Федеральное государственное автономное образовательное

учреждение высшего образования

«КРЫМСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ

имени В.И. Вернадского»

(ФГАОУ ВО «КФУ им. В.И. Вернадского»)

Физико-технический институт

(наименование структурного подразделения)

**ДНЕВНИК**

по учебной эксплуатационной практике

Направление подготовки 09.03.04 "Программная инженерия"

Курс — 2 Группа ПИ-б-о-202(1)

Вид практики —Учебная практика, эксплуатационная

Тип практики — распределенная

Сроки прохождения практики с «03» мая 2022г. по «25» июня 2022г.

Содержание практики: разработка пиксельной 2D игры раннера с элементами РПГ на игровом движке Unity под Android платформы

(указываются основные виды и задачи профессиональной деятельности, в выполнении которых обучающийся приобретает опыт)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **№** | **Индивидуальные задания, раскрывающие содержание практики** | **Планируемые результаты практики** | **Совместный рабочий график (план) проведения практики** | **Отметка о выполнении** |
| 1. | (Индивидуальные задания должны формировать все показатели всех закрепленных за практикой компетенций) | Указываются показатели компетенций | Указываются сроки выполнения заданий | Проставляется отметка «выполнено» в случае успешного выполнения задания |
| 2. | Разработка плана игры, создание макета | План и структура проекта | 4.05.2022 – 16.05.2022 |  |
| 3. | Создание спрайтов игровых объектов | Нарисованные пиксельные спрайты | 16.05.2022 – 23.05.2022 |  |
| 4. | Создание анимации персонажей | Нарисованные анимации персонажей | 23.05.2022 – 27.05.2022 |  |
| 5. | Реализация игровых механик | Готовые механики игры | 27.05.2022 – 18.06.2022 |  |
| 6. | Создание пользовательского интерфейса | Графический пользовательский интерфейс | 18.06.2022 – 25.06.2022 |  |
| 7. |  |  |  |  |

Прохождение инструктажа по ознакомлению с требованиями охраны труда, техники безопасности, пожарной безопасности, а также правилами внутреннего трудового распорядка подтверждаю

Обучающийся \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(подпись) (ФИО обучающегося)

Руководитель практики от КФУ\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(подпись) (ФИО руководителя)

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО

ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

«Крымский федеральный университет имени В.И. Вернадского»

Физико-технический институт

ОТЧЁТ

по учебной эксплуатационной практике

обучающегося Писляков Олег Игоревич

Факультет\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Кафедра компьютерной инженерии и моделирования

Направление подготовки/специальность 09.03.04 Программная инженерия

Направленность(профиль)/специализация\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Курс 2 Группа ПИ-б-о-202(1) Форма обучения очная

Руководитель практики от Университета \_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_

(подпись) (должность, ФИО)

Руководитель практики от профильной организации \_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_

(подпись) (должность, ФИО)

Отчёт защищён с оценкой \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

# **ОГЛАВЛЕНИЕ**

[ОГЛАВЛЕНИЕ 5](#_Toc107373134)

[ВВЕДЕНИЕ 6](#_Toc107373135)

[ТЕХНИЧЕСКОЕ ЗАДАНИЕ 7](#_Toc107373136)

[ХОД РАБОТЫ 8](#_Toc107373137)

[1. Анализ инструментария. 8](#_Toc107373138)

[2. Создание спрайтов игровых объектов. 8](#_Toc107373139)

[3. Создание анимации игровых персонажей. 9](#_Toc107373140)

[4. Реализация игровых механик. 11](#_Toc107373141)

[5. Создание пользовательского интерфейса. 24](#_Toc107373142)

[ВЫВОД 30](#_Toc107373143)

# ВВЕДЕНИЕ

Игровая индустрия постоянно развивается и в неё приходит всё больше и больше новых интересных проектов. Актуальность видеоигр будет понятна всегда, это таким образом, на что люди готовы тратить время досуга. Смешивая разные жанры игр, механики и графику получаются действительно инновационные и необычные работы.

Целью проекта является эксперимент по созданию игры со смесью двух жанров, а именно:

1. Раннер – игра, где персонаж постоянно бежит/летит/ползет/прихрамывает вперед по теоретически бесконечному игровому миру.

2. РПГ - [жанр](https://ru.wikipedia.org/wiki/%D0%9A%D0%BB%D0%B0%D1%81%D1%81%D0%B8%D1%84%D0%B8%D0%BA%D0%B0%D1%86%D0%B8%D1%8F_%D0%BA%D0%BE%D0%BC%D0%BF%D1%8C%D1%8E%D1%82%D0%B5%D1%80%D0%BD%D1%8B%D1%85_%D0%B8%D0%B3%D1%80) [компьютерных игр](https://ru.wikipedia.org/wiki/%D0%9A%D0%BE%D0%BC%D0%BF%D1%8C%D1%8E%D1%82%D0%B5%D1%80%D0%BD%D0%B0%D1%8F_%D0%B8%D0%B3%D1%80%D0%B0), основанный на элементах игрового процесса традиционных [настольных ролевых игр](https://ru.wikipedia.org/wiki/%D0%9D%D0%B0%D1%81%D1%82%D0%BE%D0%BB%D1%8C%D0%BD%D0%B0%D1%8F_%D1%80%D0%BE%D0%BB%D0%B5%D0%B2%D0%B0%D1%8F_%D0%B8%D0%B3%D1%80%D0%B0).

Второй особенностью проекта послужит то, что всё это подано в обертке пиксельной графики, так сильно запомнившейся и выделившейся во времена первых игровых приставок и 8-битных игр на картриджах. Вся работа выполнена одним человеком, благодаря чему игра попадает под категорию Инди. Для разработки был выбран лидирующий игровой движок Unity, в частности разработка 2D игры раннера с элементами РПГ.

# ТЕХНИЧЕСКОЕ ЗАДАНИЕ

Наименование игры – «RunPG».

Краткая характеристика: игра предназначена для людей всех возрастов и категорий. Бесконечность игры, постепенное нарастание уровня сложности, а также повышение уровня и характеристик персонажа держат пользователя заинтересованным.

Игра представляет из себя цикличный уровень, по которому бежит персонаж. Ему на встречу случайным образом попадают враги, у которых в зависимости от времени увеличивается сложность. Также есть уменьшенная вероятность появления сундука, при приближении к которому, из него выпадает случайное количество зелий.

