

МИНОБРНАУКИ РОССИИ
Федеральное государственное бюджетное образовательное учреждение
высшего образования
**«САРАТОВСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ
ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ИМЕНИ Н. Г. ЧЕРНЫШЕВСКОГО»**

Кафедра математической кибернетики и компьютерных наук

**РАЗРАБОТКА БАЗОВОГО ФУНКЦИОНАЛА ДЛЯ ОБУЧАЮЩЕЙ
КОМПЬЮТЕРНОЙ РОЛЕВОЙ ИГРЫ**

КУРСОВАЯ РАБОТА

студентки 2 курса 211 группы
направления 02.03.02 — Фундаментальная информатика и информационные
технологии
факультета компьютерных наук и информационных технологий
Никитенко Яны Валерьевны

Научный руководитель
доцент, к.т.н.

М. В. Хамутова

Заведующий кафедрой
доцент, к. ф.-м. н.

С. В. Миронов

СОДЕРЖАНИЕ

ВВЕДЕНИЕ	3
1 Анализ предметной области разработки компьютерных игр	4
1.1 Постановка задачи разработки игры (Актуальность, цель, назначение и описание целевой аудитории)	4
1.2 Исследование понятия и классификация компьютерных игр	4
1.3 Обоснование и обзор выбранного жанра игры	5
1.4 Описание геймплея разрабатываемой игры	5
1.5 Обоснование и выбор инструментов создания игры (анализ игровых движков, графических редакторов и других инструментов разработки игры)	6
2 Описание разработки игры	9
2.1 Общий алгоритм функционирования	11
2.2 Создание игровых классов и объектов.....	15
2.3 Создание 2D модели персонажа	20
2.4 Создание уровней	21
2.5 Создание интерфейсов	24
2.6 Тестирование игры	26
ЗАКЛЮЧЕНИЕ	29
СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ	30
Приложение А Полный код для работы инвентаря.....	31
Приложение Б Flash-носитель с отчетом о выполненной работе	36

ВВЕДЕНИЕ

Целью курсового проекта является разработка базового функционала для компьютерной игры используя Unity. Реализация таких задач как:

- Передвижения персонажа
- Инвентарь(добавление предметов, проверка на заполненность инвентаря)
- Система цепочки заданий
- Боевая Система
- Диалоговая система

Проект является игровым приложением в жанре RPG(анг.Role-playing game). Основу игрового процесса составляет выполнения различных заданий(в первую очередь главного сюжетного задания),развития навыков и характеристик в ходе сражений и использования навыков. При желании разработчик может добавлять другие дополнительные механики(к примеру ремесло), которые помогут выделить игру на фоне конкурентов.

В данной курсовой работе будет разработаны базовые функции для игры, которые выше перечислены были.

1 Анализ предметной области разработки компьютерных игр

В самом общем смысле, компьютерную игру можно определить как интерактивную программную систему, в которой один или несколько игроков, воздействуя через устройства ввода (клавиатура, мышь, джойстик, сенсорный экран и т.д.), управляют виртуальными объектами (персонажами, процессами, средами) в рамках заданных правил с целью достижения определенных результатов, часто связанных с вызовом, развлечением, обучением или социальным взаимодействием.

1.1 Постановка задачи разработки игры (Актуальность, цель, назначение и описание целевой аудитории)

Славянская мифология, является богатейшим пластом культурного наследия Восточной Европы, остается значительно менее представленной в массовой игровой индустрии по сравнению с скандинавской, греческой или кельтской мифологиями. Существует растущий спрос со стороны аудитории на аутентичные и глубокие погружения в это самобытное мифологическое пространство.

Несмотря на наличие некоторых игр, затрагивающих славянскую тематику, ощущается дефицит полноценных РПГ, целостно и уважительно раскрывающих мир славянских духов, божеств, обрядов и эпических сказаний. Большинство проектов либо поверхностны, либо используют мифологию лишь как декоративный фон.

В этой игре в центре сюжета оказываются именно мифические создания, их проблемы, взгляд на мир людей и непосредственно конфликт с ним.

Собственно цель проекта популяризировать славянский фольклор и культуру, в особенности среди молодого поколения. Т.е основной целевой аудиторией являются люди от 14 до 30 лет.

1.2 Исследование понятия и классификация компьютерных игр

Из-за огромного разнообразия игр, их целей, механик и платформ, создание единой, всеобъемлющей классификации является сложной задачей. Классификации часто пересекаются и дополняют друг друга. Как правило классифицировать игры можно по:

1. Жанру
2. Платформе

3. Количество игроков и режиму взаимодействия
4. Модели монетизации
5. Целевому назначению и содержанию

1.3 Обоснование и обзор выбранного жанра игры

Компьютерная ролевая игра (англ. computer role-playing game, обозначается аббревиатурой CRPG или RPG) — жанр компьютерных игр, в котором 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. [1]

Данный жанр был выбран по следующим причинам:

- Личная мотивация - автор давно знаком с этим жанром, хорошо знаком с механиками и искренне любит его
- Зачастую игры вдохновленные мифами выпускаются в этом жанре

1.4 Описание геймплея разрабатываемой игры

Основой геймплея является: выполнение заданий(убийства монстров, сбор предметов, защита и сопровождение NPC, разведка, разговор с NPC(Non-player character) и т.д), развития персонажа (повышение характеристик, улучшение и получение новых навыков), исследование мира, занесение в кодекс информации о мире.

В этом проекте, будет реализовано пока только выполнения заданий со сбором и убийством монстров. А так же кодекс, в прологе будет одна запись о существе "Жар-птица"на начало игры, что-бы игрок больше узнал за кого он будет играть. Далее заполнение кодекса должно будет стать частью игрового процесса. Заполнение кодекса будет происходить, во время путешествий по миру игры.

В дальнейшем в развитии проекта все элементы будут получать развитие.

В данной курсовой работе будет реализован только пролог с базовыми и простыми механиками, которые познакомят с тем, что его будет ждать в дальнейшем.

В основном информация для кодекса берется из книги "Славянские мифы от Велеса и Мокоши до Птицы Сирин и Ивана Купалы"автора А. Л Барков. [2]

1.5 Обоснование и выбор инструментов создания игры (анализ игровых движков, графических редакторов и других инструментов разработки игры)

На данный момент времени существует множество инструментариев для облегчения процесса создания видеоигр. Одним из таких являются движки, редакторы кода и другие.

