

ПЕТРОЗАВОДСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ИНСТИТУТ МАТЕМАТИКИ И ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ
КАФЕДРА ИНФОРМАТИКИ И МАТЕМАТИЧЕСКОГО ОБЕСПЕЧЕНИЯ

Направление подготовки бакалавриата

09.03.04 Программная инженерия

Профиль направления подготовки бакалавриата

“Системное и прикладное программное обеспечение”

Отчет о практике по научно-исследовательской работе:

РАЗРАБОТКА 2D ROGUELIKE ИГРЫ НА ДВИЖКЕ UNITY

Выполнил

студент группы 22207:

И. Е. Мельников _____
подпись

Научный руководитель

к.т.н., доцент кафедры ИМО:

С. А. Марченков _____
подпись

Итоговая оценка

оценка

Содержание

Введение	3
1 Реализация	5
1.1 Создание проекта	5
1.2 Подготовка к созданию игры	6
1.3 Выбор спрайтов	8
1.4 Боевая система	17
1.5 Анимация	19
1.6 Меню	20
1.7 Добавление контента	22
2 Заключение	23

Введение

Мой выбор пал на игру жанра Roguelike. Что это за жанр? Принято считать, что характерными особенностями классического roguelike являются генерируемые случайным образом уровни, пошаговость и необратимость смерти персонажа, то есть в случае его гибели игрок не может загрузить игру и должен начать её заново.

Затем было необходимо определиться с движком. Unity — это бесплатный игровой движок, на котором разрабатывают мобильные игры и проекты для ПК и консолей, а также удобный инструмент для начинающих разработчиков, в котором реально создавать проекты в одиночку.

Unity довольно актуальный инструмент в наше время, например, помимо разработки игр, разработчики самого разного уровня используют движок для создания программного обеспечения с использованием VR технологий, и это не всегда игры. Поэтому в моем проекте создание игры послужит в том числе и как способ ознакомления с этим движком.

На Unity созданы такие популярные проекты как Genshin Impact, Hearthstone, Outlast, Cuphead, Pokemon GO и другие... И не без причины, ведь Unity обладает рядом преимуществ, такими как:

- Доступность. Начать разработку и выпускать свои первые проекты можно бесплатно.
- Низкий порог вхождения в разработку. Некоторые игры получится собрать, даже если вы не умеете писать код.
- Обучение. Для новичков создали подробные бесплатные обучающие материалы.

- Поддержка сообщества. Комьюнити Unity-разработчиков большое, поэтому велика вероятность, что с возникшей проблемой кто-то уже сталкивался и готов помочь.

Рассматривая огромное кол-во других 2D игр, было решено реализовать графику в стиле Pixel-Art, т.к. с ней легко работать, можно использовать почти любой графический редактор.

Цели:

- Приобрести навыки и опыт работы с движком Unity.
- Повысить свою квалификацию по ходу работы с C# и системой Tilemap для Unity.
- Закрепить имеющиеся навыки во время работы с:
 - Языками разметки: LaTeX, Beamer.
 - Языком программирования: C#.
 - Веб-сервисом для хостинга: GitHub.

Задачи:

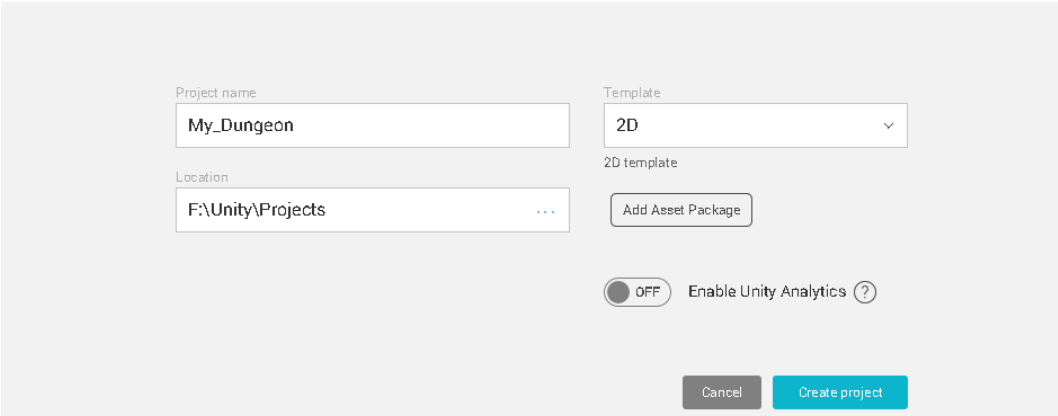
- Освоиться с интерфейсом Unity.
- Создание Roguelike игры.
- Сборка готового билда для скачивания другими пользователями.
- Добавление контента в игру.