При уничтожении врага игрок получает опыт, который накапливаясь до определённого значения повышает уровень. За это игрок получает очки улучшений, за счёт которых он может увеличить свои характеристики.

У персонажа есть два вида атак: ближняя и дальняя. При ближней атаке игрок бьёт в небольшом радиусе перед собой, нанося урон противникам, попавшим в область поражения. Дальняя атака запускает магический огненный шар, который имеет увеличенную перезарядку, а также стоимость в мане.

Для восстановления запасов здоровья и маны игроку предоставляются зелья, каждого вида которых он может иметь до 3 штук.

В случае поражения игрока открывается окно меню, которое предлагает попробовать себя ещё раз или закрыть игру.

Требования к устройству: Android версии 7 и выше.

# ХОД РАБОТЫ

## Анализ инструментария.

Среда разработки: игровой движок Unity - межплатформенная среда разработки компьютерных игр, разработанная американской компанией Unity Technologies. Unity позволяет создавать приложения, работающие на более чем 25 различных платформах, включающих персональные компьютеры, игровые консоли, мобильные устройства, интернет-приложения и другие.

Редактор программного кода: Visual Studio 2019 - стартовая площадка для написания, отладки и сборки кода, а также последующей публикации приложений.

Графический редактор: Aseprite - редактор изображений с доступным исходным кодом, предназначенный в первую очередь для рисования и анимации в пиксельной графике.

Тестовая платформа: Unity Remote 5 - универсальное решение, которое, за счёт одновременно открытого на ноутбуке движка и копии проекта на телефоне, позволяет работать непрерывно и эффективно, а также изменять контент в реальном времени.

## Создание спрайтов игровых объектов.

Для создания спрайтов объектов использовалась специализированная программа Aseprite, имеющая все полезные функции для рисования в стиле пиксель-арт и не имеющая лишних. Вся графика была нарисована в этом редакторе. Работа с программой показана на примере создания образа главного персонажа на рисунке ниже (Рис. 1):

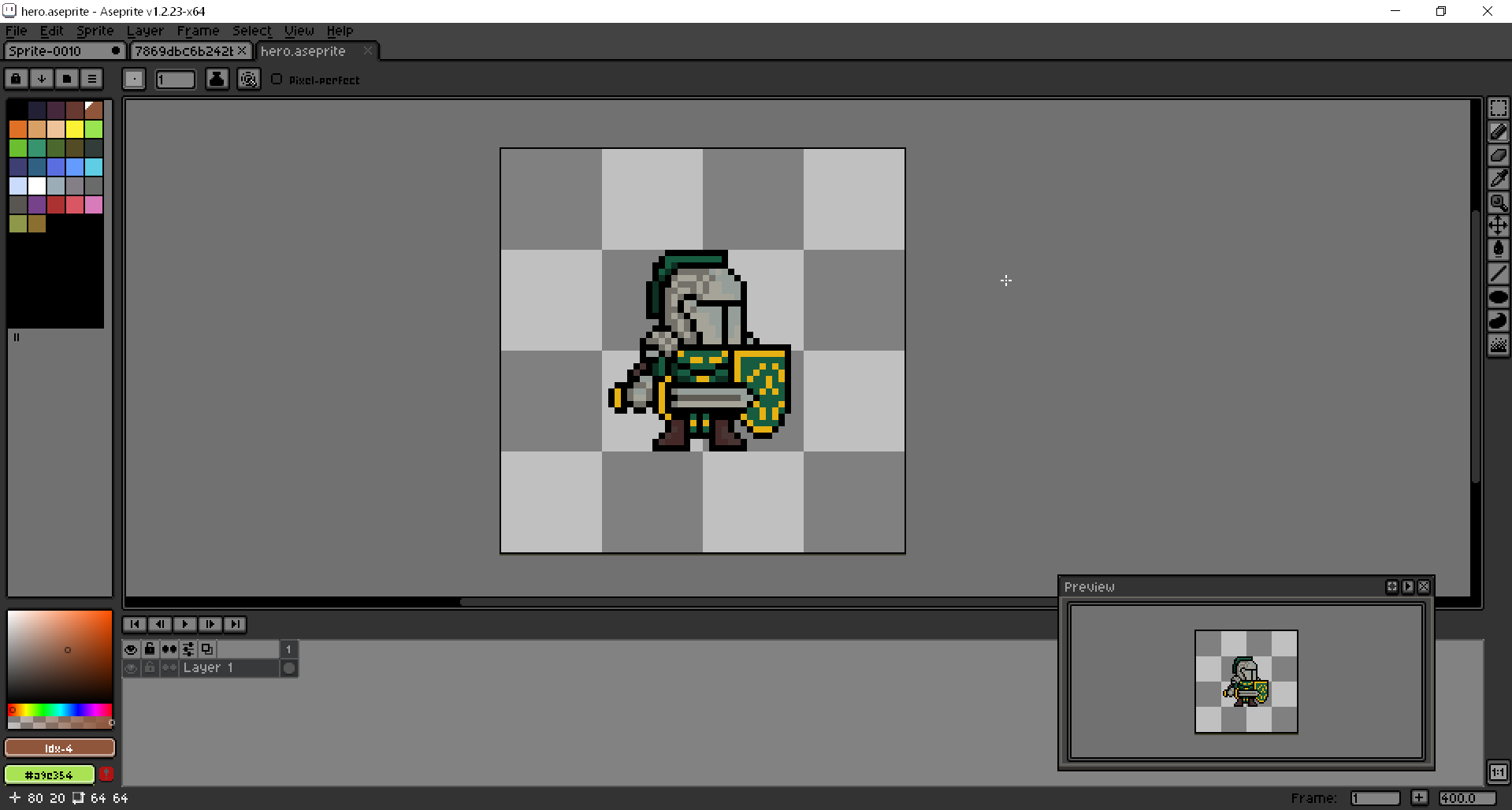


Рис. 1. Создание спрайта главного персонажа.

