Министерство науки и высшего образования Российской Федерации

ФГБОУ ВО «Алтайский государственный университет»

Школа развития цифровых компетенций «Digital Up» (цифровая кафедра)

**ИНДИВИДУАЛЬНОЕ ЗАДАНИЕ**

**на итоговый проект «Разработка карточной игры Before Justice»**

**по ДПП ПП «Основы Gamedev и VR-разработки»**

|  |  |  |  |
| --- | --- | --- | --- |
| п/п  № | Задание | Исполнитель | Рабочий график (план) выполнения |
| 1 | Проектирование архитектуры и баланса игры | Скоснягин М.С. | 10.05.2025 – 19.05.2025 |
| 2 | Отрисовка спрайтов и элементов UI | 20.05.2025 – 26.05.2025 |
| 3 | Разработка системы карточных боёв и реализация UI элементов и переходов между сценами | 27.05.2025 – 20.06.2025 |
| 4 | Подготовка отчета и презентации | 20.06.2025 – 25.06.2025 |

Руководитель проекта   
Старший преподаватель кафедры культурологии и дизайна Каратаев Алексей Антонович

\_\_\_\_\_\_\_\_\_\_\_\_\_ /\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ / «\_\_\_ » \_\_\_\_\_\_\_\_\_ 2025 г.

(подпись)

СОГЛАСОВАНО:

Руководитель ДПП ПП

канд. физ.-мат. наук, доцент Козлов Д.Ю.

\_\_\_\_\_\_\_\_\_\_\_\_\_ /\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_/ «\_\_\_\_»\_\_\_\_\_\_\_\_\_\_2025 г.

(подпись)

Министерство науки и высшего образования Российской Федерации

ФГБОУ ВО «Алтайский государственный университет»

Школа развития цифровых компетенций «Digital Up» (цифровая кафедра)

Отчет о выполнении итогового проекта по ДПП ПП

«**Основы Gamedev и VR-разработки**»

**«Разработка карточной игры Before Justice»**

Исполнитель:

Скоснягин М.С

«\_\_\_\_» \_\_\_\_\_\_\_\_ 2025 г.

Руководитель проекта:

Старший преподаватель кафедры культурологии и дизайна

Каратаев Алексей Антонович

«\_\_\_\_» \_\_\_\_\_\_\_\_ 2025 г.

г. Барнаул, 2025

**Оглавление**

[Введение 4](#_Toc201940249)

[ГЛАВА I. АНАЛИТИЧЕСКАЯ ЧАСТЬ 5](#_Toc201940250)

[1.1 Обоснование выбранной темы и описание игры 5](#_Toc201940251)

[1.2 Анализ основных игровых механик 5](#_Toc201940252)

[1.3 Подбор и анализ референсов 6](#_Toc201940253)

[1.4 Балансировка и математический анализ игровых механик 6](#_Toc201940254)

[1.5 Инструменты разработки 7](#_Toc201940255)

[ГЛАВА II. РАЗРАБОТКА ПРОЕКТА 8](#_Toc201940256)

[2.1 Архитектура системы 8](#_Toc201940257)

[2.2 Ключевые алгоритмы системы 8](#_Toc201940258)

[2.3 Основные связи объектов и элементов UI 9](#_Toc201940259)

[2.4 Тестирование игры на работоспособность и соответствие заданным требованиям 10](#_Toc201940260)

[Вывод 11](#_Toc201940261)

[Приложение 12](#_Toc201940262)

# **Введение**

**Цель проекта -** разработка мобильной карточной игры в жанре rogue-like “Before Justice”.

**Задачи проекта:**

* Проектирование архитектуры и баланса игры
* Создание спрайтов игровых объектов и необходимых элементов интерфейса
* Реализация системы пошаговых карточных боёв
* Реализация элементов UI и перехода между игровыми сценами
* Подведение вывода по результатам работы

**Актуальность и востребованность проекта:**

Мобильные rogue-like игры с цикличным геймплеем остаются популярными благодаря своей реиграбельности и адаптивности под короткие игровые сессии. Такие проекты, как "Slay the Spire" и "Balatro", доказали, что даже сложные механики могут быть удобны для платформ с сенсорным управлением. Во-вторых, сеттинг мистического вестерна редко используемый, но перспективный вариант, который может привлечь внимание игроков, уставших от стандартных фэнтези или постапокалиптических миров. Это подтверждается успехом как инди проектов вроде "Darkwatch" или "Weird West", так и крупных, как "Hunt: Showdown 1896" где атмосфера мистического дикого запада стала одним из основных преимуществ.

