разработать визуальный стиль игры;

-разработать методы управления игровым персонажем;

-протестировать игру;

-разработать справочники, документы, отчеты и другие объекты конфигурации для данного проекта.

Объектом исследования является компьютерные игры.

Предметом исследования является влияние игрового процесса на получаемые эмоции от игры.

# **1 Техническое обоснование проекта**

Игра предназначена для зарегистрированных пользователей из любой точки мира. Данная игра предоставляет несколько интересных механик и увлекательный игровой процесс, а также возможность просматривать свой счет и счет других игроков.

Проект возник вследствие изучения ситуации на игровом рынке. В ходе исследования были изучены проекты «Vampire Survivors, Noita». В ходе изучения были выявлены достоинства и недостатки. (См. Табл. 1)

Таблица 1 – Объекты исследования

|  |  |  |
| --- | --- | --- |
|  | «Vampire Survivors» | «Noita» |
| Достоинства | - Приятный визуальный стиль.  - Простой игровой процесс | - Приятная графика и яркие цвета.  - Интересная механика игрового процесса.  - Высокая реиграбельность |
| Недостатки | - Редкие обновления  - Плохая оптимизация | - Высокий порог вхождения  - Без информации из вне не разобраться |

# **2 Формирование технического задания**

# **2.1 Условия применения системы**

Системы запускается под управлением операционной системы Windows.

Системные требования при установке .NET Framework (См. Табл. 2)

Таблица 2 – Системные требования .Net

|  |  |
| --- | --- |
|  | Требования |
| Процессор | 1 ГГц |
| ОЗУ | 512 Мб |
| |  |  | | --- | --- | | Минимальный объем дискового пространства 32-разрядная ОС | 4,5 ГБ | |  |  | | 4,5 ГБ |
| |  |  | | --- | --- | | Минимальный объем дискового пространства 64-разрядная ОС |  | | 4,5 ГБ |

**Поддерживаемые операционные системы:** Windows XP и выше.

Системные требования, предъявляемые при установке OpenServer (См. Табл. 3)

Таблица 3 – Системные требования Open Server

|  |  |
| --- | --- |
|  | Требования |
| Процессор | 1 ГГц |
| Минимальный объем дискового пространства 64-разрядная ОС | 5 Гб |
| ОЗУ | 500 Мб |

Windows 7 SP1 x64 / Windows Server 2008 R2 SP1 или новее (64 битные ОС);

Для запуска игры требуется 200 Мб свободного дискового пространства. 500 МБ ОЗУ. Подключение интернета.

Правила инсталляции системы.

Для инсталляции системы на сервере необходимо:

Установить программу OpenServer;

Включить компонент My SQL;

Импортировать БД В PhpMyAdmin.

Для установки игры требуется:

Установить платформу .NET;

Разархивировать игру в любую папку.

# **Сценарий взаимодействия с пользователем**

Взаимодействие с пользователем с системой происходит через интерфейсное меню

Состав скомпилированного проекта:

* 2dGame\_Data
* MonoBleedingEdge
* 2dGame.exe
* UnityCrashHandler64.exe
* UnityPlayer.dll
* WinPixEventRuntime.dll

Все эти файлы находятся в папке «Game»

Файлы БД

- players.sql

# **3 Создание пользовательского интерфейса**

Для игры приятный и понятный интерфейс не менее важен чем сама игра. Поэтому к нему есть требования:

- Легкость использования;

- Доступность;

- Проверка входных данных;

- Низкие требования к оборудованию.

# **3.1 Реализация пользовательского интерфейса**

При открытии игры игрок будет видеть окно авторизации, в которое нужно ввести логин или пароль (См. Рис 1). Если игрок не зарегистрирован, то снизу указана кнопка, которая переведет на сайт для регистрации аккаунта (См. Рис. 2).

