СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Темой данной курсовой работы является проектирование и реализация стратегии 2D на кроссплатформенном игровом движке Unity. Unity будет использован только для графического представления игры, а логика будет прописана в прикрепленных скриптах.

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. Выпуск Unity состоялся в 2005 году и с того времени идёт постоянное развитие.

Основными преимуществами Unity являются наличие визуальной среды разработки, межплатформенной поддержки и модульной системы компонентов. К недостаткам относят появление сложностей при работе с многокомпонентными схемами и затруднения при подключении внешних библиотек.

Выживания - режим в компьютерных играх, где игрок должен выживать в игре как можно дольше, пока игра посылает различные задания. Также существует вариант, когда игрок должен достичь какого-либо момента. Особенно распространён этот режим в играх по защите башен, когда игрок должен улучшать части башен, чтобы отражать нарастающие удары противника. Режим можно сравнить с классическими аркадными играми, когда игрок должен улучшать себя, чтобы защититься от противника. Этот режим мог дать непредсказуемое окончание игры, что очень нравилось игрокам. Другой популярной темой режима выживания является борьба с естественными потребностями человека: необходимостью есть, спать, сохранять тепло и душевное равновесие.

Популярнейшая серия игр с таким режимом с противниками-зомби - это Left 4 Dead, а также игра из серии Call of Duty: Call of Duty: World at War, игра по защите башен Plants vs. Zombies и Gears of War 2. Также, игры-песочницы, такие как Minecraft, дают преимущество: монстры появляются только ночью.

В режиме выживания нередко встречается режим перманентной смерти: когда со смертью главного героя игра заканчивается без возможности загрузить ранее сохраненный прогресс (примеры: DayZ, Don't Starve, разрабатываемая The Long Dark).

1 АНАЛИЗ ИСХОДНЫХ ДАННЫХ

RPG – это жанр компьютерных игр, основанный на элементах игрового процесса традиционных настольных ролевых игр. В ролевой игре игрок управляет одним или несколькими персонажами, каждый из которых описан набором численных характеристик, списком способностей и умений; примерами таких характеристик могут быть очки здоровья (англ. hit points, HP), показатели силы, ловкости, интеллекта, защиты, уклонения, уровень развития того или иного навыка и т. п.

У компьютерных ролевых игр много общего с настольными ролевыми играми наподобие Dungeons and Dragons - жаргон, сеттинги, механика игрового процесс. Обычно игрок управляет одним или несколькими главными героями («партией») и добивается победы, выполняя задания («квесты»), участвуя в тактических боях и доходя до самого конца сюжета. Ключевая особенность CRPG в том, что персонажи растут в способностях, и этот рост, зачастую, контролируется игроком. CRPG, за исключением поджанра «action RPG», редко полагаются на физическую координацию и реакцию.

Ролевые игры обычно полагаются на продуманный сюжет и игровой мир. Сюжет обычно делится на серию заданий. Игрок отдаёт персонажу команды, и он выполняет их в соответствии с численными показателями, отвечающими за качество выполнения команды. Когда персонаж набирает определённое количество очков опыта, он получает очередной уровень, и эти показатели увеличиваются. В жанре ролевых игр часто предлагается более сложное и динамичное взаимодействие игрока с компьютерными персонажами, чем в других жанрах - это может обеспечиваться как развитым искусственным интеллектом, так и жёстко запрограммированным поведением персонажей.

Для разработки данного курсового проекта будут использованы кроссплатформенный игровой движок Unity 2019.4.32f1 (только для графического представления игры), среда разработки JetBrains Rider 2021.2.2 и C# в качестве языка программирования (для реализации логики и анимации).

2 ПРОГРАММНОЕ ПРОЕКТИРОВАНИЕ

2.1 Меню

Меню является важной составляющей любого приложения, так как если оно устроенно удобно и понятно, то пользователь будет легко ориентироваться в нем и больше шансов, что он воспользуется этим приложением ещё не один раз. Порой из-за непонятного и заумного меню люди отказываются от той или иной программы, так как просто не понимают, как ей пользоваться.

Игровое меню данной стратегии будет разделено на главное меню и игровое.

В главном меню будут доступны следующие пункты:

– Play;
– Options (с выбором уровня сложности);
– QUIT.

Примеры главного меню и выбора уровня сложности игры представлены на рисунках 2.1

Рисунок 2.1 – Главное меню

В меню опций пользователь сможет включить, выключить, выбрать громкость музыки, а также выбрать игра будет простой или же сложной. На рисунке 2.2 представлен пример опционального подменю.

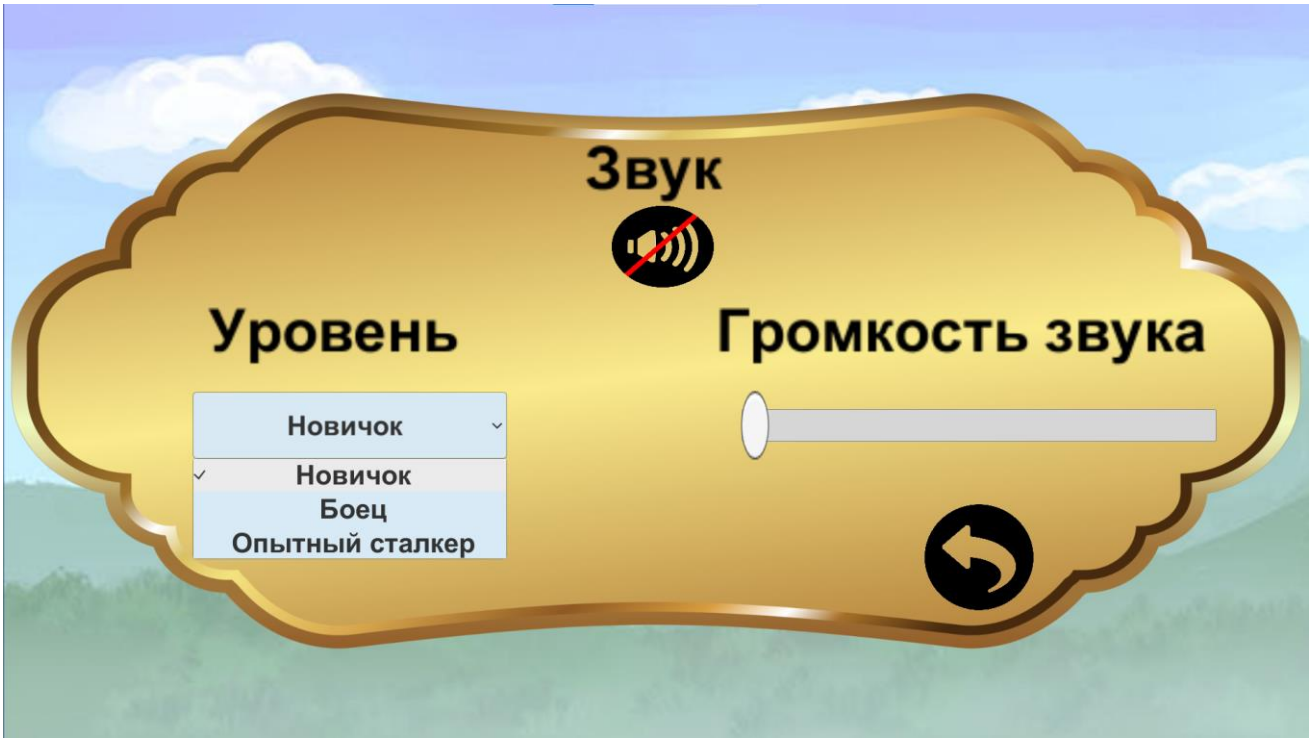


Рисунок 2.2 - Опции

В игровом меню игрок сможет выбрать среди следующих действий:

– Continue (вернуться к игре);
– Exit (перейти в главное меню);

Пример реализации контекстного меню можно увидеть на рисунке 2.3.

Рисунок 2.3 – Игровое меню

2.2 Игровой процесс

Игровой процесс будет построен на следующих принципах (основные):

– Игровой рельеф генерируется каждый раз по-новому;
– Противники и ресурсы появляются в случайных местах;
– Изначально у вас в арсенале имеется только пистолет;
– Вам необходимо выжить среди средневековых людей;

Пример реализации начальной карты можно увидеть на рисунке 2.4.

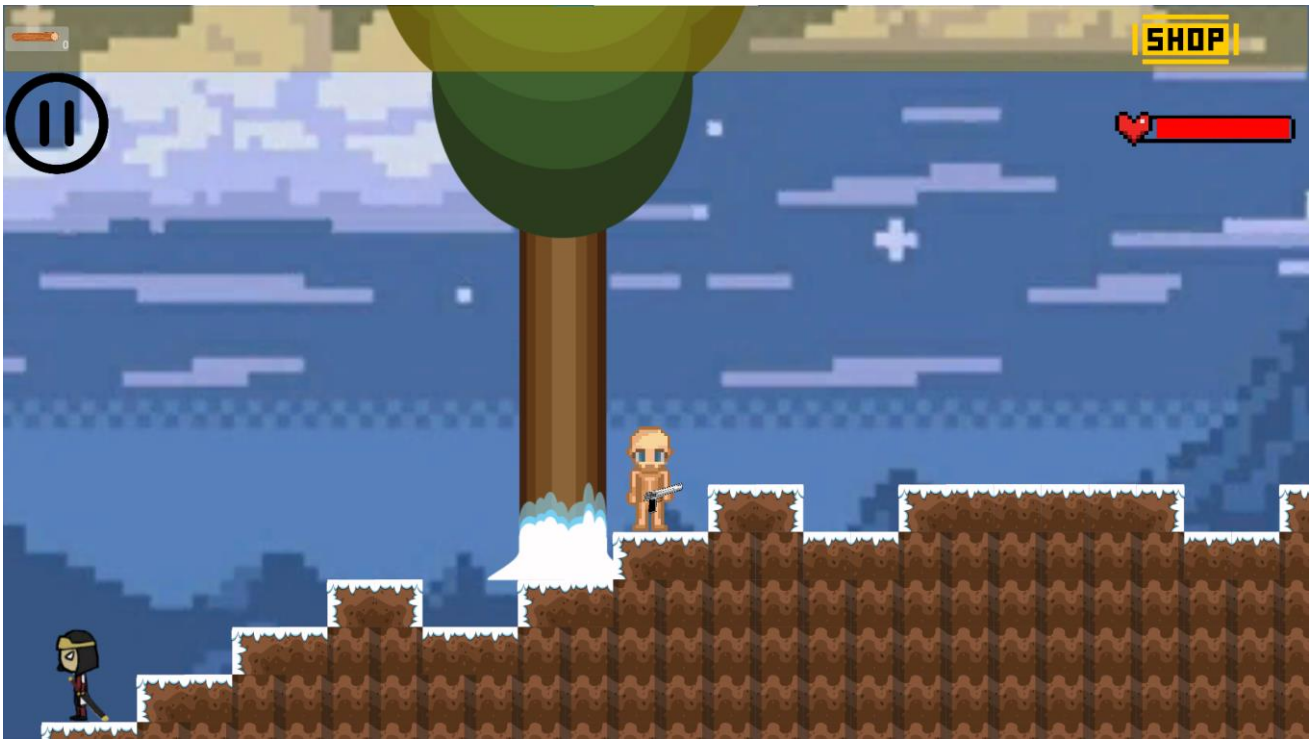


Рисунок 2.4 – Начальная карта

В скором времени, когда вы накопите необходимое количество ресурсов, можно будет улучшить своё вооружение, прикупив его в магазине. Для этого необходимо нажать на кнопку в верхнем правом углу под названием Shop. После чего вам предоставится возможность ознакомится и, возможно, что-то прикупить. Пример вооружения представлен на рисунке 2.5