1 Реализация

1.1 Создание проекта

Запускаем движок и на странице проектов выбираем «Новый». Задаем необходимые параметры, а именно: название проекта, шаблон графики, расположение на диске. Также есть возможность включить аналитику от Unity, чтобы позволить им собирать данные вашего проекта и предоставлять Вам аналитику схожих проектов.

После загрузки проекта появляется возможность настроить под себя планировку элементов движка и приступить к работе.



The image shows the 'New Project' dialog box in the Unity Hub application. At the top, there are navigation links for 'Projects' and 'Learn', and action buttons for 'New', 'Open', and 'My Account'. The main area contains several input fields: 'Project name' with the text 'My_Dungeon', 'Template' with a dropdown menu showing '2D', and 'Location' with the text 'F:\Unity\Projects'. There is also a '2D template' section with an 'Add Asset Package' button. At the bottom, there is a toggle switch for 'Enable Unity Analytics' which is currently set to 'OFF'. Finally, there are two buttons at the bottom right: 'Cancel' and 'Create project'.

Рис. 1 – Создание проекта

1.2 Подготовка к созданию игры

Начать стоит с необходимых разделов:

- В окне проекта создаем папку Artwork. И внутри нее создаем папки Animations и Artwork. Здесь будет находиться все необходимое, что касается аниматоров, анимаций, Tile'ы для Tilemap'ов наших уровней, атласы и шрифты.
- Папка Prefab будет хранить готовые игровые объекты, которые надо будет использовать больше одного раза, чтобы не создавать их заново.
- Папка Scenes будет хранить уровни игры, или же сцены, со всеми размещенными на ней объектами.
- Папка Scripts будет хранить написанные на C# скрипты для игровых объектов.
- После этого создаем главную сцену, которая послужит Intro-уровнем игры.