## Создание анимации игровых персонажей.

Для создания анимации использовалась та же программа, что и для рисования спрайтов. Aseprite имеет функционал создания фреймов, на каждом из которых рисуется кадр анимации, после чего задаётся скорость перехода между ними. Работа по созданию анимации показана на рисунке ниже (Рис. 2):

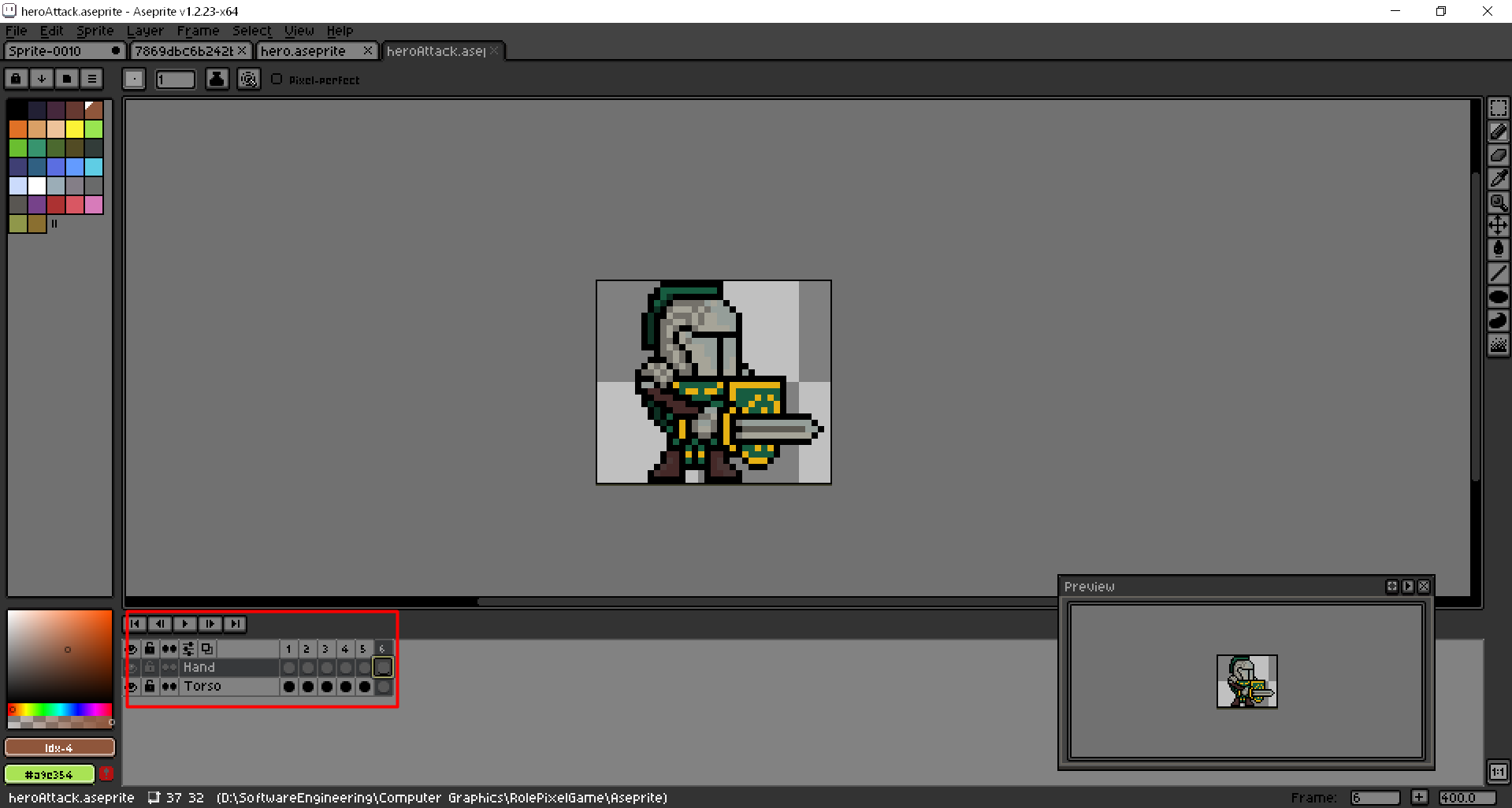


Рис. 2. Создание анимации главного персонажа.

После завершения покадровой отрисовки анимации файл сохраняется в формате Spritesheet (Рис. 3). Другими словами, каждый кадр дублируется в ряд на одном изображении.



Рис. 3. Демонстрация формата Spritesheet.

После чего в Unity нарезается данное изображение на равные отрезки, выбирается скорость изменения кадров анимации и подключается к персонажу с необходимыми условиями запуска анимации (Рис. 4).