Изображение выглядит как текст

Автоматически созданное описание

Рисунок 1 - Окно авторизации

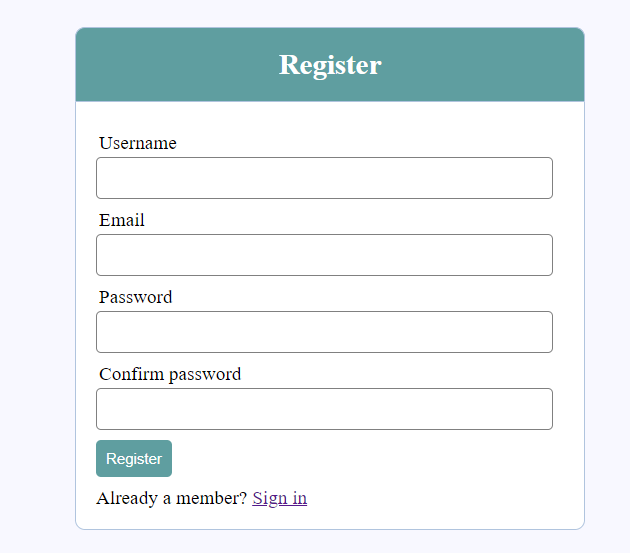


Рисунок 2 - Регистрация на сайте

При введении данных от аккаунта, игрок будет перемещен на главную страницу игры (См. Рис. 3)

Изображение выглядит как текст

Автоматически созданное описание

Рисунок 3 - Главное меню

При нажатии на кнопку Play игрок сможет начать играть в саму игру (См. Рис. 4).

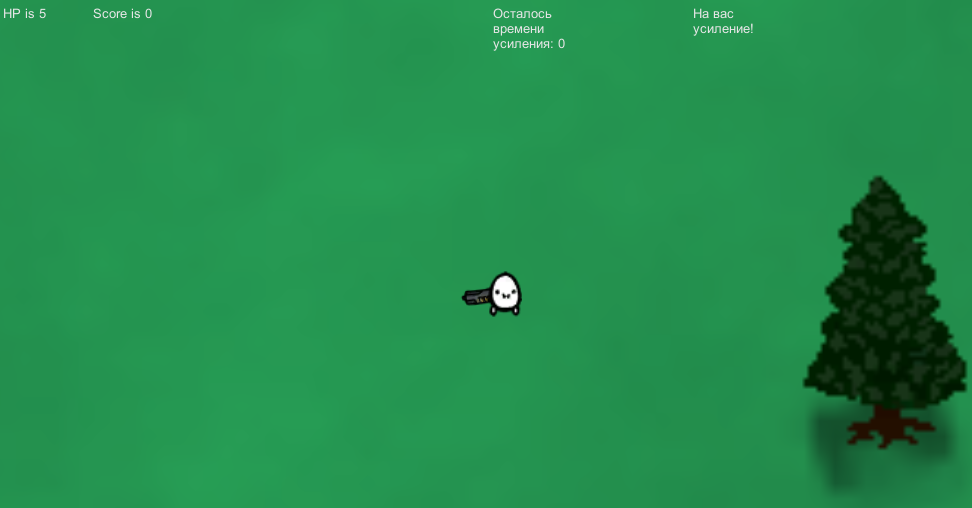


Рисунок 4 - Игровой процесс

При нажатии на кнопку Settings игрок будет перемещен на страницу настроек игры (См. Рис. 5).

Изображение выглядит как текст

Автоматически созданное описание

Рисунок 5 - Настройки игры

При получении смертельного урона игрок будет перенесен в меню проигрыша, где можно будет начать заново (См. Рис 6).

Изображение выглядит как текст

Автоматически созданное описание

Рисунок 6 - Меню проигрыша

# **3.2 Создание программного продукта**

Основные компоненты проекта в Unity, использованные в программе:

Sprite Renderer – Один из важнейший компонентов, который используется в каждом объекте игры. Он нужен для того, чтобы обрисовывать спрайты на объектах (См. Рис. 7).