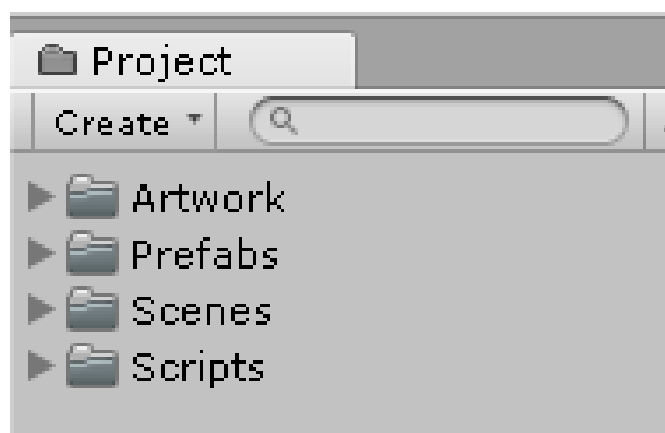


Рис. 2 – Разделы

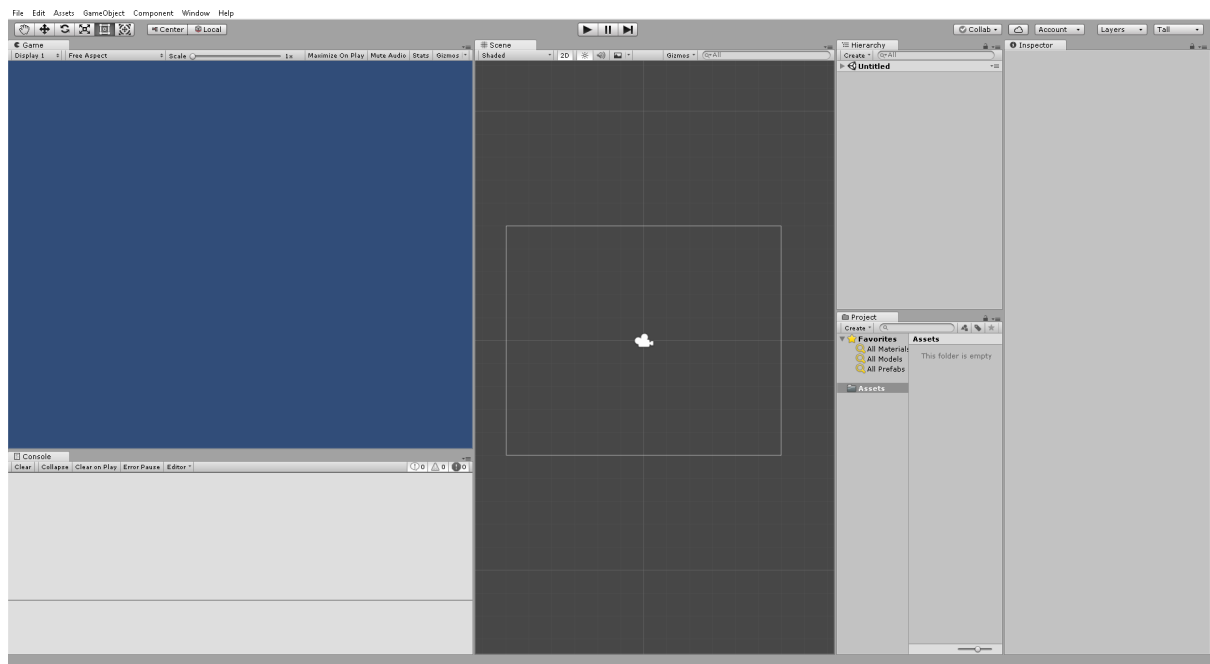


Рис. 3 – Планировка движка. Главная сцена

1.3 Выбор спрайтов

Ввиду своих слабых творческих способностей было принято решение использовать готовый сет спрайтов, бесплатный, в том числе и для коммерческих проектов.

Сет представляет из себя набор около 300 различных спрайтов различных моделей оружия, персонажей, монстров, стен, сундуков и всего другого необходимого для жанра Roguelike.

После загрузки атласа вырезаем из него первые спрайты, а именно, игровых персонажей. Что удобно в данном наборе спрайтов, так это то, что большинство необходимых спрайтов уместается в размеры 16 на 16 пикселей, что позволяет иметь некий стандартизированный их размер.

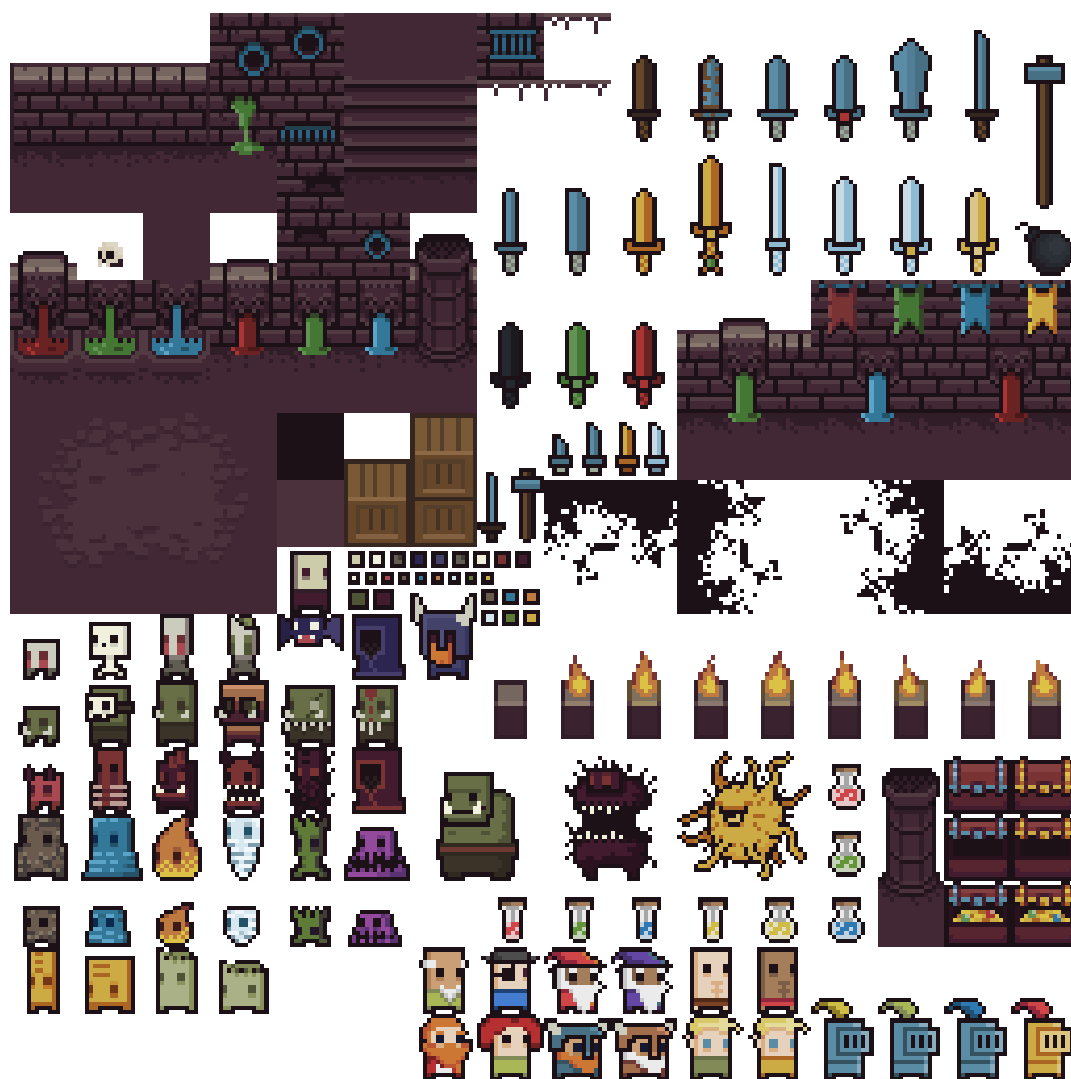


Рис. 4 – Сет спрайтов

Написание кода

После добавления первых объектов можно приступить к написанию первых скриптов.