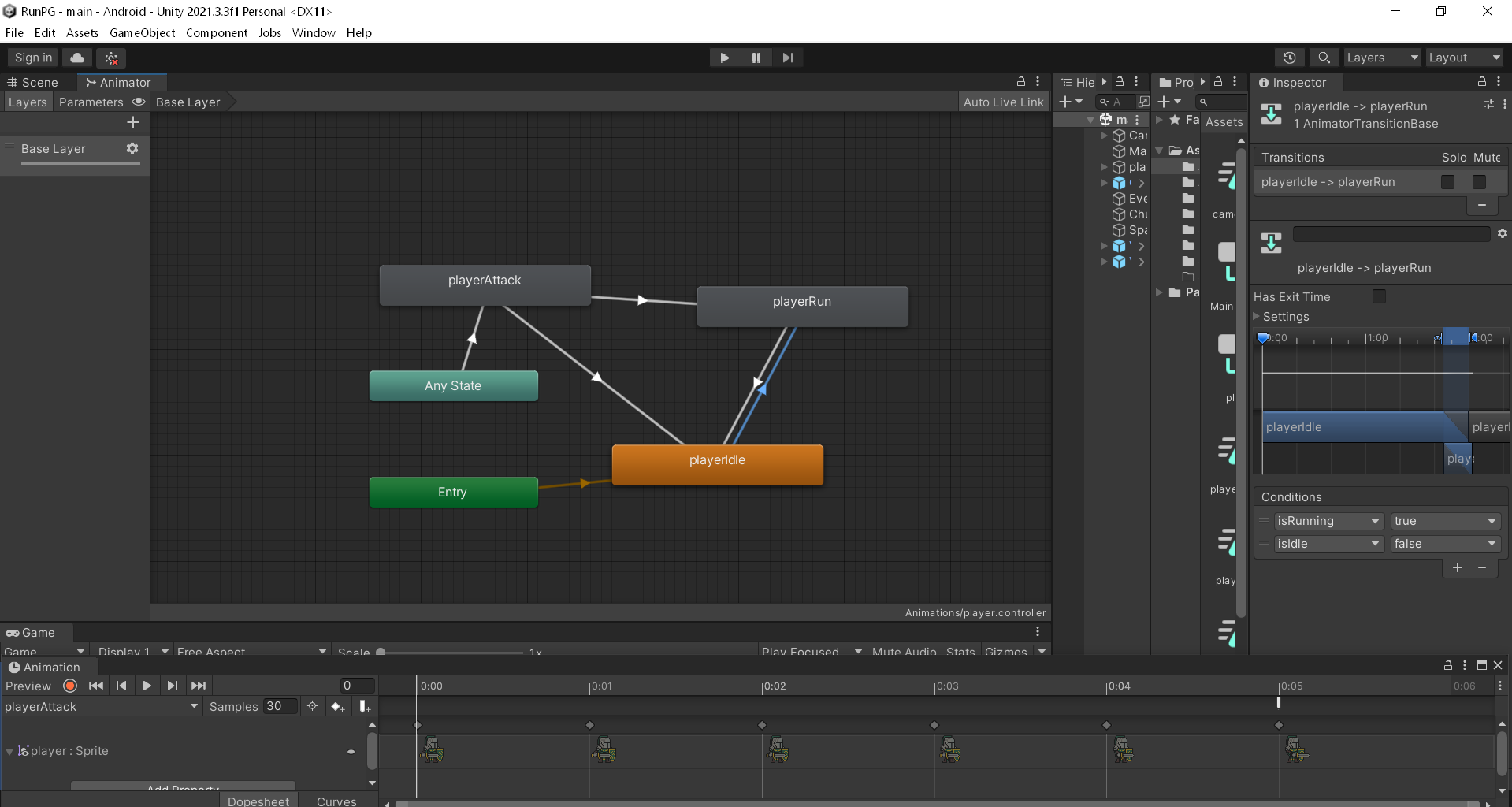


Рис. 4. Анимации в Unity.

В переходах анимации задаются условия, которые обрабатываются и вызываются непосредственно из скрипта (Рис. 5).

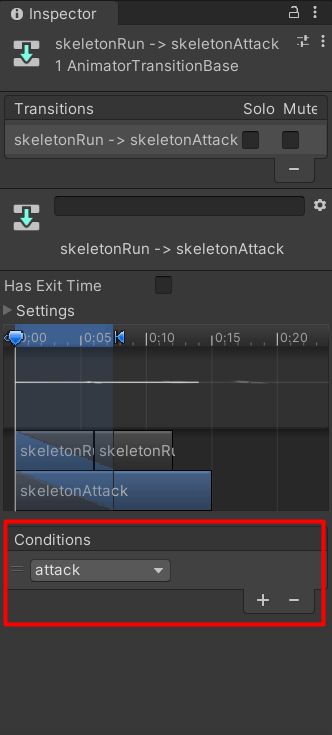




Рис. 5. Условия перехода анимации.

## Реализация игровых механик.

На экране сразу можно видеть основные объекты игры. В первую очередь – объект Chunk. Он представляет из себя совокупность фона и поверхности, по которой перемещается игрок, а также точек начала и конца чанка, необходимые для объединения старого и нового чанка, для бесконечной их генерации. Листинг и рисунок (Рис. 6) прилагаются ниже:

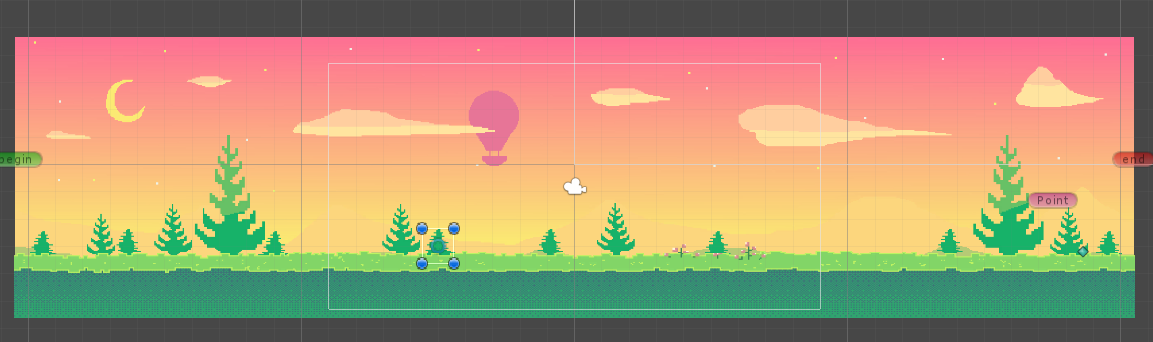


Рис. 6. Объект Chunk.

sing System.Collections;

using System.Collections.Generic;

using UnityEngine;

public class TerraPlaser : MonoBehaviour

{

public Transform player;

public Chunk[] chunkPrefabs;

public Chunk firstChunk;

private List<Chunk> spawnedChunks = new List<Chunk>();

void Start()

{

spawnedChunks.Add(firstChunk);

}

void Update()

{

if(player.position.x > spawnedChunks[spawnedChunks.Count - 1].end.position.x - 20)

{

SpawnChunk();

}

}

private void SpawnChunk()

{

Chunk newChunk = Instantiate(chunkPrefabs[Random.Range(0, chunkPrefabs.Length)]);

newChunk.transform.position = spawnedChunks[spawnedChunks.Count - 1].end.position - newChunk.begin.localPosition;

spawnedChunks.Add(newChunk);

if(spawnedChunks.Count >= 3)

{

Destroy(spawnedChunks[0].gameObject);

spawnedChunks.RemoveAt(0);

}

}

}

Также на сцене присутствует персонаж, которым управляет игрок. Помимо компонентов Unity, к нему прикреплены следующие скрипты:

Player.cs – содержит основную логику управления персонажем.