Рисунок 2.5 – Пример вооружения

Там же, в верхнем левом углу, расположен индикатор здоровья вашего персонажа, который изменяется в зависимости от удара, который пришелся по вам. Пример неполного здоровья представлен на рисунке 2.6

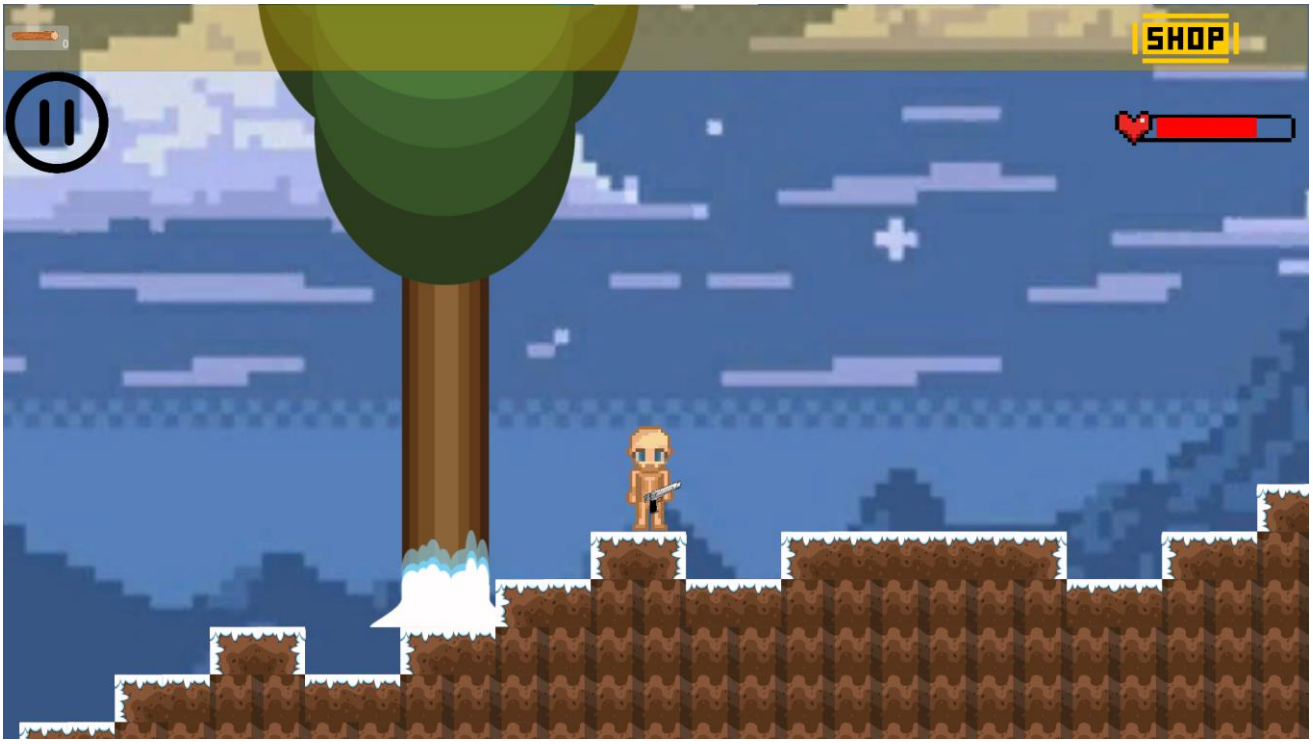


Рисунок 2.6 – Пример неполного здоровья

2.3 Боёвка

В ходе выживания вам как-то придётся давать отпор средневековым людям, именно для этого у вас, успешного инженера, изначально имеется отличное огнестрельное орудие, девять миллиметров, которого смогут дать отпор любому противнику. Пример стрельбы представлен на рисунке 2.7

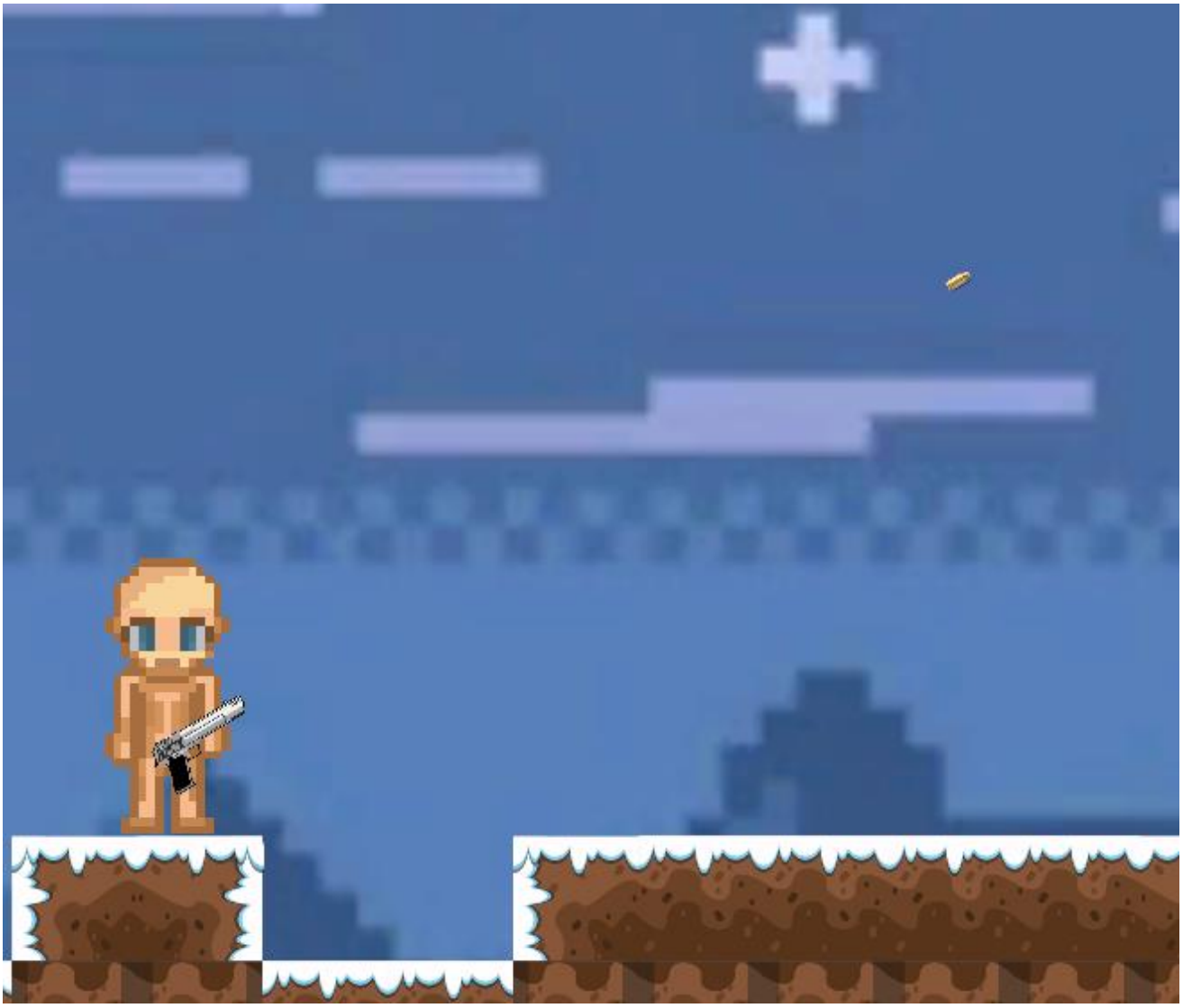


Рисунок 2.8 – Пример стрельбы

3 ПРОГРАММНАЯ РЕАЛИЗАЦИЯ

3.1 Реализация меню

Перед началом создания меню создаем сцену. После этого добавляем UI объект «Canvas». Для этого в окне «Hierarchy» кликаем правой кнопкой мыши и появившемся контекстном меню выбираем следующее: «UI»→«Canvas». Теперь необходимо его настроить. В разделе «Canvas» в пункте «Render Mode» выбираем «Screen Space - Camera» и перетаскиваем в пункт «Render Camera» камеру, чтобы камера смотрела на «Canvas» и все элементы были видны в дальнейшем. Далее в разделе «Canvas Scaler» в пункте «UI Scale Mode» выбираем «Scale With Screen Size», а пункте «Reference Resolution» задаем размеры окна для игры. В моём случае это 1920x1080. Настройка «Canvas» показана на рисунке 3.1.

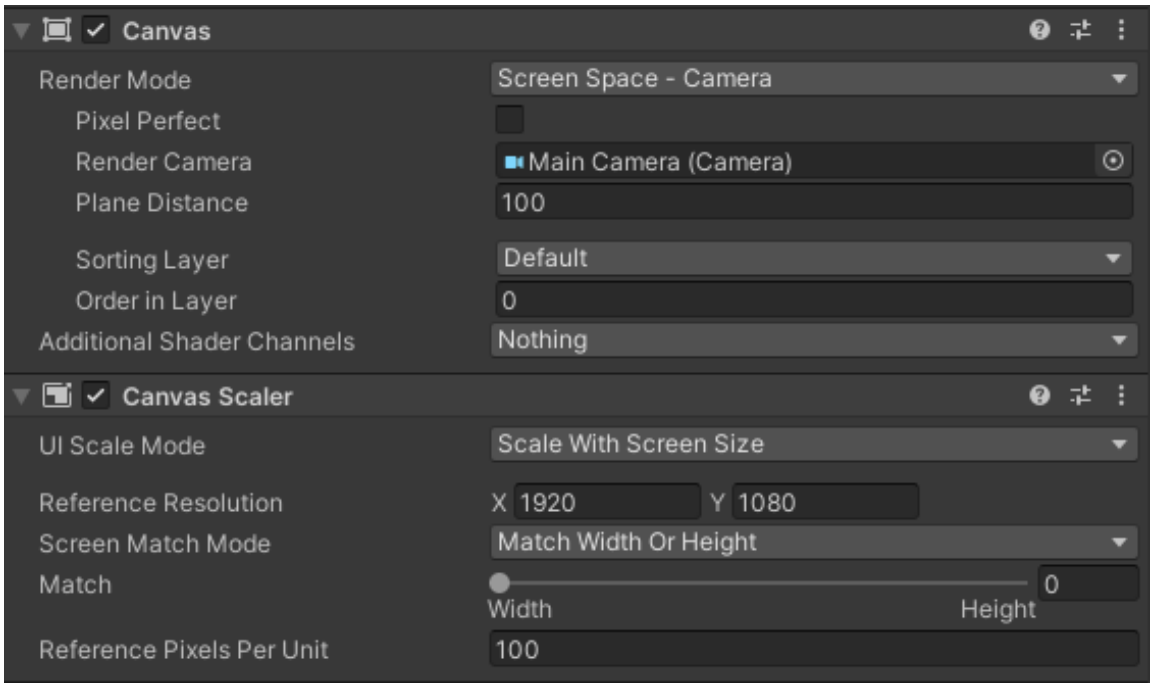


Рисунок 3.1 – Настройка «Canvas»

Теперь перейдем к созданию графики. Для этого в том же окне «Hierarchy» кликаем правой кнопкой мыши и появившемся контекстном меню выбираем следующее:

– «UI»→«Text» (для текстовый элементов);
– «UI»→«Image» (для графиеских изображений);
– «UI»→«Button» (для кнопок);
– «UI»→«Dropdown» (для выпадающего списка);
– «UI»→«Slider» (для слайдера);
– «UI»→«Toggle» (для переключателя).