Начнем с добавления компонента главному герою, а именно скрипта Player.cs.

```
1  using System.Collections;
2  using System.Collections.Generic;
3  using UnityEngine;
4
5  public class Player : Mover
6  {
7      private SpriteRenderer spriteRenderer;
8      private bool isAlive = true;
9
10     protected override void Start()...
15     protected override void ReceiveDamage(Damage dmg) ...
23     protected override void Death()...
28
29     private void FixedUpdate()...
37
38     public void SwapSprite(int skinId) ...
42     public void OnLevelUp()...
47     public void SetLevel(int level) ...
52     public void Heal(int healingAmount) ...
63     public void Respawn()...
70 }
```

Рис. 5 – Player.cs

По началу в Player.cs будет находится все, что связано с движением объекта. Обработка координат и векторов будет происходить в функции FixedUpdate. Также создается Box Collider. Он отвечает за область объекта которая будет сталкиваться с другими, тоже имеющими такую область.

Создадим скрипт CameraMotor.cs для движения камеры и реализуем в нем функцию LateUpdate. Скрипт будет указывать камере, какой объект необходимо отслеживать, а также границы, в которых этот объект

будет отслеживаться.

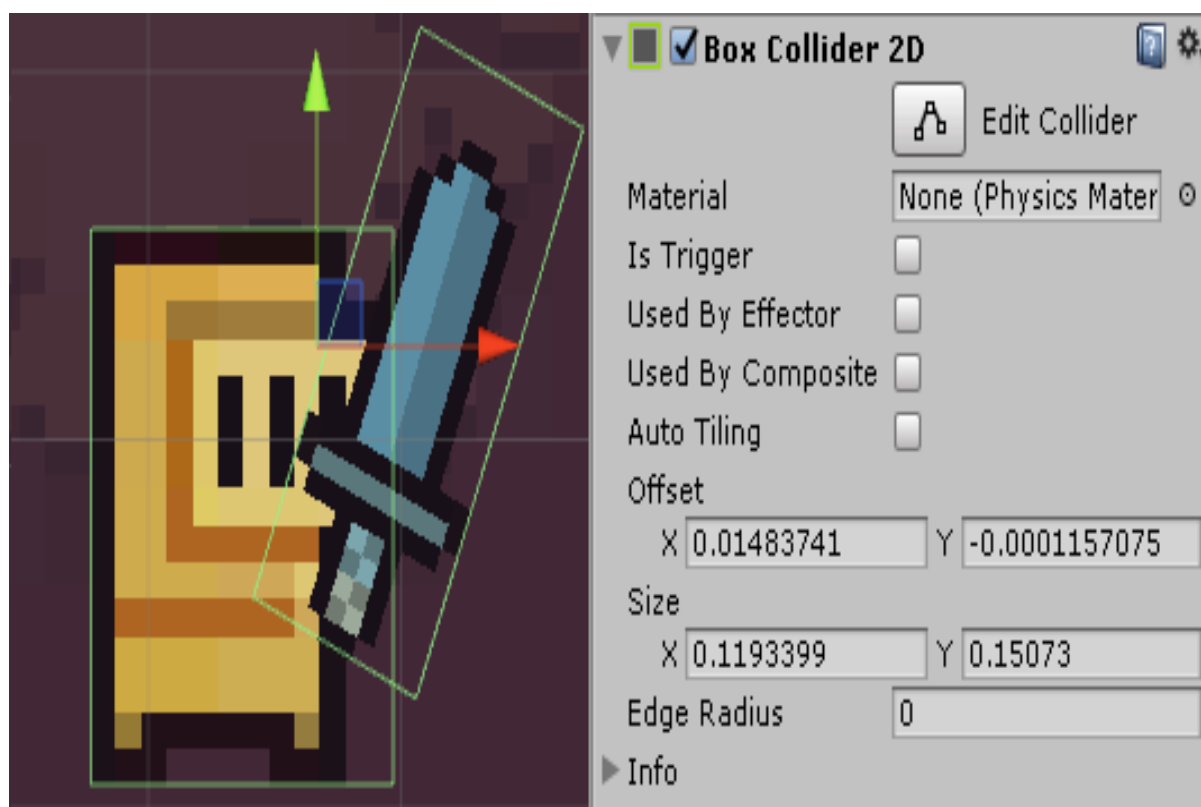


Рис. 6 – Box Collider

После этого можно отвлечься и заняться прорисовкой уровня, чтобы персонаж не ходил по пустоте. Для этого переносим все необходимые спрайты в палитру, и заполняем ими созданные Tilemap'ы. Слоев должно быть несколько, чтобы можно было отрисовывать более детальные объекты с помощью накладывания одного на другой. Одним из слоев будет слой, блокирующий игроку доступ в определенные области. Для этого добавляем Tilemap Collider, отвечающий за столкновение и снимаем галочку с рендера данного слоя в игре.