На данный момент все ныне существующие игровые движки используют параллельные вычисления, однако не во всех аспектах – так, например многопоточной обработке подвергаются рендеринг изображения, обработка физических вычислений, обработка звуковых дорожек. Несмотря на это, логика игры, ее правила и механики обрабатываются в одном потоке одного ядра ЦП, что, при большом масштабе проекта выливается в так называемые “физзы” – моменты, когда в очереди на выполнение оказывается множество команд и время обработки значительно вырастает.

Решить данную проблему призван архитектурный паттерн Entity – Component – System (ECS), который заменяет традиционную для разработки игр парадигму объектно-ориентированного программирования на параллельное программирование и работу с данными.

Был проведен анализ следующих игровых движков.

— Unity

Unity является одним из популярных движков, использующий язык программирования C#. Данный продукт предоставляет множество платных и бесплатных асетов, которые помогут разработчикам в создании игры. Также существует пробная версия дополнительных программных пакетов D.O.T.S., позволяющая применять параллельные вычисления ко всем игровым составляющим — логика, физика, видео- и звуковые потоки.

В Unity используется среда .NET и язык программирования C# — самый популярный в разработке игр. Движок компилирует код C# для каждого целевого устройства, поэтому вы можете развертывать приложения для ПК, мобильных устройств, консолей, а также платформ AR и VR. [3]

— Unreal Engine 4

Unreal Engine универсальный и очень гибкий движок, использующий язык программирования C++. Он наполнен множеством бесплатных

инструментов разработки.

В версии 4 компания Epic Games сделала упор на простоту пользования. В соответствии с фокусом UE4 на простоте, он включает новую визуальную систему сценариев под названием «Blueprints» (преемник «Kismet» UE3), которая позволяет быстро разрабатывать игровую логику без использования кода, что приводит к уменьшению разрыва между техническими художниками, дизайнерами и программистами. [4]

Его преимуществом является поддержка UltraHD разрешения графики и прямая работа с памятью с помощью встроенных библиотек. Параллельные вычисления можно осуществлять с помощью встроенного в C++ планировщика задач, однако его невозможно применить к чему-либо, кроме логической части игры и физическим вычислениям.

— CryEngine

CryEngine на данный момент представляет разработчикам возможность делать в своих проектах фотoreалистичную графику. [5] Данный движок предоставляет встроенную поддержку мультипоточных вычислений, однако, так же, как и в Unreal Engine, они применимы лишь на логическую и физическую части игрового проекта.

Движок использует язык программирование C++.

— Godot

Godot развивающийся движок, использующий собственный язык программирования - GDScript. Присутствует поддержка C++ и C#, в скором времени и Python. Идеально подойдет для знакомства с разработкой игр, если у пользователя уже есть небольшой опыт в программировании. Также имеет встроенный функционал распараллеливания процессов с помощью планировщика задач.

Узлы организованы внутри «сцен», которые являются повторно используемыми, инстанцируемыми, наследуемыми и вложенными группами узлов. Узлы соединены сигналами, которые могут передавать объекты данных. Все игровые ресурсы, включая скрипты и графические активы, сохраняются как часть файловой системы компьютера (а не в базе данных). Это решение для хранения предназначено для облегчения совместной работы между командами разработчиков игр, использующими системы контроля версий программного

обеспечения. [6]

— GameMaker Studio

GameMaker Studio — один из самых известных игровых движков наравне с Unity и Unreal Engine. На нем сделаны многие инди-хиты вроде Undertale, а еще у него очень низкий порог вхождения: для GameMaker необязательно умение программировать, и все взаимодействия можно настраивать буквально «перетягиванием».

GameMaker позволяет создавать кроссплатформенные и многожанровые видеоигры с использованием пользовательского визуального языка программирования с функцией перетаскивания или языка сценариев , известного как Game Maker Language (GML), который можно использовать для разработки более сложных игр. GameMaker изначально был разработан, чтобы позволить начинающим программистам создавать компьютерные игры без особых знаний в области программирования, используя эти действия. Последние версии программного обеспечения также ориентированы на продвинутых разработчиков. [7]

2 Описание разработки игры

Для разработки игры используются: функционал движка Unity, версия 6000.0.32f1, язык программирования C#, редактор кода Microsoft Visual Studio Community 2022 (64-разрядная версия) - Current версия 17.4.4, растровый графический редактор Krita версия 5.2.2 и редактор изображений Aceprite версия v1.3.14.2.

В самом начале проекта был нарисован спрайт персонажа, которым будет управлять игрок и портрет персонажа в меню. Ниже представлены этапы рисования портрета в Krita(см. рисунки 1 и 2), а также готовый первый спрайт(см. рисунок 3).

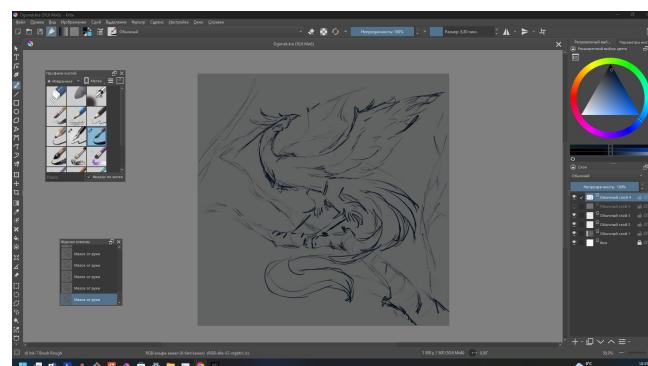


Рисунок 1 – Самый первый набросок портрета

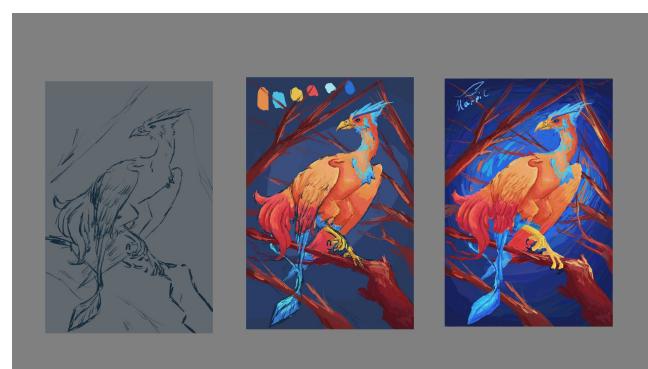


Рисунок 2 – Этапы рисования портрета

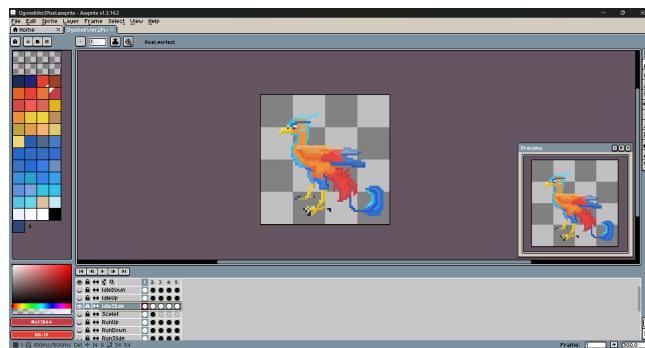


Рисунок 3 – Самый первый спрайт

После было создано небольшое окружение для тестирования механик.