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

using UnityEngine.UI;

public class Player : MonoBehaviour

{

public float speed;

public float dir;

public int currentExp;

public int maxExp;

public int level;

private PlayerStats stats;

private Rigidbody2D rb;

private Animator anim;

private Animator camAnim;

private StatsPanel sp;

public int levelPointsInt;

public Image levelUp;

public DeathMenu deathMenu;

private void Start()

{

sp = FindObjectOfType<StatsPanel>();

stats = FindObjectOfType<PlayerStats>();

rb = GetComponent<Rigidbody2D>();

anim = GetComponent<Animator>();

camAnim = GameObject.FindGameObjectWithTag("MainCamera").GetComponent<Animator>();

}

private void FixedUpdate()

{

rb.velocity = new Vector2(dir \* speed, rb.velocity.y);

if (currentExp >= maxExp)

{

currentExp -= maxExp;

maxExp \*= 2;

level++;

stats.health = stats.maxHealth;

stats.mana = stats.maxMana;

levelPointsInt++;

}

GameObject.FindGameObjectWithTag("LevelOrb").GetComponentInChildren<Text>().text = level.ToString();

if (levelPointsInt > 0) levelUp.gameObject.SetActive(true);

else levelUp.gameObject.SetActive(false);

if (stats.health <= 0) Death();

}

public void Death()

{

deathMenu.gameObject.SetActive(true);

Destroy(gameObject);

Time.timeScale = 0f;

}

public void TakeDamage(int damage)

{

camAnim.SetTrigger("shake");

if (damage > stats.defense)

{

damage -= stats.defense;

stats.health -= damage;

}

}

public void HealthUp()

{

if (levelPointsInt > 0)

{

stats.maxHealth += 5;

levelPointsInt--;

}

}

public void ManaUp()

{

if (levelPointsInt > 0)

{

stats.maxMana += 5;

levelPointsInt--;

}

}

public void DefenseUp()

{

if (levelPointsInt > 0)

{

stats.defense += 1;

levelPointsInt--;

}

}

public void DamageUp()

{

if (levelPointsInt > 0)

{

stats.damage += 1;

levelPointsInt--;

}

}

public void CritChanceUp()

{

if (levelPointsInt > 0)

{

stats.critChance += 1;

levelPointsInt--;

}

}

public void CritRateUp()

{

if (levelPointsInt > 0)

{

stats.critRate += 5;

levelPointsInt--;

}

}

}

PlayerStats.cs – содержит характеристики персонажа.

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

public class PlayerStats : MonoBehaviour

{

public int defense;

public int maxHealth;

public int health;

public int mana;

public int maxMana;

public int damage;

public int critChance;

public int critRate;

void Start()

{

health = maxHealth;

mana = maxMana;

}

}

Fireball.cs – содержит логику использования магического снаряда.

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

public class Fireball : MonoBehaviour

{

public float startTimeBtwShots;

private float timeBtwShots;

public int cost;

private PlayerStats stats;

public GameObject fireball;

public Buttons fireballButton;

public Transform shotPoint;

private void Start()

{

stats = FindObjectOfType<PlayerStats>();

}

private void FixedUpdate()

{

if (timeBtwShots <= 0 && stats.mana >= cost)

{

fireballButton.img.sprite = fireballButton.sprite;

if (fireballButton.isAttackClicked == true)

{

Instantiate(fireball, shotPoint.position, transform.rotation);

stats.mana -= cost;

timeBtwShots = startTimeBtwShots;

}

} else

{

fireballButton.img.sprite = fireballButton.spriteDisactive;

timeBtwShots -= Time.fixedDeltaTime;

}

}

}

К предыдущему скрипту привязан ещё один, описывающий логику поведения снаряда (Рис. 7), а также частиц, испускаемых им при попадании. FireballObject.cs:



Рис. 7. Магический снаряд.

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

public class FireballObject : MonoBehaviour

{

public float speed;

public float lifeTime;

public float distance;

public int damage;

public GameObject effect;

public LayerMask whatIsSloid;

private Animator camAnim;

private void Start()

{

camAnim = GameObject.FindGameObjectWithTag("MainCamera").GetComponent<Animator>();

}

private void Update()

{

lifeTime -= Time.deltaTime;

RaycastHit2D hitInfo = Physics2D.Raycast(transform.position, transform.right, distance, whatIsSloid);

if (hitInfo.collider != null)

{

if (hitInfo.collider.CompareTag("Enemy"))

{

hitInfo.collider.GetComponent<Enemy>().TakeDamage(damage);

}

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

camAnim.SetTrigger("shake");

Destroy(gameObject);

}

transform.Translate(Vector2.right \* speed \* Time.deltaTime);

if (lifeTime <= 0) Destroy(gameObject);

}

}

PlayerAttack.cs – содержит логику ближней атаки персонажа.

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

public class PlayerAttack : MonoBehaviour

{

private float timeBtwAttack;

public float startTimeBtwAttack;

private float speed;

private Player player;

private PlayerStats stats;

public Transform attackPos;

public LayerMask enemy;

public float attackRange;

public Animator anim;

public Buttons attackButton;

private void Start()

{

stats = FindObjectOfType<PlayerStats>();

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

speed = player.speed;

}

private void Update()

{

if (timeBtwAttack <= 0)

{

attackButton.img.sprite = attackButton.sprite;

if (attackButton.isAttackClicked == true)

{

player.speed = 0;

anim.SetBool("isRunning", false);

anim.SetTrigger("attack");

timeBtwAttack = startTimeBtwAttack;

}

else

{

player.speed = speed;

anim.SetBool("isRunning", true);

}

}

else

{

attackButton.img.sprite = attackButton.spriteDisactive;

timeBtwAttack -= Time.deltaTime;

}

}

public void OnAttack()

{

Collider2D[] enemies = Physics2D.OverlapCircleAll(attackPos.position, attackRange, enemy);

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

{

if(Random.Range(1, 100) > (100 - stats.critChance))

{

enemies[i].GetComponent<Enemy>().TakeDamage(stats.damage + stats.critRate);

}

else

{

enemies[i].GetComponent<Enemy>().TakeDamage(stats.damage);

}

}

}

public void OnDrawGizmosSelected()

{

Gizmos.color = Color.red;

Gizmos.DrawWireSphere(attackPos.position, attackRange);

}

}

PoisonsUse.cs – содержит логику использования зелий (Рис. 8).



Рис. 8. Спрайт зелья здоровья.