# **ГЛАВА I. АНАЛИТИЧЕСКАЯ ЧАСТЬ**

* 1. **Обоснование выбранной темы и описание игры**

Выбор темы для итогового проекта обусловлен уникальным сочетанием жанровых особенностей карточных rogue-like игр и атмосферы мистического вестерна, что редко встречается на игровом рынке. "Before Justice" представляет собой однопользовательскую карточную игру, где игрок принимает роль помощника шерифа, выжившего после таинственной катастрофы в шахтёрском городке. Игровой процесс построен на цикле "бой - подготовка - следующий бой" с постоянной ротацией противников и ограниченными ресурсами действий.

Сеттинг мистического вестерна реализован через визуальный стиль пиксель-арт. Сюжетная канва минималистична - герой должен выживать в окружении враждебных существ в попытке сбежать из города. Уникальность проекта заключается в адаптации классических rogue-like механик под тематику Дикого Запада, где вместо магических заклинаний игрок использует револьверы, динамит и подручные средства.

* 1. **Анализ основных игровых механик**

Основу геймплея составляет система пошаговых карточных боёв, где игрок управляет действиями героя через ограниченный набор карт. Каждый бой начинается с трёх действий, которые тратятся на использование карт из руки. Ключевой механикой является разделение оружия на огнестрельное (требующее патронов) и холодное (не требующее боеприпасов), что создаёт стратегический выбор между мощными, но ограниченными атаками и стабильным, но слабым уроном.

Особого внимания заслуживает система динамита - игрок должен активировать карту в один ход (что тратит действие), а использовать в следующий, рискуя получить урон при задержке. Враги реализуют различные тактики боя: Призрачный Скелет делает точные выстрелы, Высокая Гончая часто уклоняется от атак, а Железный Шахтёр полагается на огромный запас здоровья.

* 1. **Подбор и анализ референсов**

За основу игровых механик взяты принципы "Slay the Spire" - цикличность прохождения, построение стратегии вокруг ограниченного набора карт и перманентная смерть. Однако "Before Justice" отличается упрощённой экономикой действий и большим акцентом на управление ресурсами (патронами, аптечками и ходами). Из "Darkest Dungeon" заимствована атмосфера нарастающего безумия и давления, адаптированная под сеттинг вестерна.

Сеттинг вдохновлён такими играми как "Hunt: Showdown" и "Blood West", где историческая достоверность сочетается с сверхъестественными элементами. Визуальное решение основано на анализе современных пиксель-арт проектов жанра rogue-like типа "The binding of Isaac", но с упором на мрачную палитру и отсутствие анимаций. Ключевое отличие от референсов - фокус на тактическом планировании 2-3 ходов вперёд при управлении ограниченными ресурсами в условиях постоянной угрозы.

* 1. **Балансировка и математический анализ игровых механик**

Балансировка врагов проводилась на основе формулы угрозы (угроза = (средний урон+здоровье)\*точность). Призрачный Скелет получил 50 HP, урон 20-30 с точностью 85% и угрозой = 63,75. Высокая Гончая с 30 HP, уроном 10-20 и точностью 95% имеет угрозу = 42,75 . Железный Шахтёр с 90 HP, уроном 5-15 и точностью 75% получил угрозу = 75 (за счёт высокого хп).

Оружие балансировалось по показателю DPA (Damage Per Action) c учётом точности и количества патронов. Дерринджер (урон 10-13, 4 патрона, точность 80%) имеет DPA=9,2 ((10+13)/2\*0.8), слабый, но многозарядный и попадается часто. Револьвер (12-17, 6 патронов, точность 85%) - DPA=12,3 не очень редкий, с хорошей точностью и уроном. Обрез дробовика (20-30, 2 патрона, точность 95%) - DPA=23,75 более редкое оружие, мощное, но всего 2 патрона. Кремневый пистолет (35-50, 1 патрон, точность 70%) - DPA=29,75 с мощной пулей, редкий, но неточный и однозарядный. Ближнее оружие – нож (9-12 урона, точность 85%) DPA=8,925. Кастет (7-10 урона, точность 95%) DPA=8,075. Для карты "Неизвестная бутылка" выбран баланс: 50% шанс лечения на 40 HP, 50% шанс урона на 15 HP. Большая аптечка лечит 35 единиц здоровья, а бинт 17. Динамит с уроном 30-50 требует двух действий (DPA=40), но наносит половину урона игроку при задержке применения, создавая рискованный, но мощный инструмент.