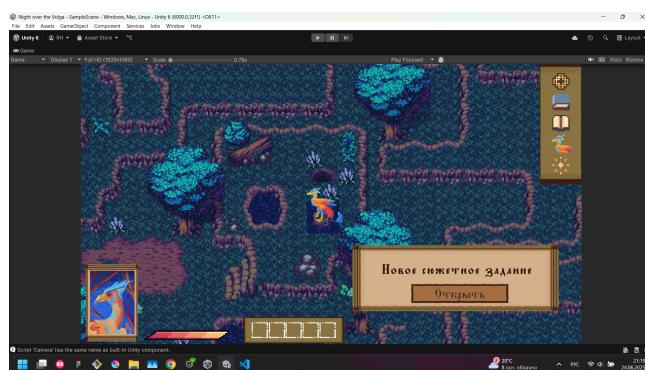


Рисунок 4 – Тестовый вариант локации

Как только было реализовано управление по нажатию кнопок, были добавлены анимации для передвижения.

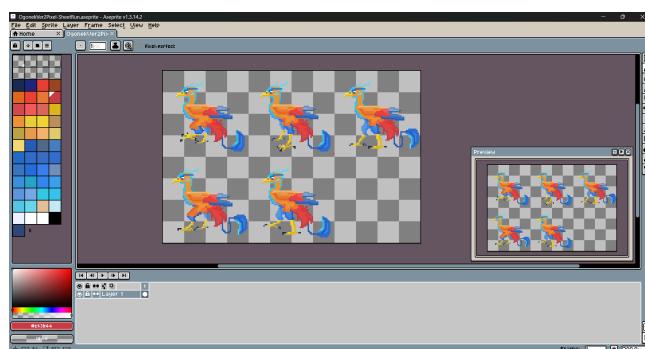


Рисунок 5 – Кадры анимации бега

Далее был создан интерфейс и реализован его функционал.
На карту были добавлены предметы для взаимодействия.
Последними этапами стали: добавления задания, врагов и выход из локации.

Более подробно каждый аспект будет рассмотрен в каждом из разделов.

2.1 Общий алгоритм функционирования

При запуске игры первым, что видит пользователь это главное меню где есть кнопки "Играть" и "Выход"(см. рисунок 6). При нажатии на первую кнопку произойдет загрузка первой сцены - картинка с предысторией и кнопкой "Продолжить"(см. рисунок 7), которая в свою очередь уже загрузит первую локацию. А на вторую выход из игры соответственно.

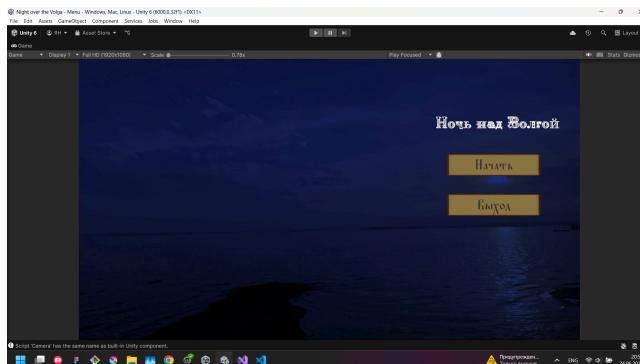


Рисунок 6 – Первоначальное меню с альтернативным названием

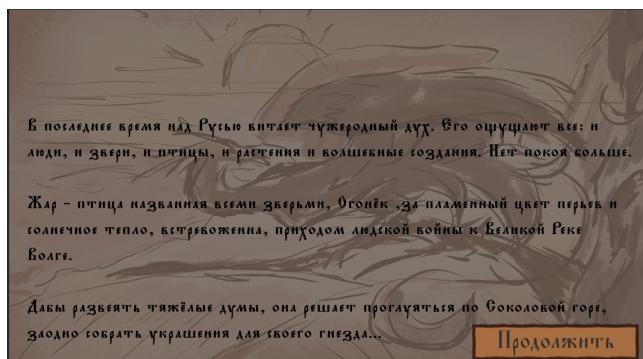


Рисунок 7 – Текст с пояснением

Загрузка локаций происходит с помощью очереди сцен, которая задается самим разработчиком(см. рисунок 8).

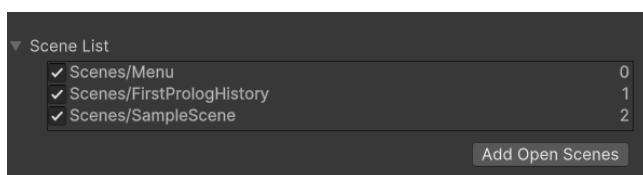


Рисунок 8 – Очередь сцен

Очередь сцен представлена в самом Unity, но загрузка осуществляется с помощью кода. Ниже представлен код из файла "MainMeun.cs"

```

using UnityEngine;
using UnityEngine.SceneManagement;
public class MainMeun : MonoBehaviour

{
    public void PlayGame()
    {
        SceneManager.LoadScene(SceneManager.GetActiveScene().buildIndex+1);
    }

    public void ExitGame()
    {
        Debug.Log("Exit");
        Application.Quit();
    }
}

```

После двух окон перед игроком предстает первая локация с главным героем, вернее героиней - жар-птицей, зовут которую Огонёк, о которой можно было немного узнать из небольшого пояснения в начале и окно с предложением, принять основное задание из длинной цепочки заданий (в данной курсовой реализована лишь самая малая её часть), завершив которую игра будет пройдена.