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

using UnityEngine.UI;

public class PoisonsUse : MonoBehaviour

{

public float startTimeBtwUse;

private float timeBtwUse;

public int countSP;

public int maxCount;

public int countHP;

private PlayerStats stats;

public Buttons spButton;

public Buttons hpButton;

private void Start()

{

countSP = int.Parse(spButton.GetComponentInChildren<Text>().text);

countHP = int.Parse(hpButton.GetComponentInChildren<Text>().text);

stats = FindObjectOfType<PlayerStats>();

}

private void FixedUpdate()

{

if (timeBtwUse <= 0 && stats.mana < stats.maxMana && countSP > 0)

{

spButton.img.sprite = spButton.sprite;

if (spButton.isAttackClicked == true)

{

stats.mana = stats.maxMana;

countSP--;

spButton.GetComponentInChildren<Text>().text = countSP.ToString();

timeBtwUse = startTimeBtwUse;

}

}

else

{

spButton.img.sprite = spButton.spriteDisactive;

timeBtwUse -= Time.fixedDeltaTime;

}

if (timeBtwUse <= 0 && stats.health < stats.maxHealth && countHP > 0)

{

hpButton.img.sprite = hpButton.sprite;

if (hpButton.isAttackClicked == true)

{

stats.health = stats.maxHealth;

countHP--;

hpButton.GetComponentInChildren<Text>().text = countHP.ToString();

timeBtwUse = startTimeBtwUse;

}

}

else

{

hpButton.img.sprite = hpButton.spriteDisactive;

timeBtwUse -= Time.fixedDeltaTime;

}

}

private void OnTriggerEnter2D(Collider2D collision)

{

if (collision.CompareTag("HPpoison"))

{

if (countHP == maxCount)

{

Destroy(collision.gameObject);

}

else

{

countHP++;

hpButton.GetComponentInChildren<Text>().text = countHP.ToString();

Destroy(collision.gameObject);

}

}

else if(collision.CompareTag("SPpoison"))

{

if (countSP == maxCount)

{

Destroy(collision.gameObject);

}

else

{

countSP++;

spButton.GetComponentInChildren<Text>().text = countSP.ToString();

Destroy(collision.gameObject);

}

}

}

}

Также имеется скрипт привязанный к камере. Он содержит логику слежения камеры за персонажем и привязку к ней некоторых вспомогательных объектов. CameraFollow.cs:

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

public class CameraFollow : MonoBehaviour

{

public Transform playerPos;

public Transform variantPos;

public Transform variantGoodPos;

public Transform spawnerPos;

private Vector3 variantOffset;

private Vector3 variantGoodOffset;

private float startPlayerPos;

private void Start()

{

variantOffset = variantPos.position - transform.position;

variantGoodOffset = variantGoodPos.position - transform.position;

startPlayerPos = playerPos.position.x;

}

private void FixedUpdate()

{

variantPos.position = new Vector3(transform.position.x + variantOffset.x, 0f, 0f);

variantGoodPos.position=new Vector3(transform.position.x+variantGoodOffset.x,0f, 0f);

spawnerPos.position = new Vector3(transform.position.x, 0f, 0f);

if (playerPos.position.x > startPlayerPos)

{

transform.position = new Vector3(playerPos.position.x - startPlayerPos, transform.position.y, transform.position.z);

}

}

}

Для реализации случайного появления врагов был создан объект – Spawner. Он содержит логику генерации врагов, их усиления в зависимости от времени, а также генерацию сундуков с пониженным шансом. Spawner.cs:

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

public class Spawner : MonoBehaviour

{

public GameObject[] variants;

public GameObject variantGood;

private float timeBtwSpawn;

public float startTimeBtwSpawn;

public float decreaseTime;

public float minTime;

public float chestSpawnChance;

public float enemyHealth;

public float increaseHealth;

public float enemyDamage;

public float increaseDamage;

public float exp;

public float increaseExp;

private void Update()

{

enemyDamage += Time.deltaTime \* increaseDamage;

enemyHealth += Time.deltaTime \* increaseHealth;

exp += Time.deltaTime \* increaseExp;

if (timeBtwSpawn <= 0)

{

int rand = Random.Range(0, variants.Length);

if (Random.Range(0, 100) >= (100 - chestSpawnChance))

{

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

}

else

{

Instantiate(variants[rand], transform.position, Quaternion.identity);

}

timeBtwSpawn = startTimeBtwSpawn;

if (startTimeBtwSpawn > minTime)

{

startTimeBtwSpawn -= decreaseTime;

}

}

else

{

timeBtwSpawn -= Time.deltaTime;

}

}

}

Спавнер генерирует варианты появления врагов и сундуков в конкретных точках, постоянно находящихся за пределами видимости игрока. Для этих точек были созданы отельные скрипты:

Point.cs – содержит логику появления врагов.

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

public class Point : MonoBehaviour

{

public Enemy[] enemies;

private Spawner spawner;

private void Start()

{

spawner = FindObjectOfType<Spawner>();

Instantiate(enemies[Random.Range(0, enemies.Length - 1)], transform.position, transform.rotation);

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

{

enemies[i].health = Mathf.RoundToInt(spawner.enemyHealth);

enemies[i].damage = Mathf.RoundToInt(spawner.enemyDamage);

enemies[i].exp = Mathf.RoundToInt(spawner.exp);

}

}

}

PointGood.cs – содержит логику появления сундуков.

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

public class PointGood : MonoBehaviour

{

public GameObject[] chest;

private Spawner spawner;

private void Start()

{

spawner = FindObjectOfType<Spawner>();

Instantiate(chest[Random.Range(0, chest.Length)], transform.position, transform.rotation);

}

}

Враги (Рис. 9) имеют свой скрипт, в которых содержится логика поведения, а также стартовые характеристики, которые не определяются скриптом спавнера. Enemy.cs:



Рис. 9. Спрайт врага.

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

public class Enemy : MonoBehaviour

{

public int health;

public int damage;

public int exp;

public float speed;

public float forceImpulse;

public float startTimeBtwAttack;

private float timeBtwAttack;

public float startStopTime;

private float stopTime;

public float normalSpeed;

private Player player;

private PlayerStats stats;

private Rigidbody2D rb;

public GameObject deathEffect;

private Animator anim;

private void Start()

{

stats = FindObjectOfType<PlayerStats>();

anim = GetComponent<Animator>();

rb = GetComponent<Rigidbody2D>();

player = FindObjectOfType<Player>();

normalSpeed = speed;

}

private void Update()

{

if (stopTime <= 0)

{

speed = normalSpeed;

}

else

{

speed = 0;

stopTime -= Time.deltaTime;

}

if(health <= 0)

{

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

Destroy(gameObject);

player.currentExp += exp;

}

transform.Translate(Vector2.left \* speed \* Time.deltaTime);

}

public void TakeDamage(int damage)

{

stopTime = startStopTime;

health -= damage;

}

public void OnTriggerStay2D(Collider2D collision)

{

if (collision.CompareTag("Player"))

{

if (timeBtwAttack <= 0)

{

anim.SetTrigger("attack");

timeBtwAttack = startTimeBtwAttack;

}

else

{

timeBtwAttack -= Time.deltaTime;

}

}

}

public void OnEnemyAttack()

{

player.TakeDamage(damage);

rb.AddForce(gameObject.transform.right \* forceImpulse,ForceMode2D.Impulse);

}

}

Для правильной реализации работы сундуков (Рис. 10) им также необходим скрипт. Chest.cs:



Рис. 10. Спрайт открытого сундука.