Сразу создаем новую сцену - следующий уровень для игрока, и отрисовываем и ее.

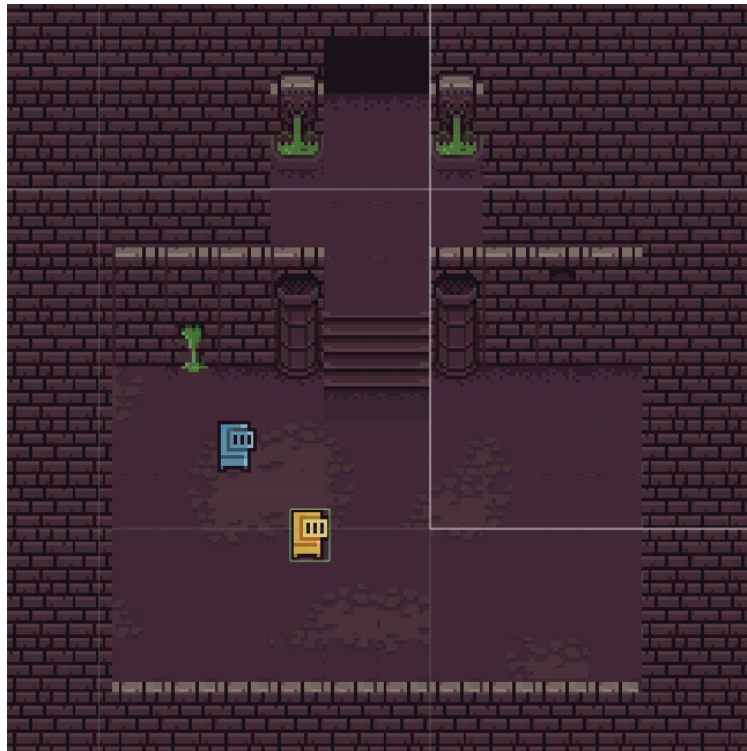


Рис. 7 – Level 1. Main

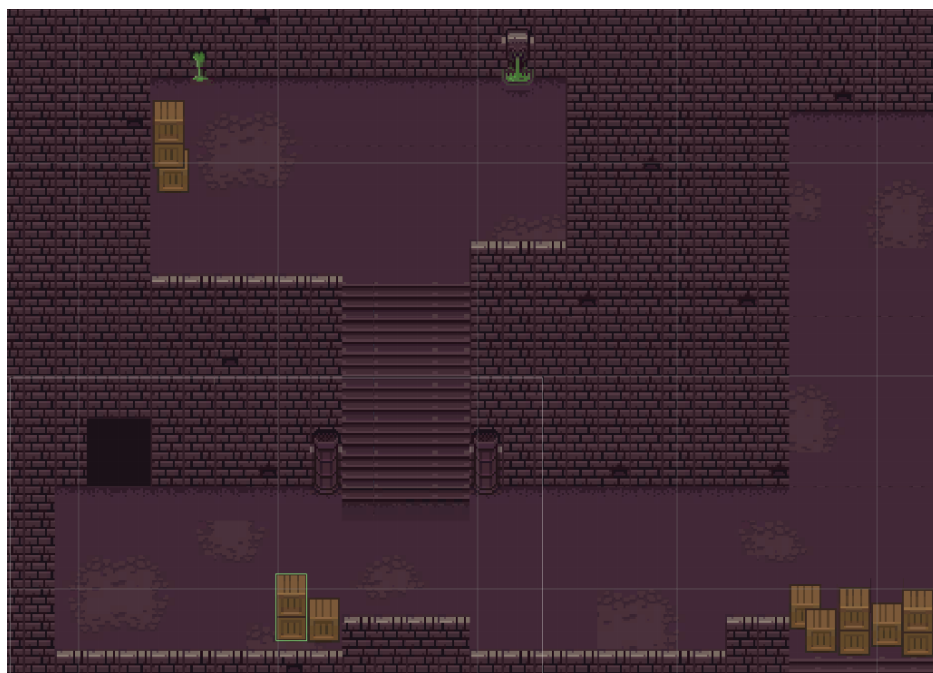


Рис. 8 – Level 2. Dungeon

Пропишем скрипт для объектов, с которыми игрок сможет взаимодействовать на отрисованной карте. В Collidable.cs реализована функция Update, которая будет отслеживать с какими именно объектами в данный момент «сталкивается» игрок.