Рисунок 9 – Начало игры

После нажатия кнопки "Принять"(см. рисунок 10), будет открыто окно с описанием задания, выполнив требования задания, его названием и наградой(в этом задание игрок получит 100 единиц опыта, которого по факту нет, т.к в

рамках этой курсовой работы отсутствует развитие и персонажа и получение навыков) игрок сможет перейти на вторую локацию, последнюю в прологе. После принятия, на персонажа будет повешен маркер активного задания(см. рисунок 11).

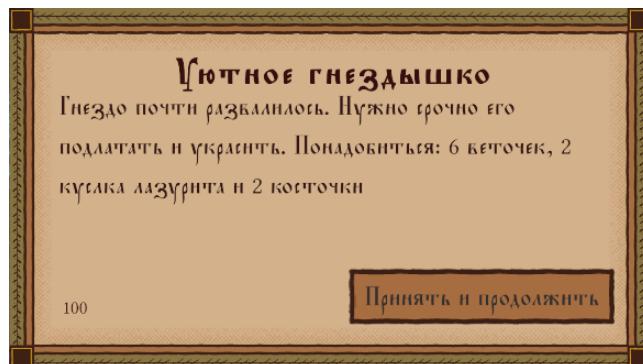


Рисунок 10 – Окно принятия задания. От сюжетных заданий нельзя отказываться



Рисунок 11 – Флаг активного задания, реализован с помощью булевой переменной

Для передвижения используется стандартная раскладка - W,A,S,D. Скорость бега задается с помощью переменной типа int, а ее значение уже в инспекторе движка. Более подробно с кодом можно ознакомиться будет в папке с полной версией проекта на flash-накопителе. Анимация происходит благодаря внутреннему компоненту Unity - Animator (см. рисунок 14).

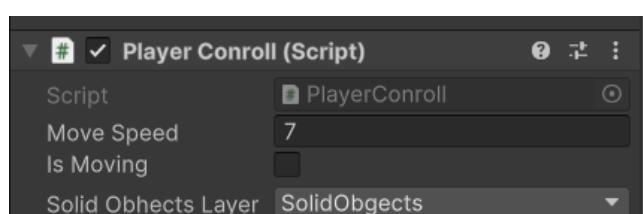


Рисунок 12 – Часть инспектора

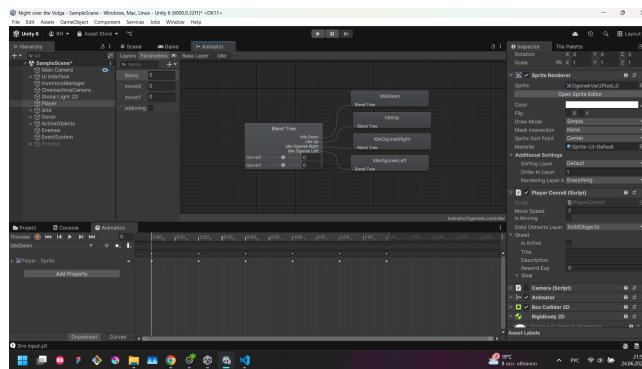


Рисунок 13 – Окно аниматора и анимаций

Атака же осуществляется на нажатие ПКМ, нажав ее персонаж выпустит "Огненный Всплеск базовая атака дальнего боя.



Рисунок 14 – Спрайт атаки

Как было представлено на рисунке 7, целью задания найти материалы, при прикосновение персонажа к предмету, он попадает в инвентарь и исчезает с локации. Если конечно же есть место в инвентаре, в противном случае объект останется на своем месте.

Код отвечающий за подбор предметов представлен ниже: С полным кодом отвечающим за функционирования инвентаря можно ознакомиться в приложение А.

```
public void Pickupitem(int id)
{
    bool result=inventoryManager.AddItem(itemsToPickUp[id]);
    if (result is true)
    {
        Debug.Log("Да");
    }
    else
```

```
{  
    Debug.Log("Нет");  
  
}  
}  
}
```

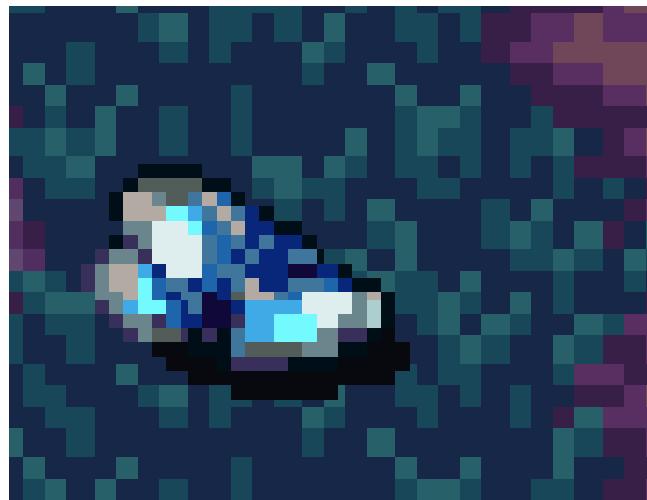


Рисунок 15 – Лазурит - один из материалов

2.2 Создание игровых классов и объектов

Самым первым объектом создавался "Player". Это один из главных объектов в проекте. В начале в окне иерархии создается пустой Game Object, затем к объекту прикрепляются компоненты, их можно увидеть на рисунке 16

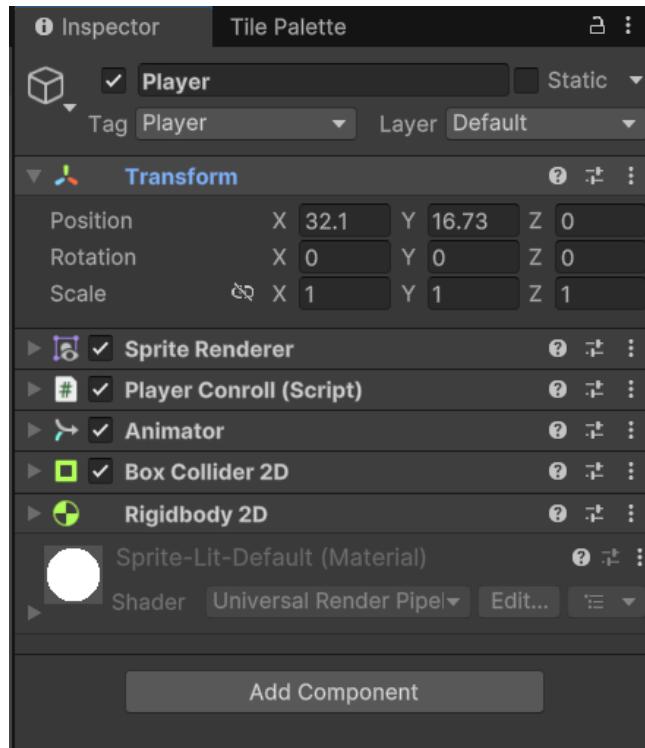


Рисунок 16 – Компоненты

Transform - отвечает за позицию, поворот и масштабирование объекта.