Transform – Так же обязательный компонент в каждом объекте, благодаря ему можно определять положение объекта в игре (См. Рис. 8).

Rigidbody2d – Требуется для придании объекту физических качеств, таких как масса и сила тяжести (См. Рис. 9).

Collider – Требуется для отслеживания столкновений в игре, на нем строятся почти все механики (См. Рис. 10).

Script – Файл, в котором пишется код, который после прикрепляется к объекту. (См. Рис. 11).

Animator – позволяет привязывать к объекту анимацию (См. Рис 12).

Все эти компоненты изображены ниже.

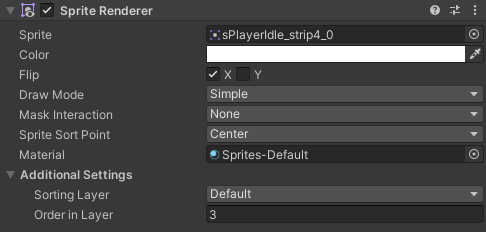


Рисунок 7 - Компонент Sprite Renderer

Изображение выглядит как текст, устройство, счетчик, табло

Автоматически созданное описание

Рисунок 8 - Компонент Transform

Изображение выглядит как текст

Автоматически созданное описание

Рисунок 9 - Компонент Rigidbody2d

Изображение выглядит как текст, дорога, электроника, снимок экрана

Автоматически созданное описание

Рисунок 10 - Компонент Collider

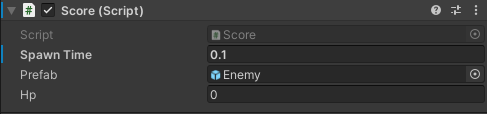


Рисунок 11 - Компонент Script

Изображение выглядит как текст

Автоматически созданное описание

Рисунок 12 - Компонент Animator

Для создания базы данных была выбрана программа PhpMyAdmin, которая позволяет управлять, редактировать, соединять различные таблицы в БД. Так же внутри есть удобный интерфейс и редактор SQL запросов. База данных игры находится на сервере что обеспечивает достойный уровень безопасности.

Игру можно запускать на нескольких компьютерах одновременно, и все они будут подключены к БД (См. Рис. 13)

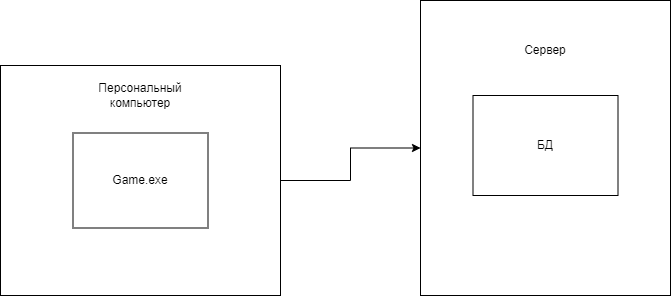


Рисунок 13 - Диаграмма развертывания

Для работы с БД со стороны клиента используется метод WWW и WWWForms (См. Рис. 14) благодаря которому происходит подключение к PHP файлу на сервере (См. рис. 15)

Изображение выглядит как текст

Автоматически созданное описание

Рисунок 14 - Отправка запроса и получение ответа

Изображение выглядит как текст

Автоматически созданное описание

Рисунок 15 - Обработка запроса со стороны сервера

При появлении противник получает координаты игрока, после чего он будет изменять свое положение для сближения с игроком. Для того что бы найти игрока используется методы FindGameObjectWithTag() и GetComponent<>. Первый метод нужен для нахождения объекта с тегом Player, а второй для того, чтобы получить координаты объекта.

Для того что бы противник начал двигаться, используется метод Vector2.MoveTowards который двигает противника к цели. (См. Рис 16).