После добавления всех элементов и их размещения в нужном иерархическом порядке, получим следующее (рисунок 3.2).

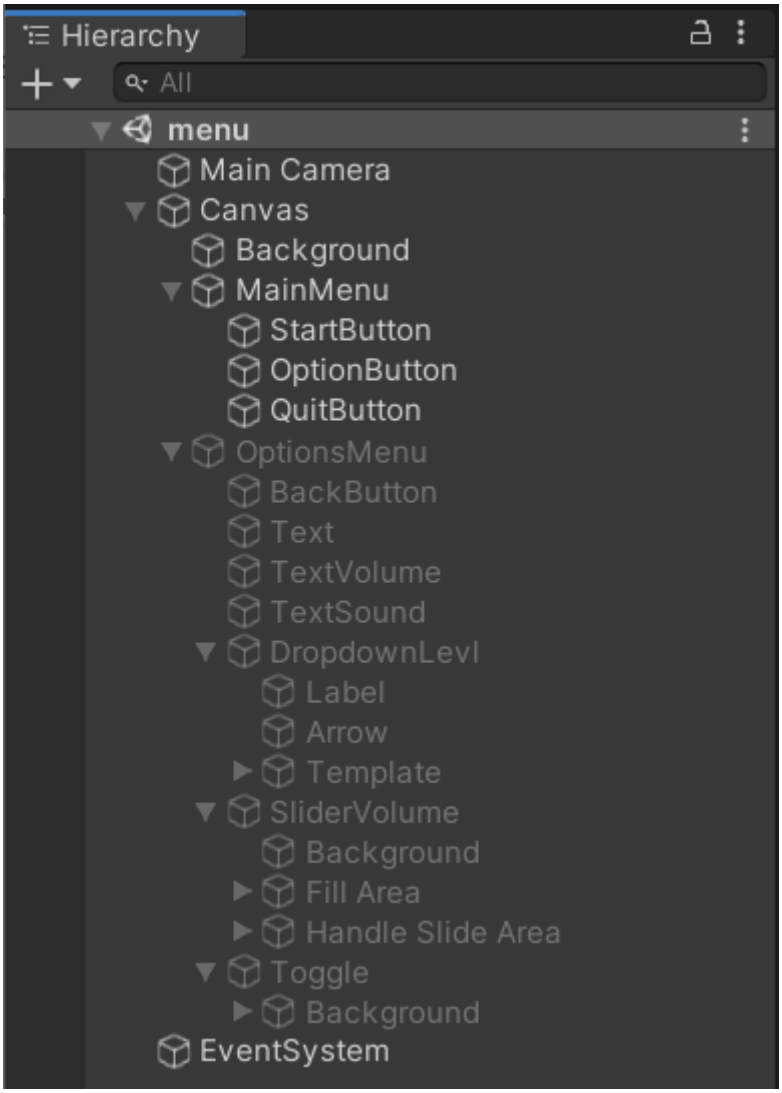


Рисунок 3.2 – Размещение элементов в иерархическом порядке

Далее создадим скрипт и пропишем логику для этих элементов. Для этого в окне «Project» кликаем правой кнопкой мыши и в контекстном меню выбираем следующее: «Create»→«C# Script». Теперь в открывшемся скрипте добавляем библиотеки «using UnityEngine» и «using UnityEngine.SceneManagement», чтобы можно было обращаться к элементам Unity. Класс MainMenu представлен ниже в листинге 3.1.

Листинг 3.1 - Класс MainMenu

```
1.   public class MainMenu : MonoBehaviour
2.   {
3.   public void PlayGame()
4.   {
5.   SceneManager.LoadScene(SceneManager.GetActiveScene().buildInde
x + 1);}
6.   public void ExitGame()
7.   {
9.Debug.Log("Игра закрылась");
10.Application.Quit();
11.  }}
```

Далее создадим «OptionMenu.cs», где будет прописана логика для меню опций (листинг 3.2).

Листинг 3.2 - Класс OptionMenu

```
1.   public class OptionMenu : MonoBehaviour
2.   {
3.   private void Start()
4.   {
5.   AudioListener.pause = !AudioListener.pause;
6.   }
7.   public AudioMixer audioMixer;
8.   public void Setvolume(float volume)
9.   {
10.  audioMixer.SetFloat("volume", Mathf.Log10(volume) * 20);
11.  }
12.  public void Sound()
13.  {
14.  AudioListener.pause = !AudioListener.pause;
15.  }
16.  }
```

PlayerPrefs необходим, чтобы сохранить данные при переходе между сценами. Для проигрывания музыки необходим элемент «Audio Source». Его настройка показана на рисунке 3.3.

Теперь прикрепим функции к элементам, после нажатия на которые они должны быть выполнены. Рассмотрим на примере меню опций. Для этого перетаскиваем скрипт «OptionMenu.cs» в элемент «Canvas». Далее в появившейся раздел перетаскиваем необходимые элементы, как показано на рисунке 3.4.

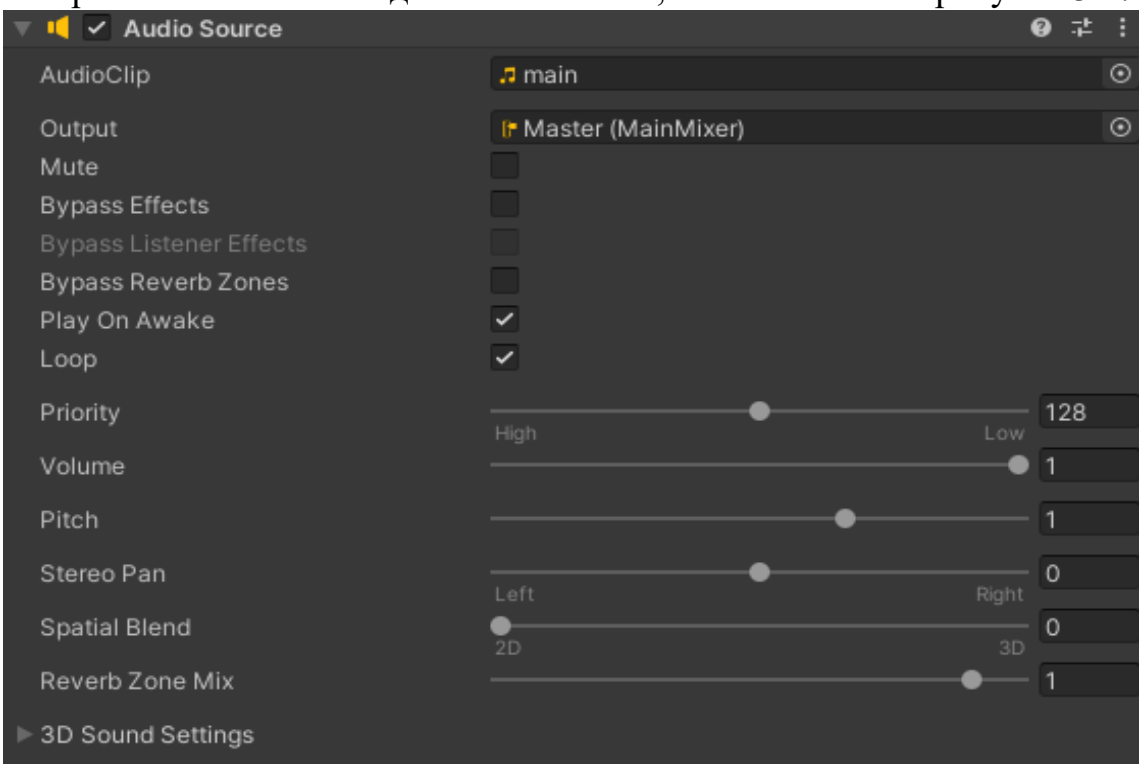


Рисунок 3.3 – Настройка «Audio Source»

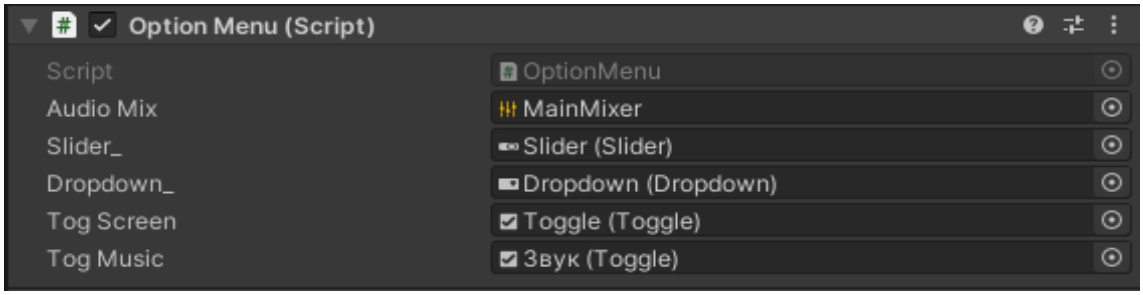


Рисунок 3.4 – Настройка «OptionMenu»

Скрипт «MainMenu.cs» настраивается аналогично.

Теперь настроим элементы. Рассмотрим на примере кнопки. Для этого выбираем эту кнопку. Далее находим раздел «Button» в окне «Inspector». Теперь нажимаем на плюс в «On Click ()» и перетаскиваем элемент, к которому прикреплен скрипт. После этого находим нужную функцию и вводим данные, которые должны отправиться на обработку после нажатия на эту кнопку. На рисунке 3.5 можно увидеть пример настройки кнопки «Легкий».

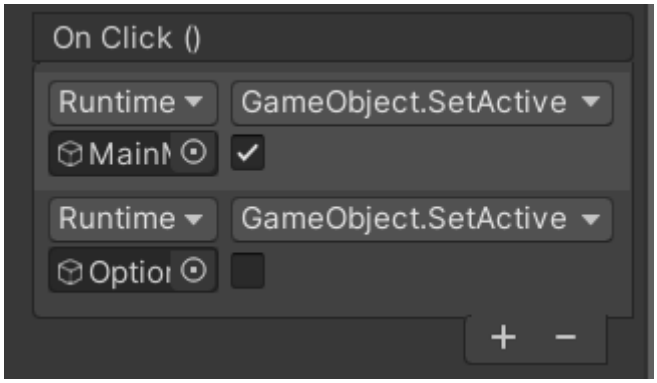


Рисунок 3.5 – Настройка кнопки «Легкий»

Остальные элементы настраиваются аналогично.

Теперь рассмотрим функции для меню во время игры.

В игровом меню будут доступны следующие действия:

– Continue (Продолжить);
– Exit (Выход в главное меню);

Реализация кнопок «Продолжить», «Главное меню» простая: в первом случае окно меню просто будет остановлена, и игра продолжится, во втором вернёт в главное меню.