Player Conroll (Script) - код отвечающий за поведение персонажа.

Animator - Содержит анимации и их порядок воспроизведения.

Box Collider 2D - форма, определяющая форму объекта для целей физических столкновений.

Rigidbody 2D - объект становится физическим телом и имеет свою физику, которую можно при желании более детально настроить.

Часть кода для управление персонажем представлена ниже:

```
private void Awake()
{
    ogonekanimator= GetComponent<Animator>();
}
```

```
private void Update()
{
```

```

if (!IsMoving)
{
    input.x = Input.GetAxisRaw("Horizontal");
    input.y = Input.GetAxisRaw("Vertical");

    Debug.Log("To input.x" + input.x);
    Debug.Log("To input.y" + input.y);

    if (input.x != 0) input.y = 0;

    if (input != Vector2.zero)
    {
        ogonekanimator.SetFloat("moveX", input.x);
        ogonekanimator.SetFloat("moveY", input.y);

        var TargetPos = transform.position;
        TargetPos.x += input.x;
        TargetPos.y += input.y;

        if(IsWallcable(TargetPos))
            StartCoroutine(Move(TargetPos));
    }
}

ogonekanimator.SetBool("isMoving", IsMoving);
}

IEnumerator Move (Vector3 TargetPos)
{
    IsMoving= true;

    while((TargetPos- transform.position).sqrMagnitude > Mathf.Epsilon)
    {

```

```

        transform.position = Vector3.MoveTowards(transform.position, TargetPos,
        MoveSpeed * Time.deltaTime);
        yield return null;
    }

    transform.position = TargetPos;
    IsMoving= false;

}

private bool IsWallcable(Vector3 TargetPos)
{
    if(Physics2D.OverlapCircle(TargetPos, 0.1f, solidObhectsLayer) != null)
    {
        return false;
    }
    return true;
}

```

С интерфейсом похожая ситуация. Начало тоже самое, но в отличие от "Player используется не пустой Game Object, а специальный - Canvas. Создается родительский объект "UI interface" от которого будет зависеть интерфейс, к нему крепиться файл "WindowManager.cs" собственно он отвечает за корректную работу окон.

```

public class WindowManager : MonoBehaviour
{
    public static WindowManager Instance { get; private set; }

    private MenuWindow _lastActiveWindows;

    public bool HaveOpenWindow => _lastActiveWindows is not null;

```

```

public void Awake()
{
    Instance = this;
}

public void Open(MenuWindow menuWindow)
{
    if(_lastActiveWindows == menuWindow)
    {
        Close();
        return;
    }
    else if (_lastActiveWindows is not null)
    {
        _lastActiveWindows.Close();
    }

    _lastActiveWindows = menuWindow;
    menuWindow.Open();
}

public void Close()
{
    if(_lastActiveWindows is not null)
    {
        _lastActiveWindows.Close();
        _lastActiveWindows = null;
    }
}

```

Более подробно создание интерфейсов, будет рассмотрено в подпункте 2.5.

2.3 Создание 2D модели персонажа

В самом начале создаются несколько набросков персонажа. Они представляют идеи как примерно будет выглядеть персонаж. После отбирается наиболее удачный, начинается проработка деталей. После этого этапа выбираются базовые цвета. Иногда на этом этапе облик персонажа может претерпеть изменения, как видно на рисунках 1 и 2, приведенных выше в разделе 2.

Спрайт тоже претерпел изменения из-за изменения портрета.

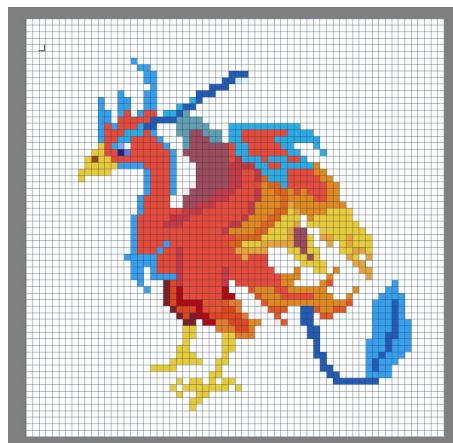


Рисунок 17 – Самый первый спрайт



Рисунок 18 – Финальный вариант

После того как был утвержден финальный вариант. Делаются кадры анимации. Так как игра изометрическая, то пришлось делать анимации для

трех позиций: влево-вправо(происходит отзеркаливание) лицом к игроку и спиной к игроку.



Рисунок 19 – Птица с разных сторон

2.4 Создание уровней

Создание уровней в Unity происходит с помощью так называемых сцен. Сцены содержат объекты игры, причем они могут использоваться как для создания меню, так и отдельных уровней.

Что бы создать уровень, нужно в начале создать Tile Palette. Для этого нужно создать текстуры окружение мира, т.е нарисовать их, либо использовать из бесплатных открытых асетов. Последний вариант был выбран. После этого нужно импортировать асет в Unity, когда он будет внутри движка, нужно открыть окно Tile Palette и в диалогом окне выбрать "New Palette" а после "Create New Palette". После нужно перенести текстуры в "Tile Palette" и нажать на кнопку сохранить, затем выбрать куда нужно сохранить Tile Asset [8].

Когда все будет сделано, можно будет "рисовать" получившимися текстурами. Когда окружение будет закончено, где есть текстуры объектов(например камней), надо добавить коллайдеры. Что бы игрок не проходил сквозь текстуры.