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

public class Chest : MonoBehaviour

{

public GameObject[] loot;

public GameObject chestOpen;

public float dropImpulse;

private int count;

private void Awake()

{

count = Random.Range(1, 2);

}

private void OnTriggerEnter2D(Collider2D collision)

{

if (collision.CompareTag("Player"))

{

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

Destroy(gameObject);

for (int i = 0; i < count; i++)

{

dropImpulse = Random.Range(0.5f, 2.5f);

GameObject drop = Instantiate(loot[Random.Range(0, loot.Length)], transform.position, Quaternion.identity);

drop.GetComponent<Rigidbody2D>().AddForce(Vector2.right \* dropImpulse, ForceMode2D.Impulse);

}

}

}

}

Помимо перечисленного также имеется скрипт Destroy.cs уничтожающий объект, если игрок отошёл от него на расстояние, с которого объект не видно. Скрипт нужен для того, чтобы не нагромождать в памяти устройства бесконечное количество сундуков, фонов и прочих объектов, которые игрок уже пробежал.

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

public class Destroy : MonoBehaviour

{

private Transform player;

private void Start()

{

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

}

void Update()

{

if (player.position.x > transform.position.x + 15)

{

Destroy(gameObject);

}

}

}

## Создание пользовательского интерфейса.

Пользовательский интерфейс (Рис. 11) данной игры имеет три типа объектов – это кнопки, ползунки и интерактивные окна.



Рис. 11. Пользовательский интерфейс.

Рассмотрим кнопки. У каждой кнопки есть перезарядка, а также запрет на использование в зависимости от условий, например, нельзя использовать дальнюю атаку при нехватке маны или нельзя использовать зелье, если соответствующий слайдер заполнен. Справа в углу есть две кнопки: ближней (Рис. 12) и дальней атаки (Рис. 13). Логика их работы описана в вышеприведённых скриптах, а именно PlayerAttack.cs и Fireball.cs соответственно.

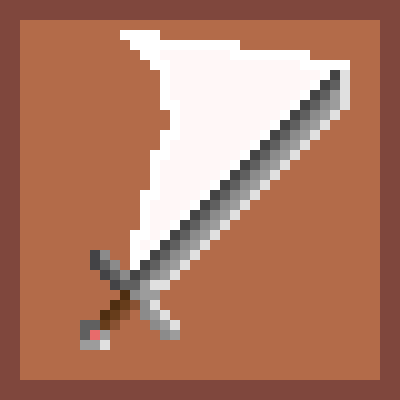


Рис. 12. Кнопка ближней атаки.



Рис. 13. Кнопка дальней атаки.

Слева в углу есть кнопки использования зелий (Рис. 14). На кнопках также отображено количество имеющихся в наличии экземпляров. Их логика также описана в скрипте PoisonsUse.cs.



Рис. 14. Кнопки использования зелий маны и здоровья соответственно.

Далее перейдём к обзору ползунков. Слева сверху видны полоски здоровья и маны (Рис. 15). Их логика описана в скрипте HealthBar.cs привязанном к главному персонажу.



Рис. 15. Ползунки здоровья и маны соответственно.

using System;

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

using UnityEngine.UI;

public class HealthBar : MonoBehaviour

{

private Player player;

private PlayerStats stats;

public Image healthBar;

public Image manaBar;

public Image levelBar;

public float fillHealth;

public float fillMana;

public float fillLevel;

private void Start()

{

player = FindObjectOfType<Player>();

stats = FindObjectOfType<PlayerStats>();

fillHealth = stats.maxHealth;

fillMana = stats.maxMana;

fillLevel = player.currentExp;

}

private void Update()

{

fillHealth = (float)stats.health / (float)stats.maxHealth;

fillMana = (float)stats.mana / (float)stats.maxMana;

fillLevel = (float)player.currentExp / (float)player.maxExp;

healthBar.fillAmount = fillHealth;

manaBar.fillAmount = fillMana;

levelBar.fillAmount = fillLevel;

}

}

Справа от ползунков показателей персонажа видна окружность, работающая на том же скрипте, что и предыдущие ползунки. Она информирует игрока о том, какое количество опыта он имеет до следующего уровня, уровень персонажа, а также то, есть ли у игрока очки для улучшения характеристик. Если очки имеются, в углу сферы появляется жёлтая стрелочка, интуитивно дающая понять, что у игрока есть незаконченная активность (Рис. 16).



Рис. 16. Индикатор уровня.

Помимо перечисленного у окружности имеется функционал кнопки, при нажатии на которую открывается меню характеристик персонажа (Рис. 17). Таким образом перейдём к рассмотрению интерактивных окон.



Рис. 17. Окно характеристик персонажа.

Итак, в окне характеристик можно наблюдать: количество накопленных очков улучшения (число сверху посередине), кнопку закрытия окна (крестик справа сверху), а также набор характеристик. При открытии этого окна игра ставится на паузу, а при закрытии продолжается с того же момента. Разберём характеристики персонажа:

1. Здоровье – характеризует то, сколько урона может выдержать персонаж.
2. Мана – характеризует запас энергии для использования дальней атаки.
3. Защита – характеризует то, сколько урона поглотится от удара врага.
4. Урон – характеризует количество здоровья, отнимаемого у врага.
5. Крит шанс – характеризует шанс нанесения врагу повышенного урона.
6. Крит – дополнительная прибавка к урону, наносимому врагу.