Сундуки должны награждать игрока золотом. Но кроме него в игре могут присутствовать другие объекты для собирания, для этого создадим класс Collectable.cs, который будет наследовать класс Collidable.cs, и пропишем в нем логику собирания.

```
1  using System.Collections;
2  using System.Collections.Generic;
3  using UnityEngine;
4
5  public class Collidable : MonoBehaviour
6  {
7      public ContactFilter2D filter;
8      private BoxCollider2D boxCollider;
9      private Collider2D[] hits = new Collider2D[10];
10
11      protected virtual void Start()...
15
16      protected virtual void Update()...
31
32      protected virtual void OnCollide(Collider2D coll)...
```

Рис. 9 – Collidable.cs

```
1  using System.Collections;
2  using System.Collections.Generic;
3  using UnityEngine;
4
5  public class Collectable : Collidable
6  {
7      // Logic
8      protected bool collected;
9
10     protected override void OnCollide(Collider2D coll)...
```

Рис. 10 – Collectable.cs

Теперь для сундуков напишем отдельный скрипт Chest.cs, который будет наследовать класс Collectable и пропишем в нем смену состояние сундука и его спрайта.

Для портала переноса на следующий уровень создадим следующий скрипт Portal.cs. Он наследует класс Collidable, проверяет что в портале находится игрок, и только тогда загружает следующую сцену.

```
1 using System.Collections;
2 using System.Collections.Generic;
3 using UnityEngine;
4
5 public class Chest : Collectable
6 {
7     public Sprite emptyChest;
8     public int goldAmount = 5;
9
10    protected override void OnCollect()
11    {
12        if(!collected)
13        {
14            collected = true;
15            GetComponent<SpriteRenderer>().sprite = emptyChest;
16            GameManager.instance.gold += goldAmount;
17            GameManager.instance.ShowText("+" + goldAmount + " g
18        }
19    }
20 }
21
```

Рис. 11 – Chest.cs

```
1 using UnityEngine;
2
3 public class Portal : Collidable
4 {
5     public string[] sceneNames;
6
7     protected override void OnCollide(Collider2D coll)
8     {
9         if(coll.name == "Player")
10        {
11            // Teleport the player
12            GameManager.instance.SaveState();
13            string sceneName = sceneNames[Random.Range(0, sceneNames.Length)];
14            UnityEngine.SceneManagement.SceneManager.LoadScene(sceneName);
15        }
16    }
17 }
18
```

Рис. 12 – Portal.cs

После этого создадим GameManager в котором будет находиться боль-

шинство варьирующихся переменных и ресурсов, таких как спрайты героев, оружия, цены на оружие, опыт, кол-во золота, кол-во жизней и др. А также референсы на скрипт для игрока, скрипт для оружия и другие будущие скрипты.

Также пропишем в нем состояние сохранения и состояние загрузки, где в и из string переменной будут грузиться некоторые данные об игроке (золото, опыт, уровень оружия).

```
6 public class GameManager : MonoBehaviour
7 {
8     public static GameManager instance;
9     private void Awake()...
10
11     // Resources
12     public List<Sprite> playerSprites;
13     public List<Sprite> weaponSprites;
14     public List<int> weaponPrices;
15     public List<int> xpTable;
16
17     // References
18     public Player player;
19     public Weapon weapon;
20     public FloatingTextManager floatingTextManager;
21     public RectTransform hitpointBar;
22     public Animator deathMenuAnim;
23     public GameObject hud;
24     public GameObject menu;
25
26     // Logic
27     public int gold;
28     public int experience;
29
30     // Floating text
31     public void ShowText(string msg, int fontSize, Color color, Vector3 position,...
32
33     // Upgrade weapon
34     public bool TryUpgradeWeapon()...
35
36     // Health bar
37     public void OnHitpointChange()...
38
39     // exp system
40     public int GetCurrentLevel()...
41     public int GetXpToLevel(int level)....
42     public void GrantXp(int xp)....
43     public void OnLevelUp()...
44
45     // on scene loaded
46     public void OnSceneLoaded(Scene s, LoadSceneMode mode)....
47
48     // Respawn
49     public void Respawn()...
50
51     // Save state
52     /* ...
53
54     public void SaveState()...
55
56     public void LoadState(Scene s, LoadSceneMode mode)....
57 }
```

Рис. 13 – GameManager.cs

Затем пропишем скрипт FloatingText.cs для всего всплывающего текста. В нем будут функции Show и Hide, а также функция отрисовки сообщения в кадре в зависимости от времени.