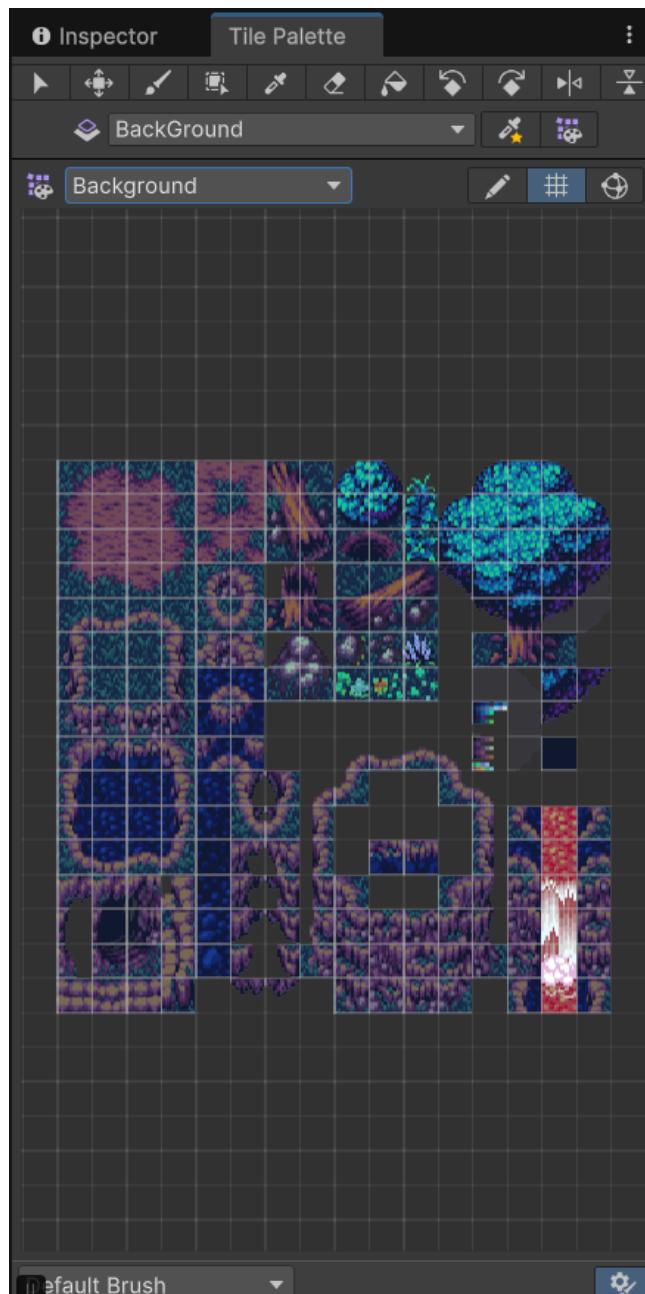


Рисунок 20 – Окно Tile Palette с текстурами

Существует множество способов создания коллайдеров. Один из них – создать новый "Tile Palette" с одним белым спрайтом. Назвать его к примеру "Solid". Но перед этим из белого спрайта сделать так называемый prefab и добавить ему компонент Collider 2D. После этого с обычными текстурами "красить" им нужные места.

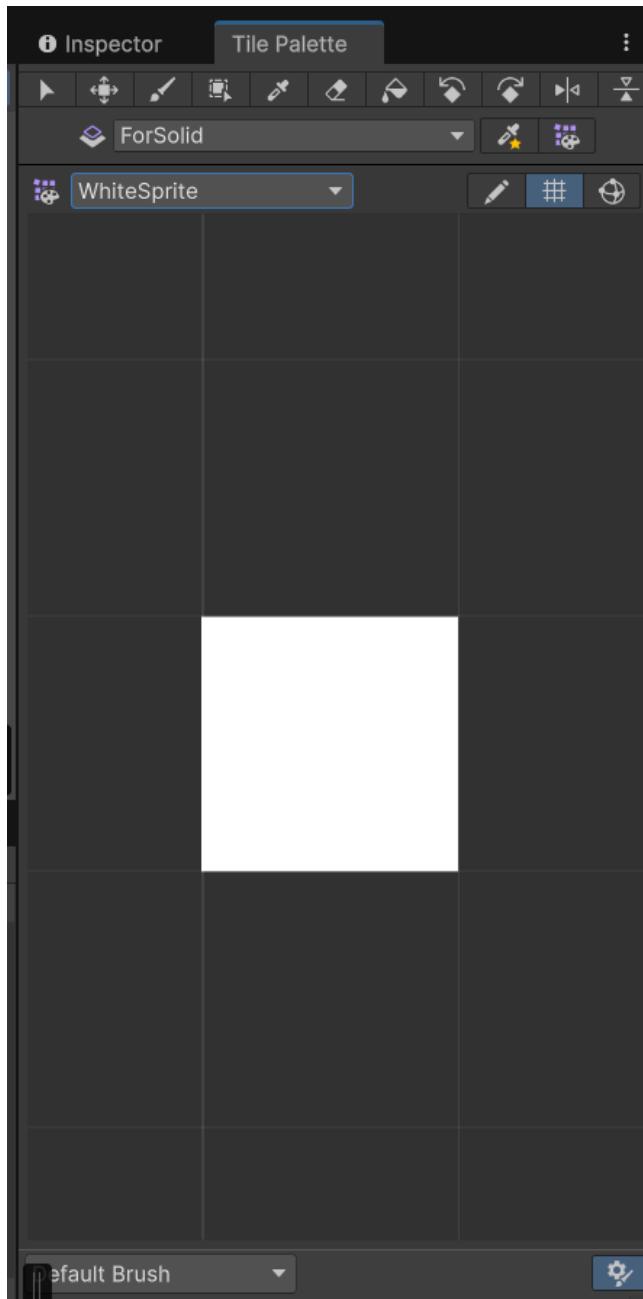


Рисунок 21 – Окно Tile Palette с белым спрайтом

Когда процесс будет завершен, нужно сделать спрайт прозрачным. Этот способ хорош тем, что нагрузка на движок и на компьютер снижается, благодаря чему приложение будет быстрее работать.

Когда уровень готов на него добавляются объекты и скрипты.

После чего идет тестирование на проверку "плотности" текстур, чтобы игрок случайно не прошел сквозь объект или стену. В конце проверяются предметы, тестировщик смотрит исчезают они при каких либо условиях, или как то ещё "странные" себя ведут особенно, если они заскриптованы. Наконец наступает очередь скриптов. Тестировщик всячески пытается выявить

"брешь" в них, сломать их. Иными словами проверяет их на "прочность". Это делается, чтобы понять, какие действия игрока могут сломать логику, к примеру тех же заданий, из-за чего оно может стать не выполнимым.

2.5 Создание интерфейсов

Интерфейс в Unity создается с помощью компонента "Canvas" [9]. Сам Canvas представляет собой абстрактное пространство, в котором производится настройка и отрисовка UI.

В начале создается родительский объект "UI interface". К нему крепиться весь интерфейс. Это сделано для того что бы каждый раз не задавать настройки окнам. Так все дочерние объекты, которыми являются окна, а также другие компоненты интерфейса, к примеру панель меню, будут наследовать параметры главного родительского объекта.