* 1. **Инструменты разработки**

В качестве основного движка был выбран Unity 2022.3.61f1 (LTS) - стабильная версия, обеспечивающая поддержку современных функций и совместимость с мобильными платформами. Этот выбор обусловлен гибкостью движка, наличием встроенных инструментов для 2D-разработки и удобной системой управления сценами. Unity также предоставляет готовые решения для UI, что ускорило реализацию карточных механик и перетаскивания объектов.

Для создания пиксель-арт графики использовался Aseprite - специализированный редактор, позволяющий быстро работать со спрайтами, анимацией и палитрами. Его ключевые преимущества — удобные инструменты для пиксель-арта (дроуинг, шейдинг, и т.д.) и экспорт в оптимизированные форматы. Код писался в Visual Studio Code с плагинами для Unity, что упростило навигацию по проекту и отладку. Дополнительно применялись стандартные инструменты Unity: TextMeshPro для читаемого UI и Input System для обработки сенсорного ввода.

# **ГЛАВА II.** **РАЗРАБОТКА ПРОЕКТА**

* 1. **Архитектура системы**

Архитектура игры построена по модульному принципу с использованием паттерна State Machine для управления игровыми состояниями. Центральным элементом является класс GameManager.cs, который контролирует переходы между основными состояниями: MainMenu, Battle, Intermission (фаза между боями) и GameOver. Класс BattleSystem.cs обрабатывает логику боя, используя систему ходов, реализованную через TurnSystem.cs - компонент, который отслеживает количество оставшихся действий и автоматически передаёт ход врагам при их исчерпании.

Класс EnemyManager.cs отвечает за генерацию врагов, создавая экземпляры противников на основе заранее заданных префабов. PlayerHand.cs управляет картами в руке игрока, реализуя логику их размещения, добавления и удаления. Важным компонентом является IntermissionManager.cs, который контролирует фазу между боями - генерацию новых карт для выбора. Все сущности разделены на логические компоненты (например, WeaponCard.cs, HealCard.cs) и визуальное представление, что упрощает модификацию и расширение.

* 1. **Ключевые алгоритмы системы**

Алгоритм генерации врагов, реализованный в классе EnemyManager.cs, использует простой случайный выбор из списка префабов без взвешенных вероятностей. В методе SpawnNewEnemy() враг выбирается через enemyPrefabs[Random.Range(0, enemyPrefabs.Count)], что обеспечивает равномерное распределение между тремя типами врагов без учёта прогресса. Система карт в IntermissionManager.cs генерирует предложения через случайный выбор пула карт (cardPools[Random.Range(0, cardPools.Count)]), где каждый пул содержит фиксированный набор из трёх карт.

ИИ врагов, реализованный в Enemy.cs, использует простую детерминированную логику: в методе AttackPlayer() враг всегда атакует игрока при наступлении его хода, проверяя только точность (if (Random.value > accuracy) для определения промаха. Алгоритм динамита в DynamiteCard.cs основан на флаге isActivated: при активации запускается таймер в Update(), и если карта не использована до проверки в CheckForExplosion(), вызывается ExplodeInHand().

* 1. **Основные связи объектов и элементов UI**

Интеграция пользовательского интерфейса реализована через систему централизованных менеджеров. Класс GameManager координирует переходы между состояниями игры (меню, бой, проигрыш), активируя соответствующие панели через прямые ссылки на UI-объекты. При запуске игровой сцены он автоматически скрывает меню и инициализирует HUD, включая отображение ходов через TurnSystem, который обновляет текстовое поле turnsText при любом изменении количества действий.