3.2 Реализация игрового процесса

Настройка параметров для «Canvas» и добавление элементов такое же как в разделе 3.1. Только здесь, как и в разделе 3.3, необходимо добавить такой 2D объект как «Tilemap». С его помощью можно сделать карту.

Для реализации игрового процесса необходимы следующие функции:

- генерация карты;
- генерация ланшафта и противников;
- функции для взаимодействия с противником и ланшафтом;
- улучшение персонажа;
- управление ресурсами;
- рюкзак, для инвентаря;
- функции для стрельбы.

Начнём с генерации карты и её рендеринга. Для это создадим новые скрипты под названием «TerrainGenerator» и «TilemapRender». Данные функции можно увидеть в листинге 3.6 и 3.7.

Листинг 3.6 – Функция TerrainGenerator

```
1.   public class TerrainGenerator : MonoBehaviour
2.   {
3.   public int Height, Width;
4.   public GroundTile Tile;
5.   [SerializeField] private bool _isGenerateOnStart;
6.   public void Start()
7.   {
8.   if (_isGenerateOnStart)
9.   {
10.  GenerateAndRender();
11.  }
12.  }
13.  public void GenerateAndRender()
14.  {
15.  var tileMap = Generate();
16.  GetComponent<TilemapRender>().Render(tileMap);
17.  GetComponent<PolygonCollider2D>().points =
tileMap.GetClosedMesh();
18.  }
19.  private ITilemap Generate()
20.  {
21.  int groudHeight = 5;
22.  HeightmapeBasedTilemap tilemap = new
HeightmapeBasedTilemap(Width, Tile);
23.  for (int x = 0; x < Width; x++)
24.  {
25.  if (x % 2 == 0)
26.  {
27.  if (groudHeight > 1)
28.  {
29.  groudHeight += Random.Range(-1, 2);
30.  }else groudHeight += Random.Range(1, 2);
31.  }
32.  tilemap.SetHeight(x,groudHeight);
33.  }
```

Продолжение листинга 3.6

```
34.  return tilemap;
35.  }
36.  }
37.  namespace CustomTilemap
38.  {
39.  public interface ICell
40.  {
41.  void Refresh(Vector2Int position, ITilemap tilemap, GameObject
gameObject);
42.  }
43.  public interface ITilemap
44.  {
45.  int Count { get; }
46.  int Height{ get; }
47.  int Width{ get; }
48.  ICell GetCell(Vector2Int position);
49.  Vector2[] GetClosedMesh();
50.  }
51.  [CreateAssetMenu(menuName = "GroundTile")]
52.  public class GroundTile : ScriptableObject, ICell {
53.  public Sprite Left, Right, Top, Bottom, TopLeft, TopRight,
BottomLeft, BottomRight, Other;
54.  public void Refresh(Vector2Int position, ITilemap
tilemap,GameObject gameObject)
55.  {
56.  SpriteRenderer render =
gameObject.GetComponent<SpriteRenderer>();
57.  render.sprite = Other;
58.  if (Exist(position + Vector2Int.right, tilemap) &&
59.  Exist(position + Vector2Int.left, tilemap) &&
60.  !Exist(position + Vector2Int.up, tilemap))
61.  {
62.  render.sprite = Top;
63.  }else if (Exist(position + Vector2Int.right, tilemap) &&
64.  !Exist(position + Vector2Int.left, tilemap) &&
65.  !Exist(position + Vector2Int.up, tilemap))
66.  {
67.  render.sprite = TopLeft;
68.  }else if (!Exist(position + Vector2Int.right, tilemap) &&
69.  Exist(position + Vector2Int.left, tilemap) &&
70.  !Exist(position + Vector2Int.up, tilemap))
71.  {
72.  render.sprite = TopRight;
73.  }
74.  }
75.  public bool Exist(Vector2Int position, ITilemap tilemap)
76.  {
77.  if (position.x < 0 || position.x >= tilemap.Width) return
false;
78.  var tile = tilemap.GetCell(position);
```

Продолжение листинга 3.6

```
79.  return tile != null;
80.  }
81.  }
82.  public class HeightmapeBasedTilemap : ITilemap
83.  {
84.  public int Count
85.  {
86.  get
87.  {
88.  return _height.Sum();
89.  }
90.  }
91.  public int Height
92.  {
93.  get
94.  {
95.  return _height.Max();
96.  }
97.  }
98.  public int Width => _height.Length;
99.  private int[] _height;
100. private ICell _cell;
101. public HeightmapeBasedTilemap(int width, ICell cell)
102. {
103. _height = new int[width];
104. _cell = cell;
105. }
106. public void SetHeight(int x, int value)
107. {
108. if (x < 0 && x < _height.Length) throw new
ArgumentOutOfRangeException("x");
109. _height[x] = value;
110. }
111. public ICell GetCell(Vector2Int position)
112. {
113. if (position.x < 0 && position.x >= _height.Length) throw new
ArgumentOutOfRangeException("x");
114. if (position.y > _height[position.x])
115. {
116. return null;
117. }
118. else
119. {
120. return _cell;
121. }
122. }
123. public Vector2[] GetClosedMesh()
124. {
125. List<Vector2> points = new List<Vector2>();
126. for (int x = 0; x < Width; x++)
```

Продолжение листинга 3.6

```
127. points.Add(new Vector2(x - 0.5f, _height[x] + 0.5f));
128. points.Add(new Vector2(x + 0.5f, _height[x] + 0.5f));
129. }
130. points.Add(new Vector2(Width - 0.5f, - 0.5f));
131. points.Add(new Vector2(-0.5f, - 0.5f));
132. return points.ToArray();
133. }
134. }}
```

Листинг 3.7 – Функция TilemapRender

```
1.   public class TilemapRender : MonoBehaviour
2.   {
3.   public void Render(ITilemap tilemap)
4.   {
5.   Clear();
6.   for (int x = 0; x < tilemap.Width; x++)
7.   {
8.   for (int y = 0; y <= tilemap.Height; y++)
9.   {
10.  var cell = tilemap.GetCell(new Vector2Int(x, y));
11.  if (cell != null)
12.  {
13.  GameObject cellGo = CreateEmpty(new Vector2Int(x, y));
14.  cell.Refresh(new Vector2Int(x,y), tilemap, cellGo);
15.  }
16.  }
17.  }
18.  }
19.  public void Clear()
20.  {
21.  foreach (Transform child in
transform.OfType<Transform>().ToList())
22.  {
23.  #if UNITY_EDITOR
24.  DestroyImmediate(child.gameObject);
25.  #else
26.  Destroy(child.gameObject);
27.  #endif
28.  }
29.  }
30.  public GameObject CreateEmpty(Vector2Int vector2Int)
31.  {
32.  GameObject result = new GameObject(ToString());
33.  var transform = result.GetComponent<Transform>();
34.  transform.parent = GetComponent<Transform>();
35.  transform.localPosition = new Vector3Int(vector2Int.x,
vector2Int.y, 0);
36.  }
37.  }
38.  result.AddComponent<SpriteRenderer>();
```

Продолжение листинга 3.7

```
39.  return result;
40.  }
41.  }
42.  }
```

Для взаимодействия персонажа с врагами, используются функции «Enemy» и «EnemyIteract», реализация которых представлена в листинге 3.8 и 3.9.

Листинг 3.8 – Функция Enemy

```
1.   public class Enemy : MonoBehaviour
2.   {
3.   [SerializeField] private float attackCooldown;
4.   [SerializeField] private float range;
5.   [SerializeField] private float colliderDistance;
6.   [SerializeField] private float damage;
7.   [SerializeField] private BoxCollider2D boxCollider;
8.   [SerializeField] private LayerMask playerLayer;
9.   private float cooldownTimer = Mathf.Infinity;
10.  private Animator anim;
11.  private PlayerInterface health;
12.  public float healthEnemy;
13.  public float speed;
14.  private Transform target;
15.  private float muveInput;
16.  public bool facingRight = false;
17.  private void Start()
18.  {
19.  target =
GameObject.FindGameObjectWithTag("Player").GetComponent<Transform>(
);
20.  }
21.  private void Awake()
22.  {
23.  anim = GetComponent<Animator>();
24.  }
25.  void Flip()
26.  {
27.  Vector3 Scaler = transform.localScale;
28.  if (muveInput < 0)
29.  {
30.  //facingRight = false;
31.  Scaler.x *= 1;
32.  }else if (muveInput > 0 /*&& facingRight == false*/)
33.  {
34.  // facingRight = true;
35.  Scaler.x *= -1;
36.  }
37.  transform.localScale = Scaler;
38.  }
```

Продолжение листинга 3.8

```
39.  }
40.  void DieEnemy()
41.  {
42.  if (healthEnemy <= 0)
43.  {
44.  Destroy(gameObject);
45.  }
46.  }
47.  private void Update()
48.  {
49.  Flip();
50.  if (Vector2.Distance(transform.position, target.position) < 5
|| healthEnemy < 1f)
51.  {
52.  muveInput = Input.GetAxis("Horizontal");
53.  transform.position = Vector2.MoveTowards(transform.position,
54.  target.position, speed * Time.deltaTime);
55.  if (muveInput == 0)
56.  {
57.  anim.SetBool("muving",false);
58.  }
59.  else
60.  {
61.  anim.SetBool("muving",true);
62.  }
63.    if (facingRight == false && muveInput < 0) Flip();
64.    if (facingRight == true && muveInput > 0) Flip();
65.  }
66.  cooldownTimer += Time.deltaTime;
67.  if (PlayerInSight())
68.  {
69.  if (cooldownTimer >= attackCooldown)
70.  {
71.  cooldownTimer = 0;
72.  anim.SetTrigger("mileeAtack");
73.  }
74.  }
75.  }
76.  // ReSharper disable Unity.PerformanceAnalysis
77.  private bool PlayerInSight()
78.  {
79.  RaycastHit2D hit = Physics2D.BoxCast(boxCollider.bounds.center
+ transform.right * range * transform.localScale.x
*colliderDistance,
80.  new Vector3(boxCollider.bounds.size.x *
range,boxCollider.bounds.size.y,boxCollider.bounds.size.z),
81.  0, Vector2.left, 0, playerLayer);
82.  if (hit.collider != null)
83.  health = hit.transform.GetComponent<PlayerInterface>();
84.  return hit.collider != null;
```

Продолжение листинга 3.8

```
85.  }
86.  private void OnDrawGizmos()
87.  {
88.  Gizmos.color = Color.red;
89.  Gizmos.DrawWireCube(boxCollider.bounds.center +
transform.right*range * transform.localScale.x * colliderDistance,
90.  new Vector3(boxCollider.bounds.size.x *
range,boxCollider.bounds.size.y,boxCollider.bounds.size.z));
91.  }
92.  private void DamagePlayer()
93.  {
94.  if (PlayerInSight())
95.  {
96.  health.TakeDamage(damage);
97.  }
98.  }
99.  public void TakeDamage(float damage)
100. {
101. healthEnemy -= damage;}
```

Листинг 3.9 – Функция EnemyIteract

```
1.   public class EnemeIteract : MonoBehaviour
2.   {
3.   private float health = 1f;
4.   private float damage = 0.25f;
5.   private void OnTriggerEnter2D(Collider2D other)
6.   {
7.   if (other.name == "Player")
8.   {
9.   other.GetComponent<PlayerInterface>().TakeDamage(damage);
10.  }
11.  }
12.  public void TakeDamage(float amount)
13.  {
14.  health -= amount;
15.  if (health <= 0f)
16.  {
17.  Destroy(gameObject);
18.  }
19.  }}
```

Для взаимодействия с ланшафтом местности были написаны функции «Tree» и «DropedItem». Листинг с номером 3.10 и 3.11.