Рядом с каждой характеристикой есть кнопка в виде стрелочки, повышающая соответствующую характеристику персонажа, при этом тратя очко улучшения. Это окно имеет собственный скрипт StatsMenu.cs:

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

using UnityEngine.UI;

public class StatsPanel : MonoBehaviour

{

public Text health;

public Text mana;

public Text defense;

public Text damage;

public Text critChance;

public Text critRate;

public Text levelPoints;

private Player player;

private PlayerStats stats;

void Start()

{

player = FindObjectOfType<Player>();

stats = FindObjectOfType<PlayerStats>();

}

// Update is called once per frame

void Update()

{

health.text = stats.maxHealth.ToString();

mana.text = stats.maxMana.ToString();

defense.text = stats.defense.ToString();

damage.text = stats.damage.ToString();

critChance.text = stats.critChance.ToString();

critRate.text = stats.critRate.ToString();

levelPoints.text = player.levelPointsInt.ToString();

}

public void Open()

{

gameObject.SetActive(true);

Time.timeScale = 0f;

}

public void Close()

{

gameObject.SetActive(false);

Time.timeScale = 1f;

}

}

Последнее окно, которое остаётся рассмотреть – окно смерти персонажа (Рис. 18). Оно соответственно открывается при утере игроком всего количества здоровья. При появлении игра ставится на паузу, задний фон краснеет, а в окне предлагается начать заново или выйти из игры. Это окно также имеет отдельный скрипт DeathMenu.cs.

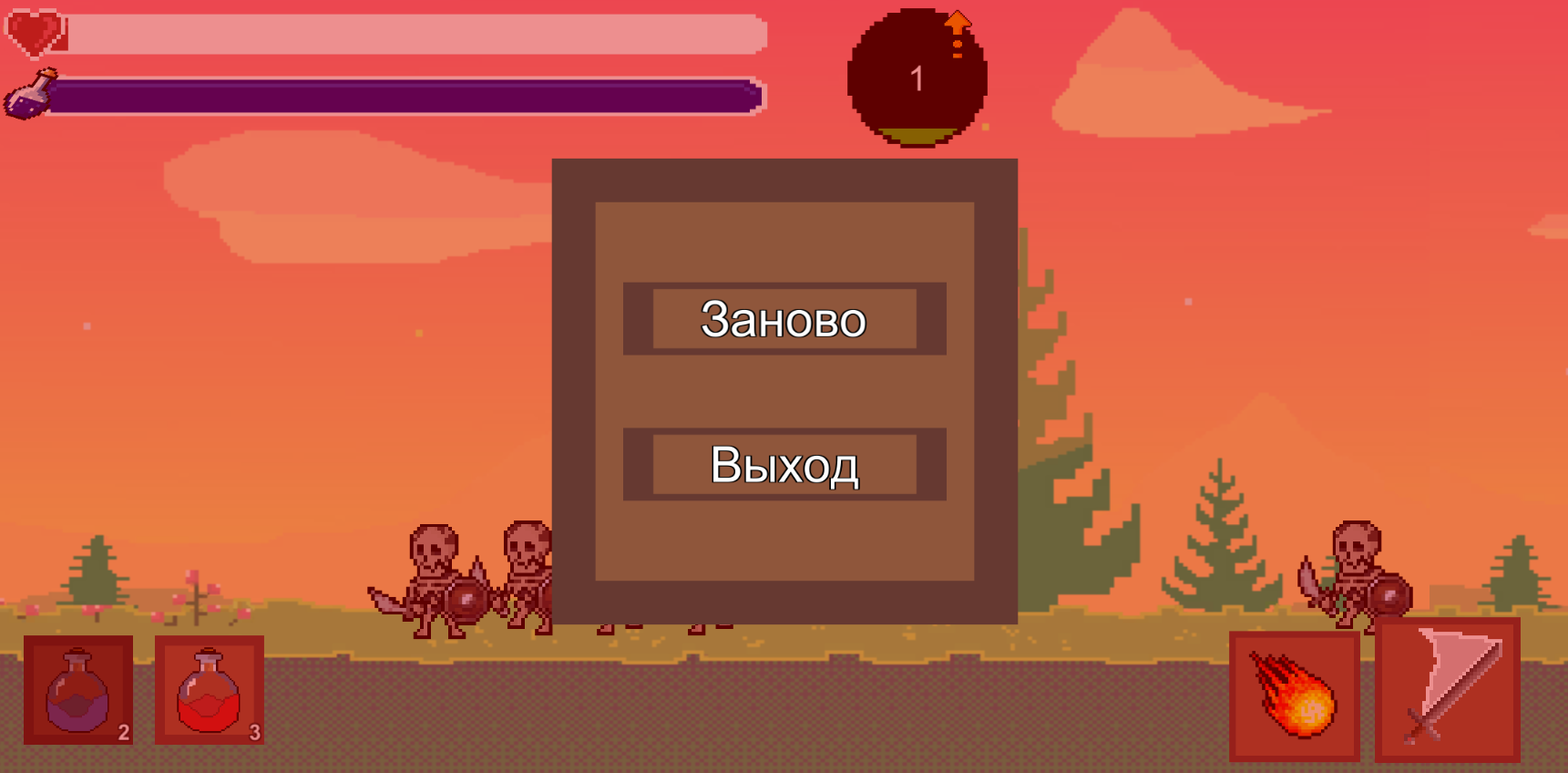


Рис. 18. Меню смерти персонажа.

# ВЫВОД

В ходе выполнения учебной эксплуатационной практики была создана игра, соответствующая требованиям, которые были поставлены изначально. По итогу получилась комплексная игра, которую можно дорабатывать также долго, как и играть в неё. Весь код и компоненты были сформированы так, чтобы можно было пополнять функционал игры. Возможно создание более глобальной идеи на основе полученного результата.

С технической части, для обновления проекта был упрощён доступ к большей части переменных, влияющих на ход работы игры, физику её объектов, а также баланса игрового процесса. Нет никаких ограничений на таким образом, чтобы добавить ещё больше анимации, сделать их плавнее и качественнее.

Интересным решением было бы добавления режима классов персонажей, расширение парка врагов, добавления системы экипировки, а также полноценного инвентаря.

Ещё более масштабной и интересной задумкой является добавление глобального сюжета, главных и дополнительных квестов, системы подземелий или смены биомов, то есть климатических условий, в зависимости от пройденного расстояния/уровня персонажа. Система торговли, крафта, зачарований предметов, создания магии и прочие идеи, которые можно плодить настолько много, насколько это позволяет фантазия. Игра не ограничена в своём потенциале.