Необходим также менеджер всплывающих сообщений FloatingTextManager.cs, чтобы реиспользовать один и тот же объект для отображения различных сообщений, каждый раз заменяя размер шрифта, цвет и т.д. В Show

пропишем также длительность и положение сообщения относительно камеры.

В GameManager'е также пропишем функцию для вызова менеджера текста.

После этого можно начать вызывать функцию в Chest.cs для вызова сообщения о награждении золотом.

```
1  using System.Collections;
2  using System.Collections.Generic;
3  using UnityEngine;
4  using UnityEngine.UI;
5
6  public class FloatingTextManager : MonoBehaviour
7  {
8      public GameObject textContainer;
9      public GameObject textPrefab;
10
11     private List<FloatingText> floatingTexts = new List<FloatingText>();
12
13     private void Update()...
14
15     public void Show(string msg, int fontSize, Color color, Vector3 posit
16
17     private FloatingText GetFloatingText()...
```

Рис. 14 – FloatingTextManager.cs

1.4 Боевая система

Перейдем к боевой системе.

Переносим оружие на сцену и добавляем ему Box Collider'а и создаем скрипт Weapon.cs. В нем будут храниться массивы с уроном и отдачей по противнику, переменные уровня оружия, кулдауна удара и другие переменные для функционирования боевой системы. Скрипт будет следить за оружием в реальном времени: его кулдауном, хитбоксом и уровнем.

В скрипте также будет находится вызов структуры для нанесения урона, и это новый скрипт Damage.cs. Он будет служить контейнером для урона по объекту, на который мы будем его отправлять.

Теперь создадим скрипт Fighter.cs, который будет наследоваться нашим игроком и врагами. Он будет наделять их здоровьем и функцией для получения урона.

В связи с этим создадим скрипт Enemy.cs, наследующий класс Fighter, содержащий все данные о враге (здоровье, опыт, хитбокс удара и урона, состояние преследования и другое).

Чтобы вооружить врага создадим скрипт EnemyHitbox, наследующий класс Collidable, где будем отправлять урон по игроку, если объекты сталкиваются.

Самое время создать скрипт движения Mover.cs, наследующий класс Fighter. Сюда перейдет большая часть скрипта Player.cs, т.к. теперь Mover будет отвечать за передвижение объекта, а Player.cs в свою очередь теперь будет наследовать класс Mover. Скрипт будет работать с векторами, следить за направлением движения, толчков и заставлять объект двигаться.

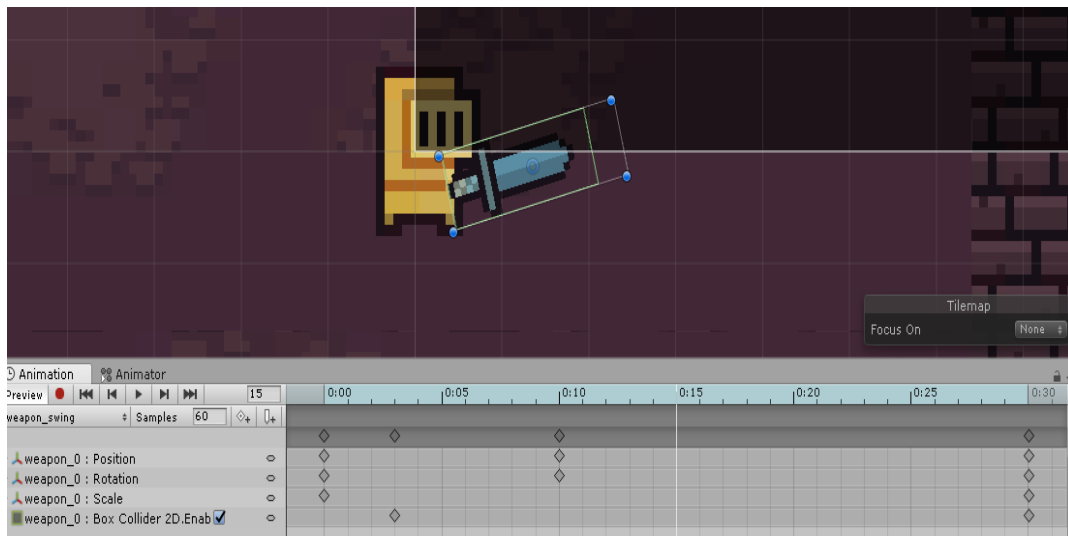


Рис. 15 – Анимация

```

1  using System.Collections;
2      using System.Collections.Generic;
3      using UnityEngine;
4
5  public abstract class Mover : Fighter
6  {
7      private Vector3 originalSize;
8
9      protected BoxCollider2D boxCollider;
10     protected Vector3 moveDelta;
11     protected RaycastHit2D hit;
12
13     public float ySpeed = 0.75f;
14     public float xSpeed = 1.0f;
15
16     protected virtual void Start()...
21
22     protected virtual void UpdateMotor(Vector3 input)...
54

```

Рис. 16 – Mover.cs

1.5 Анимация

Создание анимации в Unity происходит довольно просто.