Изображение выглядит как текст

Автоматически созданное описание

Рисунок 16 - Реализация противника

При появлении персонажа на поле им можно управлять стрелками на клавиатуре или кнопками W, A, S, D. Для того что бы двигать персонажа был использован Rigidbody2d и Vector2. Для того что бы понять какую кнопку нажимает игрок используется метод Input.GetAxisRaw(). Так как игрок — это физический объект, для его движение используется rb.MovePosition() в котором происходят вычисления. (См. Рис 17)

Изображение выглядит как текст

Автоматически созданное описание

Рисунок 17 - Реализация движения

# **4 Тестирование**

Для проверки работоспособности игрового приложения было проведено функциональное тестирование и юзабилити-тестирование.

Функциональное тестирование – это тестирование ПО в целях проверки реализуемости функциональных требований, то есть способности ПО в определенных условиях решать задачи, нужные пользователям.

Тесть № 1 Авторизация в игре.

Входные данные: Игрок вводит логин и пароль для авторизации.

Ожидаемый результат: Неудачная попытка входа, Успешная попытка входа.

Полученный результат: совпадает с ожидаемым (См. Рис. 18)

Тест успешно пройден.



Рисунок 18 - Тест Авторизации

Тест № 2 Перемещение и стрельба

Входные данные: игрок управляет персонажем перемещает персонажа и стреляет в выбранном направлении.

Ожидаемый результат: Персонаж перемещается и стреляет в выбранном направлении.

Полученный результат: совпадает с ожидаемым (См. Рис. 19)

Тест успешно пройден.



Рисунок 19 - Тест Перемещение и стрельба

Тест №3 Получение урона от противника

Входные данные: игрок сталкивается с одним или несколькими противниками.

Ожидаемый результат: Уровень здоровья персонажа снизился на количество задетых противников.

Полученный результат: совпадает с ожидаемым (См. Рис 20)



Рисунок 20 - Тест Получение урона

Тест № 4 Попадание пулей по врагу.

Входные данные: Игрок стреляет в противника.

Ожидаемый результат: Выпущенный снаряд сталкивается с противном, убивая его, на счет игрока поступает 10 очков.

Полученный результат: совпадает с ожидаемым (См. Рис 21).



Рисунок 21 - Тест Убийство Противника

Тест № 5 Получение усиления

Входные данные: Игрок подходит к усилению, тем самым подбирая его.

Ожидаемый результат: Игрок получил усиление в виде здоровья, скорости, щита.

Полученный результат: совпадает с ожидаемым (См. Рис. 22,23)

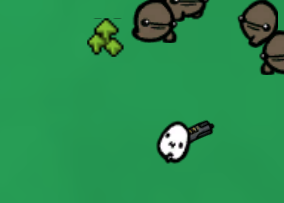


Рисунок 22 - Тест Игрок нашел усиление

Изображение выглядит как текст

Автоматически созданное описание

Рисунок 23 - Тест игрок получил усиление

Тест № 6 Изменение количества противников

Входные данные: Игрок пытается изменить количество противников в настройках

Ожидаемый результат: Игрок вписывает значение и изменяет глобальную переменную

Полученный результат: совпадает с ожидаемым (См. Рис. 24)

Изображение выглядит как текст

Автоматически созданное описание

Рисунок 24 - Тест Количество противников

Юзабилити-тестирование (проверка эргономичности) – метод оценки удобства продукта в использовании, основанный на привлечении пользователей в качестве испытателей и суммировании полученных от них выводов. При прохождении данного теста были привлечены три пользователя. Тестировщикам было предложено решить следующие задачи:

-авторизироваться;

-начать игру;

-убить несколько противников;

-изменить количество противников;

-получить усиление;

-проверить свой результат после игры.

Все задачи были решены всеми тестировщиками без затруднений. Пользователи отметили интуитивность управления и правил игры. Юзабилити-тестирование прошло успешно.

# **ЗАКЛЮЧЕНИЕ**

В результате выполнения курсового проекта была разработана игра на движке Unity. Был разработан интерфейс для пользователя. В котором можно авторизироваться под своей учетной записью, а также возможностью настройки игры под себя. Реализованный игровой процесс простой в изучении и понимании для пользователя.