Листинг 3.10 – Функция Tree

```
1.   public class Tree : MonoBehaviour
2.   {
3.   public float Health = 100;
4.   public GameObject TreeDropedElement;
5.   public Sprite Destroyed;
6.   void Start()
```

Продолжение листинга 3.10

```
7.    }
8.    public void OnClick()
9.    {
10.   Health -= 25;
11.   if (Health <= 0)
12.   {
13.   int count = Random.Range(4, 6);
14.   var p = transform.position;
15.   for (int i = 0; i < count; i++)
16.   {
17.   var go = Instantiate(TreeDropedElement, new Vector3(p.x, p.y +
i, p.z), Quaternion.identity);
18.   go.GetComponent<DropedItem>().Explode();
19.   }
20.   Destroy(gameObject);
21.   }
22.   }
23.   public void OnEnter()
24.   {
25.   Debug.Log("Enter");
26.   }
27.   public void OnExit()
28.   {
29.   Debug.Log("Exit");
30.   }}
```

Листинг 3.11 – Функция DropedItem

```
1.    public class DropedItem : MonoBehaviour
2.    {
3.    public void OnCollisionEnter2D(Collision2D collision)
4.    {
5.    if (collision.gameObject.tag == "Player")
6.    {
7.    Bag.Instance.AddWood(1);
8.    Destroy(gameObject);
9.    }
10.   }
11.   public void Explode()
12.   {
13.   GetComponent<Rigidbody2D>().AddForce(new
Vector2(Random.Range(-40f,40f), Random.Range(0f,10f)));
14.   }}
```

Для подсчёта собранных ресурсов, необходимо было создание, так называемого рюкзака, для этого использовались функции «Bag» и «BagUI». Листинг предоставлен под номерами 3.11 и 3.12.

Листинг 3.11 – Функция Bag

```
1.    public class Bag : MonoBehaviour
2.    {
```

Продолжение листинга 3.11

```
3.   public int WoodCount { get; private set; }
4.   public static Bag Instance { get; private set; }
5.   public event System.Action<Bag> OnUpdate;
6.   private void Awake()
7.   {
8.   Instance = this;
9.   }
10.  public void AddWood(int count)
11.  {
12.  WoodCount += count;
13.  if (OnUpdate != null)
14.  {
15.  OnUpdate(this);
16.  }
17.  }
18.  void Update()
19.  {
20.  }
21.  private void OnDestroy()
22.  {
23.  Instance = null;}}
```

Листинг 3.12 – Функция BagUI

```
1.   public class BagUI : MonoBehaviour
2.   {
3.   public Text WoodCount;
4.   private void Start()
5.   {
6.   Bag.Instance.OnUpdate += UpdateUI;
7.   }
8.   void UpdateUI(Bag bag)
9.   {
10.  WoodCount.text = bag.WoodCount.ToString();
11.  }
12.  private void OnDestroy()
13.  {
14.  Bag.Instance.OnUpdate -= UpdateUI;}}
```

И на последок, для полноценного выживания не хватало оружия, поэтому были созданы функции «Gun» и «Bulet», листинги которых предоставлены под номером 3.13 и 3.14.

Листинг 3.13 – Функция Gun

```
1.   public class Gun : MonoBehaviour
2.   {
3.   public float offset;
4.   public GameObject bullet;
5.   public Transform shotPoint;
6.   private SpriteRenderer sr;
7.   private float timeBtwShots;
8.   public float StartTimeBtwShots;
```

Продолжение листинга 3.13

```
9.   void Update()
10.  {
11.  sr = GetComponent<SpriteRenderer>();
12.  Vector3 difference =
Camera.main.ScreenToWorldPoint(Input.mousePosition) -
transform.position;
13.  float rotZ = Mathf.Atan2(difference.y, difference.x) *
Mathf.Rad2Deg;
14.  transform.rotation = Quaternion.Euler(0f, 0f, rotZ + offset);
15.  if ((transform.rotation.z) < 180 && (transform.rotation.z) >
360)
16.  { sr.flipY = true; sr.flipX = true; }
17.  else { sr.flipY = false; sr.flipX = false;}
18.  if (timeBtwShots <= 0)
19.  {
20.  if (Input.GetMouseButton(0))
21.  {
22.  Instantiate(bullet, shotPoint.position, transform.rotation);
23.  timeBtwShots = StartTimeBtwShots;
24.  }
25.  }
26.  else
27.  {
28.  timeBtwShots -= Time.deltaTime;
29.  }
30.  }}
```

Листинг 3.14 – Функция Bulet

```
1.   public class Bulet : MonoBehaviour
2.   {
3.   public float speed;
4.   public float lifetime;
5.   public float distance;
6.   public float damage;
7.   public LayerMask whatIsSolid;
8.   void Update()
9.   {
10.  RaycastHit2D hitInfo = Physics2D.Raycast(transform.position,
transform.up, distance, whatIsSolid);
11.  if (hitInfo.collider != null)
12.  {
13.  if (hitInfo.collider.CompareTag("Enemy"))
14.  {
15.  hitInfo.collider.GetComponent<Enemy>().TakeDamage(damage);
16.  }
17.  Destroy(gameObject);
18.  }
19.  transform.Translate(Vector2.up * speed * Time.deltaTime);}}
```

3.3 Сопроводительная документация

Сопроводительная документация по разработанному программному продукту предоставляется в составе технического задания (приложение А) согласно ГОСТ 19.201-78.

Требования к сопроводительной документации устанавливаются государственными стандартами ЕСПД.

3.4 Анализ программного обеспечения

Анализ программ - это процесс автоматического анализа поведения компьютерных программ в отношении таких свойств, как корректность, надежность, безопасность и живучесть. Анализ программ фокусируется на двух основных областях: оптимизация программ и корректность программ.

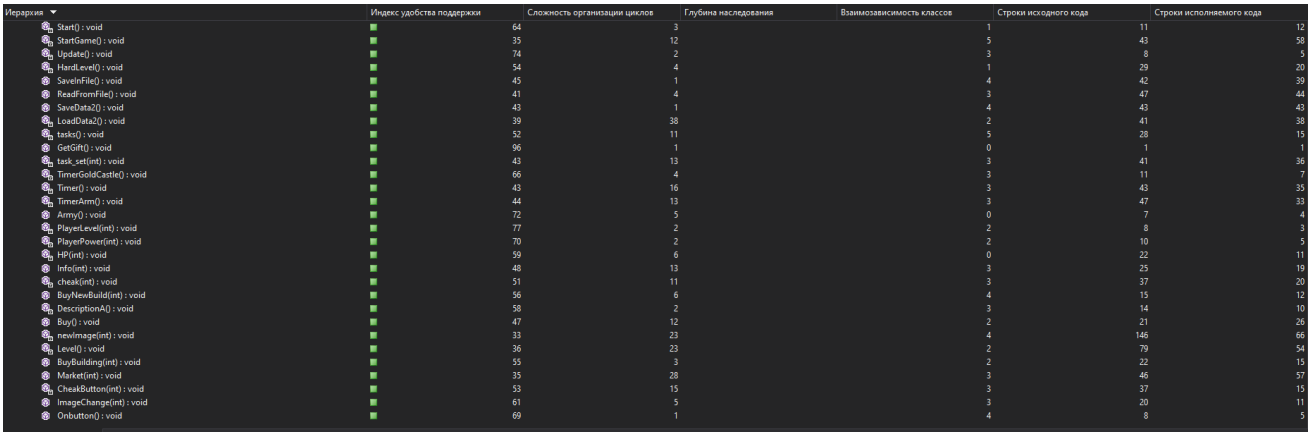
Для анализа данного программного обеспечения используем анализ метрик кода.

Метрика программного обеспечения – мера, позволяющая получить численное значение некоторого свойства программного обеспечения или его спецификаций.

Так как в JetBrains Rider нет встроенного анализа метрик, воспользуемся Microsoft Visual Studio для данного анализа. Для этого перенесем код в проект Visual Studio. При анализе метрик будем учитывать следующие критерии:

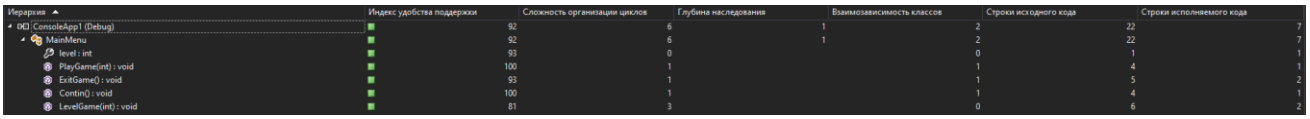
- индекс удобства поддержки – оценивает простоту обслуживания кода;
- сложность организации циклов – определяет число ветвей;
- глубина наследования – определяет число уровней в иерархии наследования объекта;
- взаимозависимость классов – определяет число классов, на которые есть ссылки;
- строки кода – приблизительно оценивает число строк исполняемого кода.

Результат анализа метрик для каждого скрипта представлен ниже на рисунках 3.6-3.10.



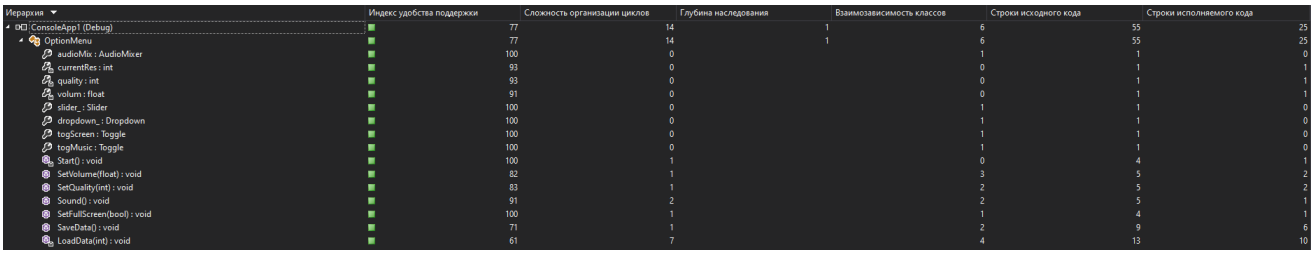
Иерархия	Индекс удобства поддержки	Сложность организации циклов	Глубина наследования	Взаимозависимость классов	Строки исходного кода	Строки исполняемого кода
Start() : void	64	3		1	11	12
StartGame() : void	35	12		5	43	58
Update() : void	74	2		3	8	5
HardLevel() : void	54	4		1	29	20
SaveInFile() : void	45	1		4	42	39
ReadFromFile() : void	41	4		3	47	44
SaveData2() : void	43	1		4	43	43
LoadData2() : void	39	38		2	41	38
tasks() : void	52	11		5	28	15
GetGift() : void	96	1		0	1	1
task_set(int) : void	43	13		3	41	36
TimerGoldCastle() : void	66	4		3	11	7
Timer() : void	43	16		3	43	35
TimerArm() : void	44	13		3	47	33
Army() : void	72	5		0	7	4
PlayerLevel(int) : void	77	2		2	8	3
PlayerPower(int) : void	70	2		2	10	5
HP(int) : void	59	6		0	22	11
Info(int) : void	48	13		3	25	19
cheak(int) : void	51	11		3	37	20
BuyNewBuild(int) : void	56	6		4	15	12
DescriptionA() : void	58	2		3	14	10
Buy() : void	47	12		2	21	26
newImage(int) : void	33	23		4	146	66
Level() : void	36	23		2	79	54
BuyBuilding(int) : void	55	3		2	22	15
Market(int) : void	35	28		3	46	57
CheakButton(int) : void	53	15		3	37	15
ImageChange(int) : void	61	5		3	20	11
Onbutton() : void	69	1		4	8	5