Система управления картами в PlayerHand использует динамический расчёт позиций: в методе UpdateCardPositions() на основе количества карт в руке и заданного расстояния (cardSpacing) вычисляются равномерные позиции с помощью линейной интерполяции. Это обеспечивает плавное перемещение карт при получении новых или использовании существующих. Для элементов здоровья реализована событийная модель: при изменении HP игрока (PlayerHealth) или врагов (Enemy) мгновенно обновляются соответствующие текстовые поля (healthText/hpText) через вызовы UpdateHealthUI().

Панель обучения и экран проигрыша — управляются через MenuManager и GameOverManager соответственно. При открытии панели обучения через ToggleTutorial() происходит приоритизация слоёв: панель перемещается поверх других элементов через transform.SetAsLastSibling(), гарантируя корректное отображение. Экран проигрыша синхронизируется с игровой статистикой: при вызове ShowGameOver() количество убитых врагов передаётся напрямую из GameStats в текстовое поле killsText, а кнопки рестарта подключаются к методам перезагрузки сцены или возврата в меню.

Для врагов применяются корутины смены цвета спрайта (DamageEffect - красный оттенок, DodgeEffect - смещение позиции), а для карт - стандартные анимации Unity при перетаскивании. Все связи между скриптами и UI-элементами установлены через инспектор Unity, что упрощает настройку и предотвращает ошибки поиска объектов во время выполнения.

Главное меню использует трёхкнопочную систему, где каждая кнопка вызывает метод MenuManager через событие onClick. Панель обучения реализована как отключаемый объект Canvas с независимым порядком рендеринга, что предотвращает конфликты слоёв. При проигрыше экран завершения не просто отображает статистику, но и приостанавливает игровое время через Time.timeScale=0, блокируя взаимодействие с игровыми объектами до выбора действия игроком.

* 1. **Тестирование игры на работоспособность и соответствие заданным требованиям**

Функциональное тестирование выявило и устранило критические ошибки: проблему с камерой в DraggableCard.GetMouseWorldPosition() (добавлена проверка if (mainCamera == null) mainCamera = Camera.main;), двойное списание ходов в фазе перехода (исправлено разделением TurnSystem и IntermissionManager), некорректное поведение карт после добавления в руку (реализована инициализация через InitializeComponents()).

Баланс-тестирование подтвердило соответствие характеристик врагов заявленным. Производились незначительные корректировки характеристик в процессе разработки. Юзабилити-тестирование выявило необходимость улучшения визуальной обратной связи - добавлены текстовые подсказки. Все базовые требования выполнены: реализованы 3 типа врагов, 6 видов оружия, система ограниченных ходов через TurnSystem.cs, цикличный геймплей через смену фаз в GameManager.cs. Средняя продолжительность сессии составляет 5-7 минут, что соответствует целевым показателям для мобильной игры в подобном жанре. Перманентная смерть реализована через сброс прогресса при поражении.

# **Вывод**

Разработка "Before Justice" стала комплексным проектом, в ходе которого удалось реализовать полноценную игровую систему в жанре карточного rogue-like. Ключевым достижением стало создание сбалансированного цикличного геймплея, где тактические решения игрока напрямую влияют на продолжительность и успех игры. Центральные механики - система ограниченных действий и рискованный менеджмент ресурсов создают напряжённую атмосферу постоянного выбора между атакой, защитой и подготовкой особых комбинаций.

Техническая реализация продемонстрировала эффективность выбранной архитектуры: модульная система с чётким разделением ответственности между компонентами (GameManager, BattleSystem, CardManager) обеспечила стабильную работу всех механик.