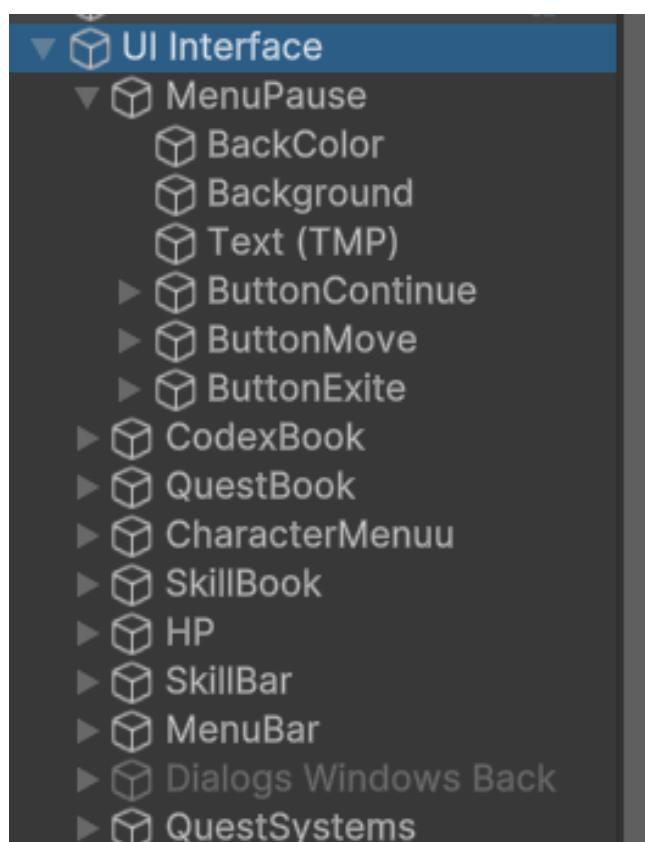


Рисунок 22 – Иерархия интерфейса

Так же к каждому дочернему объекту крепятся нужные компоненты. К примеру код. Каждое окно имеет свой код с соответствующим функционалом. Подробно с ним можно ознакомиться в папке с исходным кодом "Scripts" на flash-накопителе. Ниже приведен пример кода из файла MenuBar.cs отвечающий

за панель меню:

```
using UnityEngine;

public class MenuBar : MonoBehaviour
{
    public MenuPause menuPause;          // Pause Menu
    public CharacterMenu characterMenu; // Character
    public QuestBook questBook;         // Quests
    public CodexBook codexBook;
    public SpellBook spellBook;

    public void Update()
    {
        if(Input.GetKeyDown(KeyCode.Escape))
        {
            if (WindowManager.Instance.HaveOpenWindow)
            {
                WindowManager.Instance.Close();
            }
            else
            {
                OpenMenuPause();
            }
        }
    }

    public void OpenMenuPause()
    {
        WindowManager.Instance.Open(menuPause);
    }
}
```

```
public void OpenCodexBook()
{
    WindowManager.Instance.Open(codexBook);
}

public void OpenQuestBook()
{
    WindowManager.Instance.Open(questBook);
}

public void OpenCharacterMenu()
{
    WindowManager.Instance.Open(characterMenu);
}

public void OpenSkillBook()
{
    WindowManager.Instance.Open(spellBook);
}

}
```

2.6 Тестирование игры

Тестирование игры происходило в несколько этапов.

Самым первым этапом стало так называемое "модульное тестирование" [10]. Это проверка на корректность работы отдельных компонентов игры отдельно друг от друга.

Для примера можно взять кнопки кодекса, при нажатии раскрывается текст(см. рисунки 23 и 24).



Рисунок 23 – Текст по умолчанию скрыт

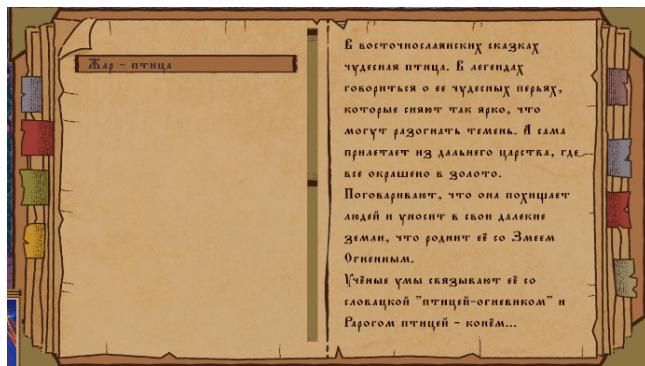


Рисунок 24 – Раскрытый текст

Вторым же этапом стало интеграционное тестирование. Это нужно, чтобы проверить как взаимодействуют компоненты между собой. К примеру: проверялось добавление предметов в инвентарь, так как, когда инвентарь попадает в инвентарь, он не просто исчезает, а становится "частью интерфейса".

А когда открыто любое окно интерфейса, оно "замораживает" все процессы в игре.

На этом основные этапы закончены. Далее идёт выявление критических ошибок, которые могут привести к вылету игры и прочему. Потом уже не значительных ошибок, одной из них стало сообщение об отсутствии canvas, данная ошибка не приводит ни к чему кроме сообщения в логе движка, а также к не самому плавному перетаскиванию объектов в инвентаре. По факту она видна только разработчику.

Так же одним из неприятных багов является прохождение текстур сквозь друг друга.



Рисунок 25 – Текстуры персонажа проходят сквозь дерево и наслаждаются на куст

ЗАКЛЮЧЕНИЕ

Процесс разработки игры – это сложный, многоэтапный и творческий процесс, требующий не только технических навыков, но и художественного видения. Несмотря на то, что современные игровые движки, такие как Unity, значительно упрощают многие аспекты разработки – от работы с графикой и физикой до управления анимацией и скриптами – создание качественной игры по-прежнему остаётся трудоёмкой задачей. Разработчикам приходится продумывать геймдизайн, баланс игровых механик, нарратив, уровень-дизайн, звуковое сопровождение и оптимизацию под разные платформы.