Рисунок 3.6 – Анализ метрик для maingame.cs



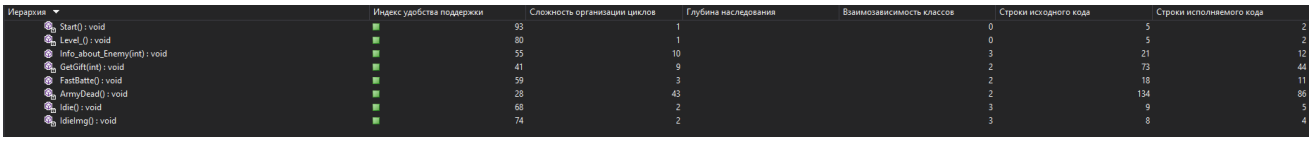
Иерархия	Индекс удобства поддержки	Сложность организации циклов	Глубина наследования	Взаимозависимость классов	Строки исходного кода	Строки исполняемого кода
ConsoleApp1 (Debug)	92	6	1	2	22	7
MainMenu	92	6	1	2	22	7
level : int	93	0		0	1	1
PlayGame(int) : void	100	1		1	4	1
ExitGame() : void	93	1		1	5	2
Contin() : void	100	1		1	4	1
LevelGame(int) : void	81	3		0	6	2

Рисунок 3.7 – Анализ метрик для MainMenu.cs



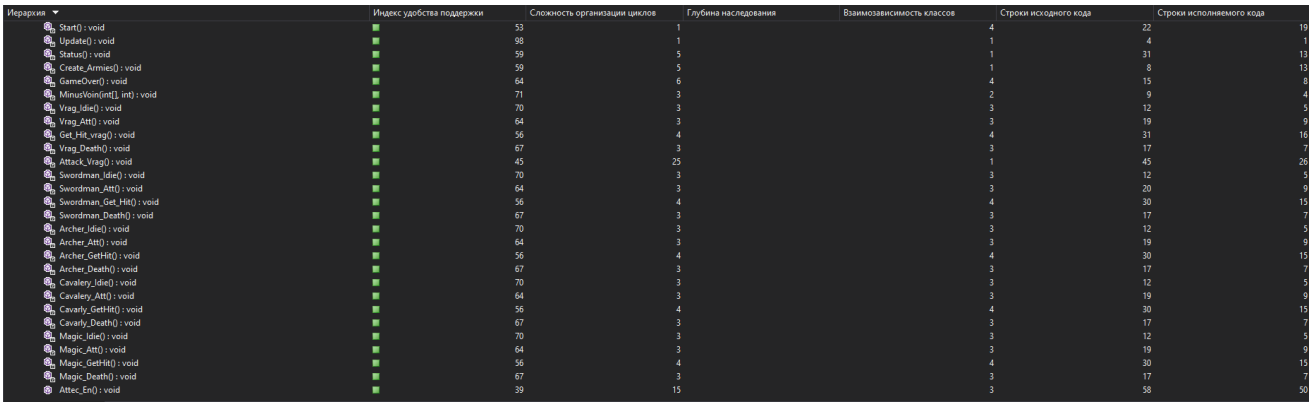
Иерархия	Индекс удобства поддержки	Сложность организации циклов	Глубина наследования	Взаимозависимость классов	Строки исходного кода	Строки исполняемого кода
ConsoleApp1 (Debug)	77	14	1	6	55	25
OptionMenu	77	14	1	6	55	25
audioMix : AudioMixer	100	0		1	1	0
currentRes : int	93	0		0	1	1
quality : int	93	0		0	1	1
volum : float	91	0		0	1	1
slider_ : Slider	100	0		1	1	0
dropdown_ : Dropdown	100	0		1	1	0
togScreen : Toggle	100	0		1	1	0
togMusic : Toggle	100	0		1	1	0
Start() : void	100	1		0	4	1
SetVolume(float) : void	82	1		3	5	2
SetQuality(int) : void	83	1		2	5	2
Sound() : void	91	2		2	5	1
SetFullScreen(bool) : void	100	1		1	4	1
SaveData() : void	71	1		2	9	6
LoadData(int) : void	61	7		4	13	10

Рисунок 3.8 – Анализ метрик для OptionMenu.cs



Иерархия	Индекс удобства поддержки	Сложность организации циклов	Глубина наследования	Взаимозависимость классов	Строки исходного кода	Строки исполняемого кода
Start() : void	93	1		0	5	2
Level_() : void	80	1		0	5	2
Info_about_Enemy(int) : void	55	10		3	21	12
GetGift(int) : void	41	9		2	73	44
FastBatte() : void	59	3		2	18	11
ArmyDead() : void	28	43		2	134	86
Idie() : void	68	2		3	9	5
IdleImg() : void	74	2		3	8	4

Рисунок 3.9 – Анализ метрик для map.cs



Иерархия	Индекс удобства поддержки	Сложность организации циклов	Глубина наследования	Взаимозависимость классов	Строки исходного кода	Строки исполняемого кода
Start() : void	53	1		4	22	19
Update() : void	98	1		1	4	1
Status() : void	59	5		1	31	13
Create_Armies() : void	59	5		1	8	13
GameOver() : void	64	6		4	15	8
MinusVoin(int[], int) : void	71	3		2	9	4
Vrag_Idie() : void	70	3		3	12	5
Vrag_Att() : void	64	3		3	19	9
Get_Hit_vrag() : void	56	4		4	31	16
Vrag_Death() : void	67	3		3	17	7
Attack_Vrag() : void	45	25		1	45	26
Swordman_Idie() : void	70	3		3	12	5
Swordman_Att() : void	64	3		3	20	9
Swordman_Get_Hit() : void	56	4		4	30	15
Swordman_Death() : void	67	3		3	17	7
Archer_Idie() : void	70	3		3	12	5
Archer_Att() : void	64	3		3	19	9
Archer_GetHit() : void	56	4		4	30	15
Archer_Death() : void	67	3		3	17	7
Cavalery_Idie() : void	70	3		3	12	5
Cavalery_Att() : void	64	3		3	19	9
Cavarly_GetHit() : void	56	4		4	30	15
Cavarly_Death() : void	67	3		3	17	7
Magic_Idie() : void	70	3		3	12	5
Magic_Att() : void	64	3		3	19	9
Magic_GetHit() : void	56	4		4	30	15
Magic_Death() : void	67	3		3	17	7
Attec_En() : void	39	15		3	58	50

Рисунок 3.10 – Анализ метрик для battle.cs

По полученным данным можно сделать следующий вывод: результат анализа кода метрик в целом хороший.

3.5 Тестирование программного обеспечения

Для тестирования приложения используем SmokeTest. Smoke Test (дымовое тестирование) – тест проверяющий основную функциональность продукта, а также его работоспособность, по результатам которого принимается решение о приемке версии продукта на дальнейшее тестирование. «Дымовой тест» обычно выполняется самим программистом; не проходившую этот тест программу не имеет смысла отдавать на более глубокое тестирование.

Список тестов, которые прошло игровое приложение, представлен ниже:

- запуск игры;
- начать игру;
- проверка работы случайной генерации мира и ланшафта;
- проверка механики выстрела;
- проверка разрушаемости объектов;
- проверка работы инвентаря;
- проверка взаимодействия с врагами;
- проверка работоспособности магазина;

Запускаем игру. Результат на рисунке 3.11.

Рисунок 3.11 – Успешный запуск игры

Теперь проверим запуск игры. Рисунок 3.12.

Рисунок 3.12 – Запуск игры

Проверим работу случайной генерации мира и ланшафта. Рисунок 3.13

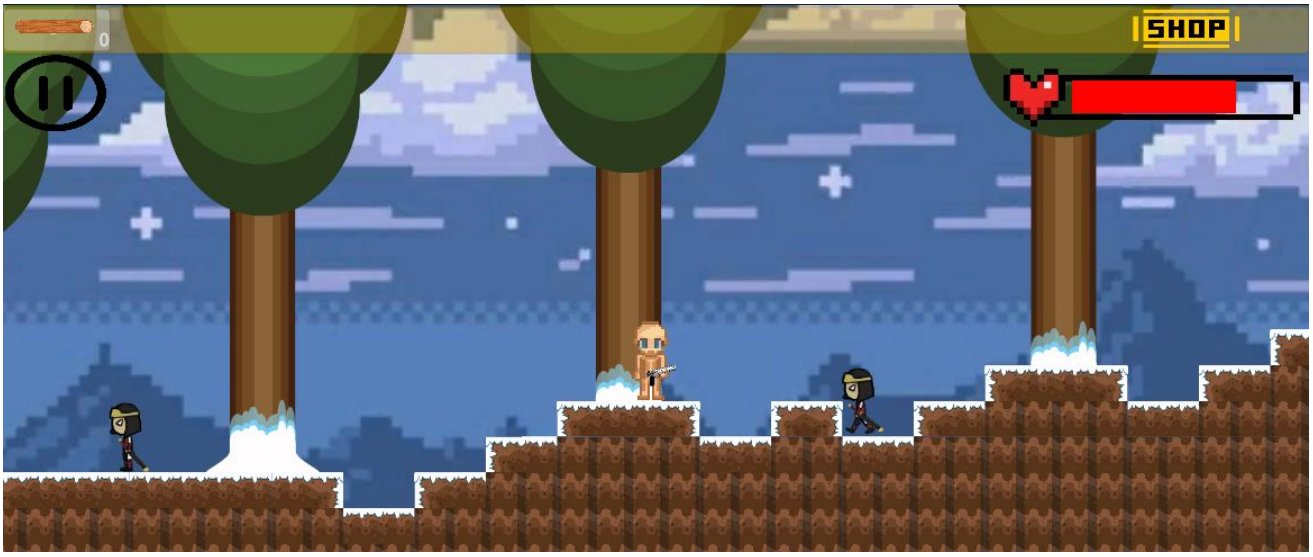


Рисунок 3.13 – Работа случайной генерации мира и ланшафта

Проверка механики выстрела. Рисунок 3.14

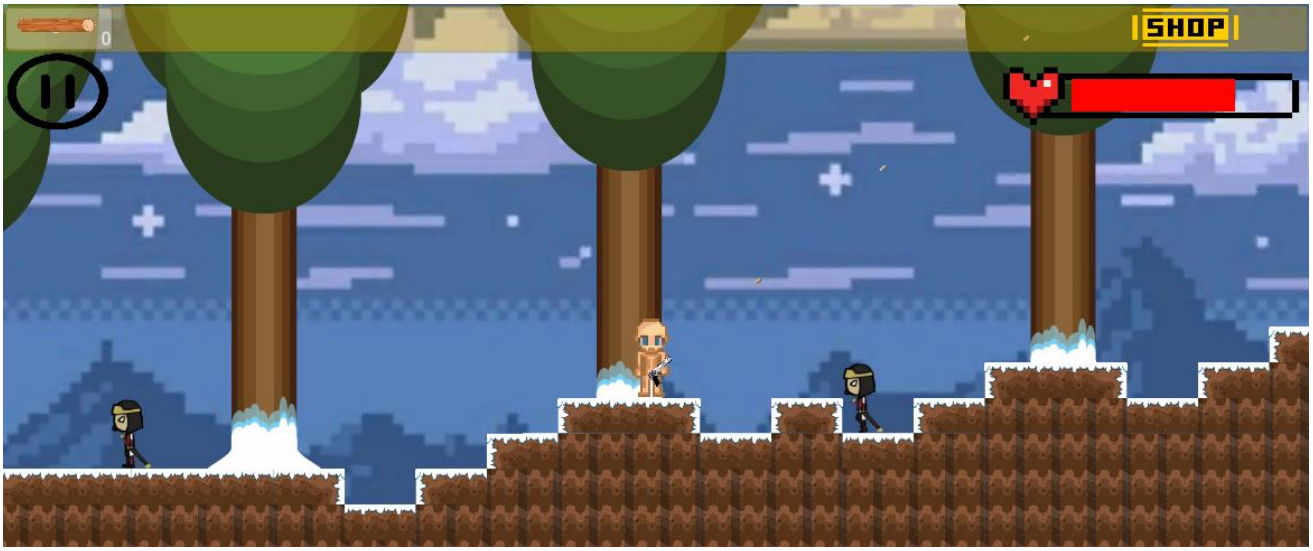


Рисунок 3.14 – проверка выстрела

Проверка разрушаемости объектов. Рисунок 3.15.