Было произведено тестирование, в котором принимали участие студенты. Тестирование не показало неработающих и неудобных объектов в игре. Все пользователи смогли выполнить список поставленных задач.

Так как все поставленные задачи были выполнены и прошли надлежащее тестирование доказывает, что цель курсового проекта выполнена.

В ходе работы над курсовым проектом была изучена предметная область игростроения, освоены принципы создания игр и создание сайта для игры. Созданная база данных позволит быстро проверить информацию о игроках и их пожеланиях.

Игра содержит в себе простой и интересный игровой процесс, понятный в использовании графический интерфейс с меню настроек и окном авторизации. Все это позволит привлекать больше аудитории к самой игре и смежных с ней проектами.

Создание базы данных было произведено при помощи MySQL и PhpMyAdmin. В ходе разработки были использованы все основные средства управления и отображения информации.

Игра разработана так, что в будущем есть возможность реализовать новые функции, доработать существующие.

# **СПИСОК ИСПЛЬЗОВАНННОЙ ЛИТЕРАТУРЫ**

1. Unity Manual 2020.3 URL: Документация по Unity

Автор: Unity Technologies. Дата публикации 2022-03-19.

(Дата обращения 15.02.2022)

Ссылка: <https://docs.unity3d.com/Manual/index.html>

1. Original MySQL API URL: Документация по MySQL

Автор: [The PHP Group](https://www.php.net/copyright.php)

(Дата обращения 22.02.2022)

Ссылка: <https://www.php.net/manual/en/book.mysql.php>

1. MySQL Improved Extension URL: Документация по MySQLi

Автор: [The PHP Group](https://www.php.net/copyright.php)

(Дата обращения 23.02.2022)

Ссылка: <https://www.php.net/manual/en/book.mysqli.php>

4. C# Documentation URL: Документация по языку C#

Автор: Microsoft

(Дата обращения 13.01.2022)

Ссылка: <https://docs.microsoft.com/ru-ru/dotnet/csharp/>

**Приложение 1**

**Программный код aiscript.cs**

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

public class aiscrip : MonoBehaviour

{

public float speed = 2f;

private Transform target;

void Start()

{

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

}

void Update()

{

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

}

}

**Приложение 2**

**Программный код Buffs.cs**

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

public class Buffs : MonoBehaviour

{

buffSpeed \_buffSpeed;

movement \_movement;

Healthbar hp;

public GameObject buffPick;

GameObject ShieldBuff;

private float buffid = 0;

public static float buffDur;

private float speed = 0;

private string buffName;

public GameObject shield;

private void OnCollisionEnter2D(Collision2D collision)

{

if (collision.gameObject.tag == "Buff")

{

\_movement = gameObject.GetComponent<movement>();

hp = gameObject.GetComponent<Healthbar>();

\_buffSpeed = collision.gameObject.GetComponent<buffSpeed>();

buffDur = \_buffSpeed.buffDuration;

buffid = \_buffSpeed.buffID;

speed = \_movement.movementSpeed;

OnBuff();

Destroy(collision.gameObject);

}

}

private void OnGUI()

{

GUI.Label(new Rect(500, 15, 75, 75), "Buff time Left is " + buffDur.ToString());

GUI.Label(new Rect(700, 15, 75, 75), "Buff type is " + buffName);

}

private void Update()

{

if(buffDur > 0)

{

buffDur -= Time.deltaTime;

Change(buffDur);

}

}

void OnBuff()

{

if (buffid == 0)

{

\_movement.movementSpeed += 2;

buffName = "Speed";

}

else if (buffid == 1)

{

hp.hp += 2;

buffName = "HEALTH";

}

else if (buffid == 2)

{

ShieldBuff = Instantiate(shield, gameObject.transform.position,Quaternion.identity);

buffName = "SUPER SHIELD";

}

}

void Change(float time)

{

if(time <= 0)