Для начала нужно добавить объекту компонент Animator. После чего в окне анимации, создаем клип и переходим непосредственно к анимации.

На концах временной шкалы задаем состояние нашего объекта, и Animator самостоятельно заполнит пространство. Единственное с чем придется повозиться, это с хитбоксом, который должен активироваться и деактивироваться на определенной секунде.

Далее в аниматоре соединяем состояния согласно логике анимации и привязываем переходы к триггерам. И все, что остается, это добавить триггеры в код. Для этого в функции удара выполняем вызов необходимого триггера.

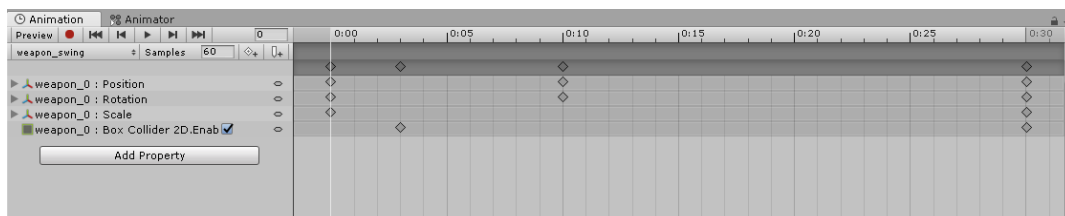


Рис. 17 – Раздел анимации

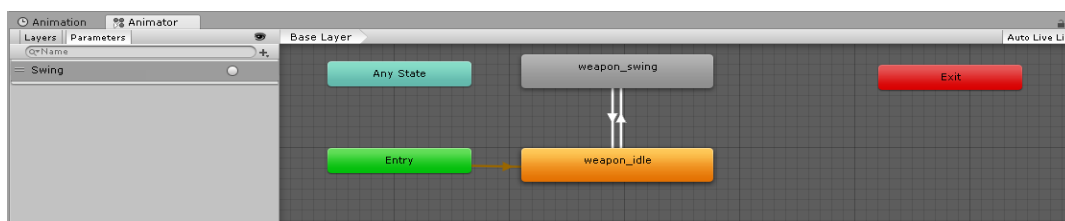


Рис. 18 – Раздел аниматора

1.6 Меню

Для вызова меню понадобится канвас с самим меню



Рис. 19 – Меню

и канвас интерфейса с кнопкой.



Рис. 20 – Интерфейс

Создаем анимацию появления меню по уже известному принципу. Триггерами анимации послужат кнопки в интерфейсе, для которых в Unity уже написан готовый скрипт. Активацией послужит иконка сундука, а деактивацией задний фон меню.

За активные кнопки в меню будет отвечать скрипт CharcterMenu.cs.

Так за смену спрайта игрока отвечает функция `OnSelectionChanged`, за апгрейд оружия `OnUpgradeClick`, а за состояние самого меню в кадре (золото,опыт,здоровье) функция `UpdateMenu`.

1.7 Добавление контента

Теперь, когда основная часть проекта готова, можно заняться добавлением разного рода контента в игру. В папке Prefab уже имеется пара интерактивных объектов, сундуки, анимированные факелы, образец врага, и с этим уже можно работать и создавать уровни наподобие того, что сделал я.



Рис. 21 – Пример уровня

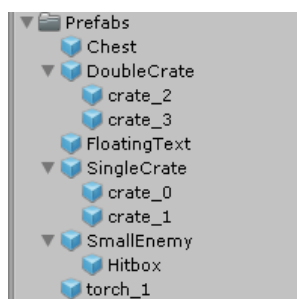


Рис. 22 – Prefab

Но это не значит, что этим ограничиваются игровые механики и ситуации, которые можно реализовать в данном проекте.

2 Заключение

В данной курсовой работе было произведено ознакомление с игровым движком Unity, в результате чего была создана 2D игра в жанре Roguelike.

Также был собран готовый билд для ознакомления с игрой другими пользователями. В игру был добавлен один готовый игровой уровень.

В ходе работы был получен опыт работы с самим движком и системой Tilemap для Unity, а также были закреплены полученные навыки работы с GitHub, Latex, Beamer.

Список литературы

1. Основная документация <https://docs.unity3d.com/2021.3/Documentation/Manual>
2. Документация <https://learn.unity.com/tutorial/introduction-to-tilemaps>
3. Руководство <https://ru.wikipedia.org/wiki/Roguelike>
4. Microsoft C# URL: <https://docs.microsoft.com/ru-ru/dotnet/csharp/tour-of-csharp/tutorials/>
5. Visual Studio Code — URL: <https://code.visualstudio.com/>