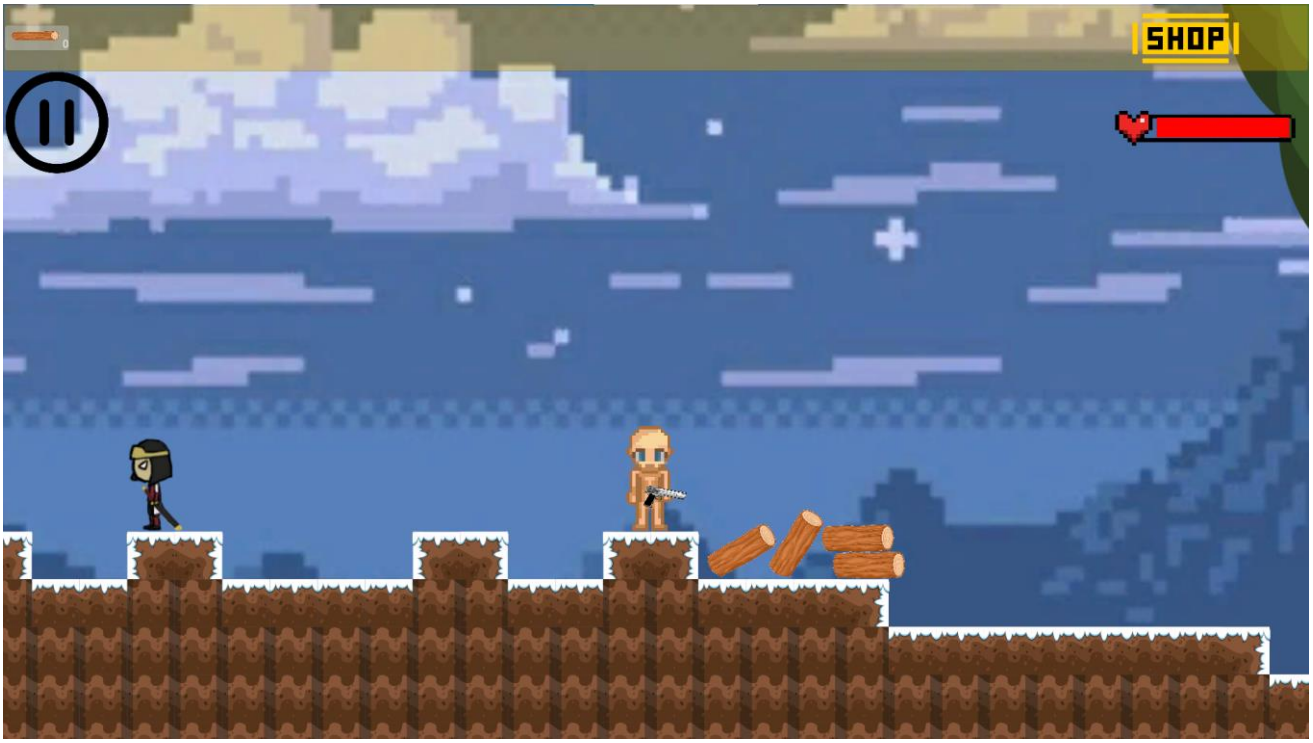


Рисунок 3.15 – Проверка разрушаемости объектов

Проверка работы инвентаря. Рисунок 3.16

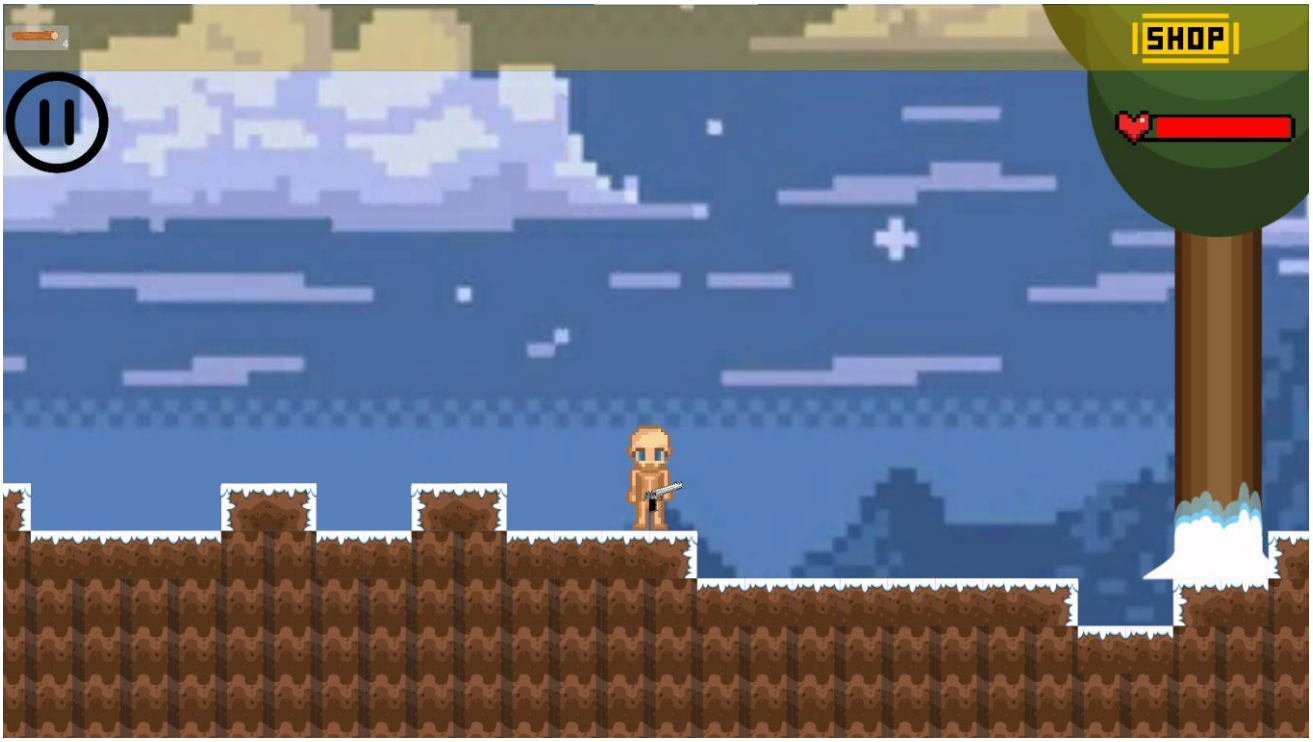


Рисунок 3.16 – Проверка работы инвентаря

Проверка взаимодействия с врагами. Рисунок 3.17

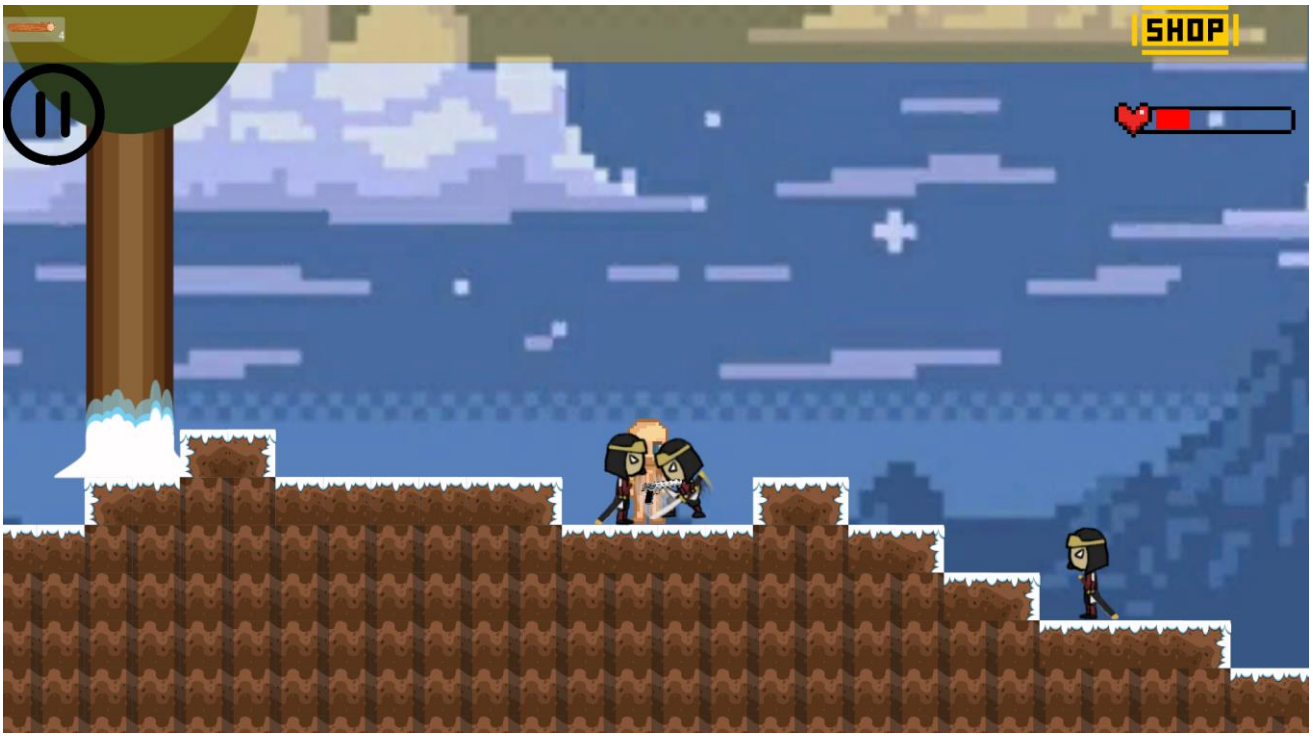


Рисунок 3.16 – Проверка взаимодействия с врагами

Проверка работоспособности магазина. Рисунок 3.17

Рисунок 3.17 – Проверка работоспособности магазина

ЗАКЛЮЧЕНИЕ

В ходе выполнения курсового проекта было написано приложение «2d выживание с элементами шутера на Unity».Приложение написано на движке Unity и языке C# под операционную систему Windows 10.