{

\_movement.movementSpeed = 4;

Destroy(ShieldBuff);

buffDur = 0;

}

}

}

**Приложение 3**

**Программный код BuffSpawn.cs**

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

public class BuffSpawn : MonoBehaviour

{

public float respawnCd = 5f;

public GameObject prefab;

// Start is called before the first frame update

void Start()

{

InvokeRepeating("buffSpawn", respawnCd, respawnCd);

}

// Update is called once per frame

void Update()

{

}

void buffSpawn()

{

int posX = Random.Range(-20, 20);

int posY = Random.Range(-20, 20);

Vector3 spawnPos = new Vector3(transform.position.x + posX, transform.position.y + posY, 0);

Instantiate(prefab, spawnPos, Quaternion.identity);

}

}

**Приложение 4**

**Программный код buffSpeed.cs**

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

public class buffSpeed : MonoBehaviour

{

public float buffDuration = 10f;

public int buffID = 0;

public int buffChangingTime = 1;

private SpriteRenderer spriteRenderer;

public Sprite[] newSprite;

// Start is called before the first frame update

private void Awake()

{

}

void Start()

{

spriteRenderer = gameObject.GetComponent<SpriteRenderer>();

InvokeRepeating("ChangeSprite",buffChangingTime, buffChangingTime);

}

// Update is called once per frame

void Update()

{

}

void ChangeSprite()

{

buffID = Random.Range(0, 3);

spriteRenderer.size += new Vector2(5, 5);

spriteRenderer.sprite = newSprite[buffID];

}

}

**Приложение 5**

**Программный код bullet.cs**

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

public class bullet : MonoBehaviour

{

public GameObject prefab;

private void OnCollisionEnter2D(Collision2D collision)

{

if (collision.transform.gameObject.tag == "Enemy")

{

// GameObject effect = Instantiate(hitEffect, collision.transform.position, collision.transform.rotation);

Destroy(collision.gameObject);

//Destroy(effect, 2f);

Score.score += 10;

Score.enemyCount--;

Instantiate(prefab, collision.transform.position,Quaternion.identity);

}

Destroy(gameObject);

}

}

**Приложение 6**

**Программный код CameraFollow.cs**

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

public class CameraFollow : MonoBehaviour

{

public Transform player;

private Vector3 offset;

// Update is called once per frame

void Update()

{

transform.position = new Vector3(player.position.x + offset.x, player.position.y + offset.y,-40);

}

}

**Приложение 7**

**Программный код dealingDamage.cs**

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

public class dealingDamage : MonoBehaviour

{

public GameObject prefab;

Healthbar hp;

private void OnCollisionEnter2D(Collision2D collision)

{

if(collision.gameObject.tag == "Player")

{

hp = GameObject.FindGameObjectWithTag("Player").GetComponent<Healthbar>();

hp.hp--;

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

Destroy(gameObject);

Score.enemyCount--;

}

}

}

**Приложение 8**

**Программный код ShieldBuff.cs**

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

public class ShieldBuff : MonoBehaviour

{

public GameObject prefab;

private void Update()

{

if (Buffs.buffDur == 0)

{

Destroy(gameObject);

}

}

private void OnCollisionEnter2D(Collision2D collision)

{

if (collision.transform.gameObject.tag == "Enemy")

{

// GameObject effect = Instantiate(hitEffect, collision.transform.position, collision.transform.rotation);

Destroy(collision.gameObject);

//Destroy(effect, 2f);

Score.score += 10;

Score.enemyCount--;

GameObject corpse = Instantiate(prefab, collision.transform.position, Quaternion.identity);

Destroy(corpse, 10f);

}

}

}

**Приложение 9**

**Программный код shooting.cs**

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

public class shooting : MonoBehaviour

{

public Transform firePoint;

public GameObject prefab;

public float bulletForce = 20f;

// Update is called once per frame

void Update()

{

if (Input.GetButtonDown("Fire1"))

{

Shoot();