В ходе данной работы были успешно выполнены ключевые задачи, направленные на разработку базовых механик игры. Так же были изучены игровые движки, что позволило выбрать самый подходящий для эффективной разработки. В будущем проект можно будет доработать, что бы он стал полноценной игрой, которую можно будет выпустить на рынок.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

- 1 Andrew Rollings, Ernest Adams. Andrew Rollings and Ernest Adams on Game Design. — Indianapolis : New Riders, 2003.
- 2 Баркова А. Славянские мифы от Велеса и Мокоши до Птицы Сирин и Ивана Купалы. — Москва : МИФ, 2021.
- 3 Программирование в Unity. — URL: <https://unity.com/ru/solutions/programming> (дата обращения: 09.06.2025). [Электронный ресурс]. Загл. с экрана. Яз. рус.
- 4 How Unreal Engine 4 Will Change The Next Games You Play. — URL: <https://web.archive.org/web/20121024091239/http://kotaku.com/5916859/how-unreal-engine-4-will-change-the-next-games-you-play> (дата обращения: 11.06.2025). [Электронный ресурс]. Загл. с экрана. Яз. анг.
- 5 Sandbox. — URL: <https://web.archive.org/web/20120301145415/http://mycryengine.com/index.php?conid=53> (дата обращения: 11.06.2025). [Электронный ресурс]. Загл. с экрана. Яз. анг.
- 6 File system. — URL: <https://docs.godotengine.org/en/stable/tutorials/scripting/filesystem.html> (дата обращения: 11.06.2025). [Электронный ресурс]. Загл. с экрана. Яз. анг.
- 7 David Vinciguerra, Andrew Howell. The GameMaker Standard. — Boca Raton : CRC Press, 2015.
- 8 Create a Tile Palette. — URL: <https://docs.unity3d.com/6000.1/Documentation/Manual/tilemaps/tile-palettes/create-tile-palette.html> (дата обращения: 25.06.2025). [Электронный ресурс]. Загл. с экрана. Яз. анг.
- 9 Холст (Canvas). — URL: <https://docs.unity3d.com/ru/2019.4/Manual/class-Canvas.html> (дата обращения: 25.06.2025). [Электронный ресурс]. Загл. с экрана. Яз. рус.
- 10 Куликов С. Тестирование программного обеспечения. Базовый курс. 3-е издание. — Минск : Четыре четверти, 2020.

ПРИЛОЖЕНИЕ А

Полный код для работы инвентаря

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class InvenoryManager : MonoBehaviour
{
    public static InvenoryManager instance;
    public int maxStackItems = 5;
    public InventorySlot[] inventorySlots;
    public GameObject inventoryItemPrefab;
    [SerializeField] private Canvas _canvas;

    private void Awake()
    {
        instance=this;
    }

    public bool AddItem(Item item)
    {
        // Check slot
        for (int i = 0; i < inventorySlots.Length; i++)
        {
            InventorySlot slot = inventorySlots[i];
            InvenoryItem itemSlot = slot.GetComponentInChildren<InvenoryItem>();
            if (itemSlot != null &&
                itemSlot.item==item &&
                itemSlot.count<maxStackItems &&
                itemSlot.item.stackbale==true)
            {
                itemSlot.count++;
            }
        }
    }
}
```

```

        itemSlot.RefreshCount();
        return true;
    }
}

//



// Check free slot
for(int i=0;i<inventorySlots.Length;i++)
{
    InventorySlot slot = inventorySlots[i];
    InvenoryItem itemSlot=slot.GetComponentInChildren<InvenoryItem>();
    if(itemSlot is null)
    {
        SpawnNewItem(item,slot);
        return true;
    }
}
return false;
//


}

void SpawnNewItem(Item item, InventorySlot slot)
{
    GameObject newItemGo = Instantiate(inventoryItemPrefab, slot.transform);
    InvenoryItem inventoryItem = newItemGo.GetComponent<InvenoryItem>();
    inventoryItem.InitialiseItem(item,_canvas);

}

using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.UI;
using UnityEngine.EventSystems;
using Unity.VisualScripting;

```

```

public class InvenoryItem : MonoBehaviour, IBeginDragHandler, IDragHandler, IEndDragHandler
{
    [Header("UI")]
    public Image image;
    public Text countText;

    [UnitHeaderInspectable] public Item item;
    [UnitHeaderInspectable] public int count = 1;
    [HideInInspector] public Transform parentAfterDrag;

    public void InitialiseItem(Item newItem, Canvas canvas)
    {
        _canvas = canvas;
        item = newItem;
        image.sprite = newItem.image;
        RefreshCount();
    }

    public void RefreshCount()
    {
        countText.text = count.ToString();
        bool textActive = count > 1;
        countText.gameObject.SetActive(textActive);
    }

    [SerializeField] RectTransform rectTransform;
    [SerializeField] Canvas _canvas;
    public void OnBeginDrag(PointerEventData eventData)
    {

```

```

        image.raycastTarget = false;
        parentAfterDrag=transform.parent;
        transform.SetParent(transform.root);

    }

    public void OnDrag(PointerEventData eventData)
    {
        rectTransform.anchoredPosition += eventData.delta / _canvas.scaleFactor;

    }

    public void OnEndDrag(PointerEventData eventData)
    {
        image.raycastTarget = true;
        transform.SetParent(parentAfterDrag);
    }

}

using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.UI;
using UnityEngine.EventSystems;
using Unity.VisualScripting;

public class InventorySlot : MonoBehaviour, IDropHandler
{
    public void OnDrop(PointerEventData eventData)
    {
        if (transform.childCount == 0)
        {
            InventoryItem inventoryItem = eventData.pointerDrag.GetComponent<InventoryItem>();
            inventoryItem.parentAfterDrag = transform;
        }
    }
}

```

```
}
```

```
using System;  
using System.Collections;  
using System.Collections.Generic;  
using UnityEngine;
```

```
public class Loot : MonoBehaviour  
{
```

```
    [SerializeField] private Item _item;  
    private bool _isCollected = false;
```

```
    private void OnTriggerEnter2D(Collider2D collision)  
    {  
        if (collision.CompareTag("Player"))  
        {  
            if (_isCollected) return;  
            _isCollected = true;  
            bool canAdd = InvenotyManager.instance.AddItem(_item);  
  
            if (canAdd)  
            {  
                Destroy(gameObject);  
            }  
        }  
    }  
}
```

ПРИЛОЖЕНИЕ Б

Flash-носитель с отчетом о выполненной работе

На приложенном flash-накопителе можно ознакомиться со следующими файлами:

Папка Course — L^AT_EX- вариант курсовой работы;

Папка Scripts — весь исходный код игры;

Папка Night over the Volga — игра;

Course.pdf — курсовая работа.