Мною были освоены навыки проектирования и реализации игр на кроссплатформенном игровом движке Unity, который был использован только для реализации графического интерфейса. Логика, физика и анимация были прописаны с помощью C#.

Также были получены навыки проектирования игрового мира, освоение механик взаимодействия предметов, персонажей, а также реализация анимации при помощи кода, а не за счет функций Unity.

В ходе проведения тестирования, ошибок обнаружено не было. Программа прошла все тесты с положительным результатом.

С помощью разработанного приложения можно не только весело провести время, но и поможет развить следующие качества:

- логическое мышление;
- продумывание тактик;
- решительность;
- ответственность;
- аналитическое мышление.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Unity - Scripting API: GameObject [Электронный ресурс] / - Режим доступа: https://docs.unity3d.com/ScriptReference/GameObject.html – Дата доступа: 20.11.2022 г.
2. Unity - Manual: Tilemap [Электронный ресурс] / - Режим доступа: https://docs.unity3d.com/Manual/class-Tilemap.html – Дата доступа: 20.11.2022 г.
3. Unity - Manual: Audio Mixer [Электронный ресурс] / - Режим доступа: https://docs.unity3d.com/Manual/AudioMixer.html – Дата доступа: 20.11.2022 г.
4. PlayerPrefs в Unity - Сохранение игры, настроек, прогресса [Электронный ресурс] / - Режим доступа: https://www.youtube.com/watch?v=5NXHO5zjWog – Дата доступа: 20.11.2022 г.
5. Как сделать сохранение в Unity через файлы? [Электронный ресурс] / - Режим доступа: https://you-hands.ru/2019/10/25/kak-sdelat-soxranenie-v-unity-cherez-fajly/ – Дата доступа: 20.11.2022 г.
6. Классификация жанров стратегий [Электронный ресурс] / - Режим доступа: https://strategycon.ru/strategy-video-game-classification/ – Дата доступа: 20.11.2022 г.
7. Чем полезны стратегические игры [Электронный ресурс] / - Режим доступа: http://css-zona.ru/news/chem_polezny_strategicheskie_igry/2015-08-06-899 – Дата доступа: 20.11.2022 г.
8. Unity - Scripting API: GameObject [Электронный ресурс] / - Режим доступа: https://docs.unity3d.com/ScriptReference/GameObject.html – Дата доступа: 20.11.2022 г.

ПРИЛОЖЕНИЕ А
(обязательно)
Техническое задание

Введение

Наименование программного продукта – «Life on the island».

Разрабатываемая программа предназначена для мониторинга системных ресурсов и производительности компьютера, для возможности отследить нагруженность определенной части ПК.

А.1 Основание для разработки

Данная программа разрабатывается в рамках курсового проекта студента учреждения образования «Полоцкий государственный университет имени Евфросинии Полоцкой» Кресс Д.А. Основание для разработки является выданное задание к курсовому проекту по теме «2d выживание с элементами шутера на Unity «Life on the island»».

А.2 Назначение разработки

Функциональным и эксплуатационным назначением программы является управление файлами на компьютере.

А.3 Требования к программному продукту

А.3.1 Требования к функциональным характеристикам

При разработке программы «Life on the island» выдвинуты следующие требования:

- возможность отображения графика процессора;
- возможность отображения графика оперативной памяти;
- возможность отображения графика диска;

– возможность отображения двух и более графиков одновременно;
– возможность остановки отображения графиков;
– возможность возобновления отображения графиков в реальном времени;

А.3.2 Требования к надежности

Данная программа должна надежно функционировать. При возникновении аппаратного или программного сбоя программа должна оповещать пользователя о проблеме

А.3.3 Условия эксплуатации

Эксплуатация программы «Life on the island» должна осуществляться на персональном компьютере с Windows 10. Минимальные требования к пользователю – умение обращаться с компьютером, знание основ работы в ОС Windows 10 и выше.

А.3.4 Требование к составу и параметрам технических средств

Для обеспечения устойчивости работы программного средства требуется: x86 или x64 процессор с тактовой частотой от 1 ГГц и выше, 1 ГБ (для 32-разрядного процессора) или 2 ГБ (для 64-разрядного процессора) ОЗУ.

А.3.5 Требования к информационной и программной совместимости

Программное средство должно удовлетворять следующему требованию: ОС Windows 10 и выше.

А.3.6 Требования к маркировке и упаковке

Требования к маркировке и упаковке отсутствуют.

А.3.7 Требования к транспортированию и хранению

Программное средство должно храниться на электронном носителе в виде исполняемого файла

А.4 Требования к программной документации

Программная документация по приложению «Life on the island» должна быть предоставлена в следующем составе:

– техническое задание. Согласно ГОСТ 19.201-78;
– пояснительная записка. Согласно ГОСТ 19.101-77.

Требования к перечисленным программным документам устанавливаются государственными стандартами ЕСПД.

А.5 Стадии и этапы разработки

Разработка программы заключается в следующем:

– анализ исходных данных и постановка задачи проектирования, разработка технического задания;
– разработка интерфейса, архитектуры и структуры программы;
– реализация и тестирование программы;
– разработка программной документации.

А.6 Порядок контроля и приемки

Контроль и приемка программного средства осуществляется в соответствии с программой и методикой испытаний.

Для проверки корректности приложения применялись следующие программные средства:

– ОС Windows 10 x64;
– среда разработки Unity 2D 2019.4.32f1 и JetBrains Rider 2021.2.2.

Тестирование программы состояло из проверки корректности работы ранее перечисленных функций. Методы испытаний: основным методом испытания программы является визуальный контроль выполнения программой требующихся функций, корректное выполнение юнит-тестов.

ПРИЛОЖЕНИЕ Б
(обязательно)
ДИАГРАММА ВАРИАНТОВ ИСПОЛЬЗОВАНИЯ

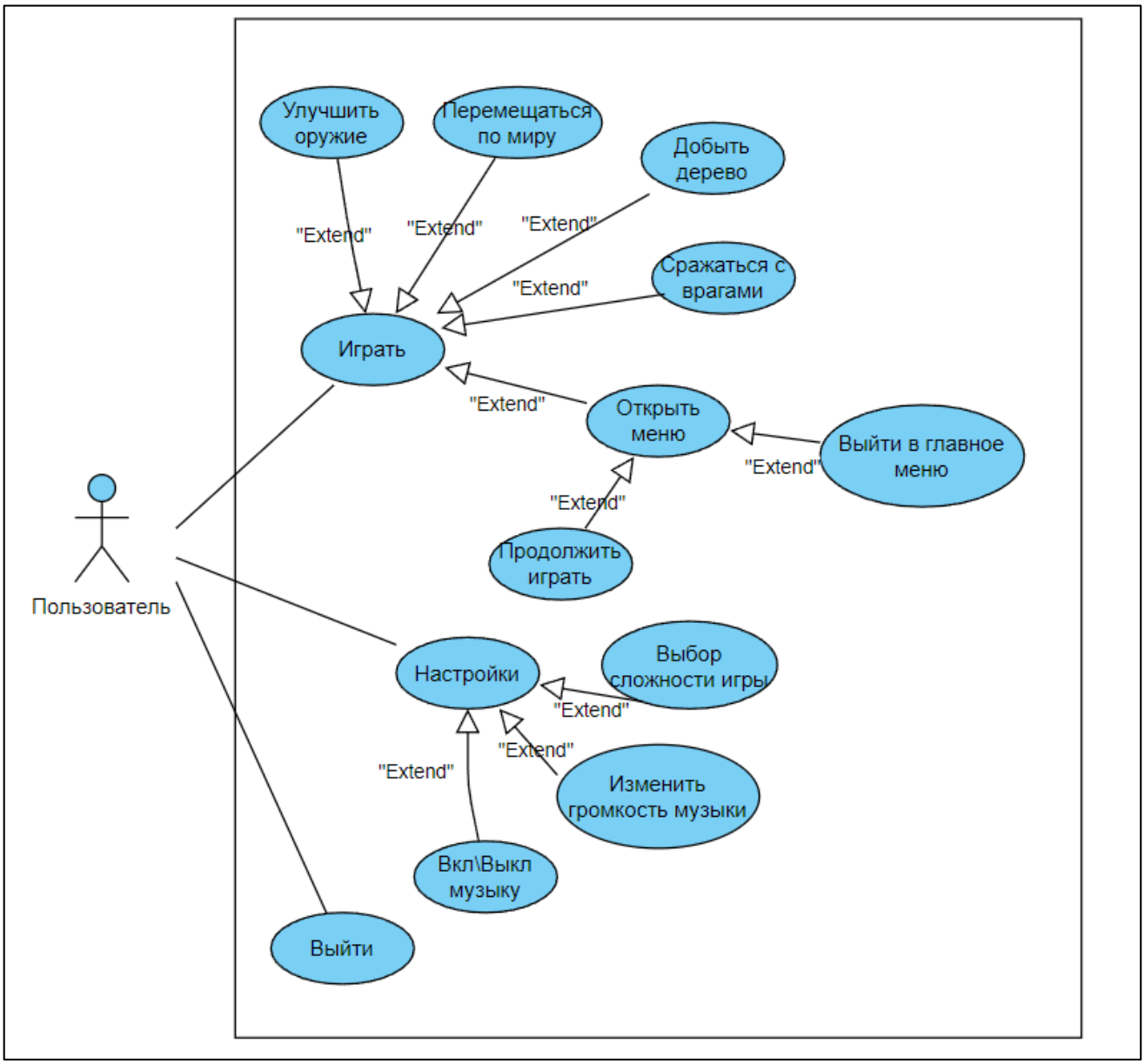


Рисунок Б.1 – Диаграмма вариантов использования

ПРИЛОЖЕНИЕ В
(обязательно)
Таблица тестирование

Таблица В.1

№	Приоритет	Тест	Уровень тестового покрытие	Результат	Ожидаемый результат
1	Наивысший	Запуск приложения	Smoke	Пройден	Откроется окно программы
2	Наивысший	Загрузка игры	Smoke	Пройден	Загрузка игры
3	Наивысший	Проверка механики выстрела	Smoke	Пройден	Выстрел
4	Наивысший	Работоспособность меню опций	Smoke	Пройден	Произойдет изменение громкости музыки, выбор сложности
5	Наивысший	Работа кнопок в магазине	Smoke	Пройден	Покупка оружий
6	Высокий	Проверка оружия при покупке	Smoke	Пройден	Замена основного оружия на купленное
7	Наивысший	Проверка разрушаемости объектов	Smoke	Пройден	Разрушение элементов ланшафта с выпадение дополнительных элементов
8	Высокий	Проверка работы инвентаря	Smoke	Пройден	Подбор дополнительных элементов в сумку с увеличением счетчика предмета

Продолжение таблицы В.1

№	Приоритет	Тест	Уровень тестового покрытие	Результат	Ожидаемый результат
9	Высокий	Проверка взаимодействия с врагами	Smoke	Пройден	Враги преследуют и атакуют персонажа на ближней дистанции
10	Высокий	Проверка работы паузы	Smoke	Пройден	Открытие окна с возможностью продолжить дальше или же выйти в главное меню