}

}

void Shoot()

{

GameObject bullet = Instantiate(prefab, firePoint.position, firePoint.rotation);

Rigidbody2D rb = bullet.GetComponent<Rigidbody2D>();

rb.AddForce(firePoint.up \* bulletForce,ForceMode2D.Impulse);

}

}

**Приложение 10**

**Программный код movement.cs**

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

public class movement : MonoBehaviour

{

public Rigidbody2D rb;

public Camera cam;

public Animator animator;

public int movementSpeed = 4;

Vector2 mvmnt;

public float animspeed;

public bool facingRight = true;

private SpriteRenderer sprite;

private void Start()

{

sprite = gameObject.GetComponent<SpriteRenderer>();

}

void Update()

{

mvmnt.x = Input.GetAxisRaw("Horizontal");

mvmnt.y = Input.GetAxisRaw("Vertical");

animspeed = Mathf.Abs(mvmnt.x) + Mathf.Abs(mvmnt.y);

animator.SetFloat("speed", Mathf.Abs(animspeed\*movementSpeed));

if (mvmnt.x > 0)

{

sprite.flipX = false;

}

else if (mvmnt.x < 0 )

{

sprite.flipX = true;

}

}

private void FixedUpdate()

{

rb.MovePosition(rb.position + mvmnt \* movementSpeed \* Time.fixedDeltaTime);

}

}

**Приложение 11**

**Программный код ShieldConnect.cs**

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

public class ShieldConnect : MonoBehaviour

{

private Transform target;

// Start is called before the first frame update

void Start()

{

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

}

// Update is called once per frame

void Update()

{

transform.position = target.position;

}

}

**Приложение 12**

**Программный код Web.cs**

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

using UnityEngine.UI;

using UnityEngine.Networking;

using UnityEngine.SceneManagement;

using System;

public class Web : MonoBehaviour

{

public static string login, password;

private string warning;

public static int DBscore;

public void PasswordText(string pass)

{

password = pass;

}

public void LoginText(string log)

{

login = log;

}

public IEnumerator Test()

{ //Создаем Запрос

WWWForm form = new WWWForm();

form.AddField("Login", login);

form.AddField("Password", password);

//Отправляем запрос

WWW req = new WWW("siteTest.loc", form);

//Поулчаем Ответ

yield return req;

Debug.Log(req.text);

string[] mas = req.text.Split(' ');

//Score DB

try

{

DBscore = int.Parse(mas[1]);

}

catch(Exception )

{

}

if (mas[0] == "True")

{

SceneManager.LoadScene(1);

}

else

{

warning = "Неправильный логин или пароль!";

}

}

public void ButtonPress()

{

StartCoroutine(Test());

}

private void OnGUI()

{

GUI.Label(new Rect(100, 15, 200, 200), warning);

}

public void OpenURL()

{

Application.OpenURL("http://site.loc/login.php");

}

}

**Приложение 13**

**Программный код Death.cs**

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

public class Death : MonoBehaviour

{

public GameObject particle;

void Start()

{

GameObject particleInstance = Instantiate(particle, transform.position, Quaternion.identity);

Destroy(particleInstance, 1f);

Destroy(gameObject, 10f);

}

}

**Приложение 14**

**Программный код rotation.cs**

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

public class rotation : MonoBehaviour

{

//Weapon Rotation Script

public float speed = 50f;

public bool facingRight = false;

public Transform player;

// Update is called once per frame

void Update()

{

Vector2 dir = Camera.main.ScreenToWorldPoint(Input.mousePosition) - transform.position;

float angle = Mathf.Atan2(dir.y, dir.x) \* Mathf.Rad2Deg;

Quaternion rotation = Quaternion.AngleAxis(angle, Vector3.forward);

transform.rotation = Quaternion.Slerp(transform.rotation, rotation, speed \* Time.deltaTime);

transform.position = player.position;

}

private void FixedUpdate()

{

}

}

**Приложение 15**

**Программный код Healthbar.cs**

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

using UnityEngine.SceneManagement;

public class Healthbar : MonoBehaviour

{

public int hp = 5;

private void Update()

{

if (hp <= 0)

{

SceneManager.LoadScene(3);

}

}

private void OnGUI()

{

GUI.Label(new Rect(10, 15, 75, 75), "HP is " + hp.ToString());

}

}

**Приложение 16**

**Программный код Score.cs**

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

public class Score : MonoBehaviour

{

public static int score = 0;

public static int enemyCount = 0;

public float spawnTime = 1000f;

public GameObject prefab;

public int hp;

GameObject[] posSpawn = new GameObject[6];

void Start()

{

InvokeRepeating("SpawnEnemy", spawnTime, spawnTime);

posSpawn = GameObject.FindGameObjectsWithTag("Point");

}

private void OnGUI()

{

GUI.Label(new Rect(100, 15, 75, 75), "Score is " + score.ToString());

}

void SpawnEnemy()

{

if (MaxEnemies.enemyMax > enemyCount)

{

Instantiate(prefab, posSpawn[Random.Range(0, 6)].GetComponent<Transform>());

enemyCount++;

}

}

}

**Приложение 17**

**Программный код detatching.cs**

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

public class detatching : MonoBehaviour

{

// Update is called once per frame

void Update()

{

transform.DetachChildren();

}

}

**Приложение 18**

**Программный код Link.cs**

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

public class Link : MonoBehaviour

{

public void UrlOpen()

{

Application.OpenURL("http://site.loc/login.php");

}

}

**Приложение 19**

**Программный код MainMenu.cs**

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

using UnityEngine.SceneManagement;

public class MainMenu : MonoBehaviour

{

// Start is called before the first frame update

public void Play(int index)

{

SceneManager.LoadScene(index);

}

}

**Приложение 20**

**Программный код following.cs**

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

public class following : MonoBehaviour

{

public Transform player;

private Vector3 offset;

// Update is called once per frame

void Update()

{

transform.position = new Vector3(player.position.x + offset.x, player.position.y + offset.y, -40);

}

}

**Приложение 21**

**Программный код ParticleScript.cs**

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

public class ParticleScript : MonoBehaviour

{

void Update()

{

Destroy(gameObject, 3f);

}

}

**Приложение 22**

**Программный код ScoreGameOver.cs**

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

using UnityEngine.SceneManagement;

using UnityEngine.UI;

public class ScoreGameOver : MonoBehaviour

{

Text endScore;

void Start()

{

endScore = GetComponent<Text>();

endScore.text +=" "+ Score.score.ToString(); ;

StartCoroutine(ScoreSend());

}

public IEnumerator ScoreSend()

{

if(Web.DBscore >= Score.score)

{

}

else

{

WWWForm form = new WWWForm();

form.AddField("Score", Score.score);

form.AddField("Login", Web.login);

WWW req = new WWW("siteTest.loc", form);

yield return req;

}

Score.score = 0;

}

}

**Приложение 23**

**Программный код Textt.cs**

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

using UnityEngine.UI;

public class Textt : MonoBehaviour

{

Text text;

// Start is called before the first frame update

void Start()

{

text = GetComponent<Text>();

text.text += ", "+Web.login;

}

}

**Приложение 24**

**Программный код MaxEnemiews.cs**

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

using UnityEngine.UI;

using UnityEngine.SceneManagement;

public class MaxEnemies : MonoBehaviour

{

public static int enemyMax = 100;

public Text text;

public Text inputText;

private void Start()

{

inputText.text = enemyMax.ToString();

}

private void Update()

{

text.text = "Максимальное количество противников = "+ enemyMax.ToString();

}

public void EnemyChange(string enemies)

{

try

{

enemyMax = System.Int32.Parse(enemies);

}

catch (System.Exception)

{

}

}

public void ButtonPressed(int x)

{

SceneManager.LoadScene(x);

}

}