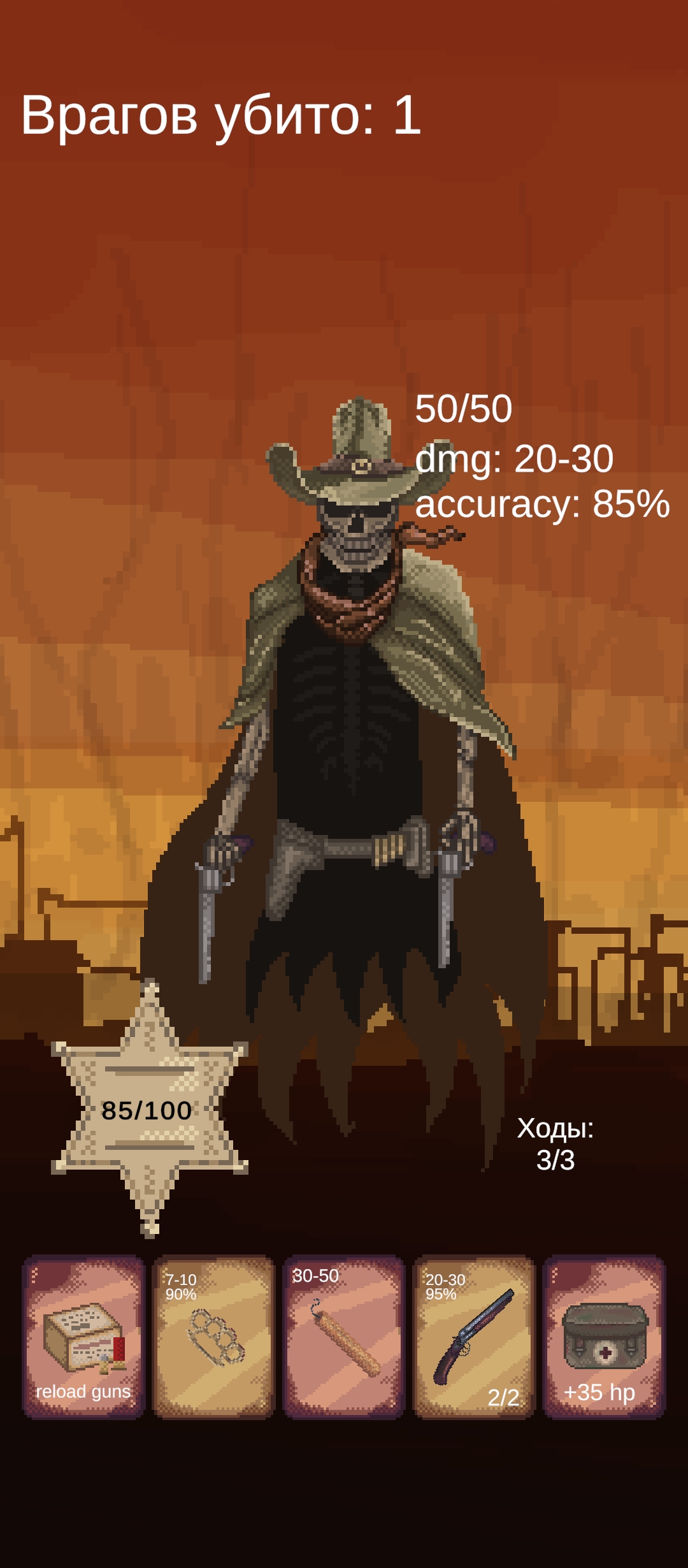
Тестирование подтвердило жизнеспособность концепции: средняя продолжительность сессии (7 мин) соответствует целевым показателям для мобильного формата. Выявились направления для развития: расширение пула врагов, добавление прокачки между попытками и внедрение нарративных элементов для усиления атмосферы.

В итоге "Before Justice" состоялся как минимально жизнеспособный продукт, демонстрирующий не только технические навыки разработки на Unity, но и понимание ключевых принципов геймдизайна. Проект доказал, что даже в ограниченных рамках можно создать запоминающийся игровой опыт, сочетающий тактическую глубину с уникальным сеттингом.

# **Приложение**

Приложение 1

Скриншоты



Приложение 2

Программный код (основные скрипты)

DraggableCard – механика пеетаскивания карт, обработка действий

using UnityEngine;

using System.Collections;

public class DraggableCard : MonoBehaviour

{

    [Header("Настройки")]

    public float returnSpeed = 0.4f;

    public bool IsDragging { get; private set; }

    private PlayerHand playerHand;

    private Vector3 startPosition;

    private Camera mainCamera;

    void Start()

    {

        InitializeComponents();

    }

    public void InitializeComponents()

    {

        playerHand = PlayerHand.Instance;

        mainCamera = Camera.main;

        startPosition = transform.position;

    }

    void OnEnable()

    {

        if (mainCamera == null) mainCamera = Camera.main;

    }

    void OnMouseDown()

    {

        if (Input.touchCount > 1) return;

        if (GetComponent<CardInHand>() == null) return;

        IsDragging = true;

        playerHand?.RemoveCardFromHand(transform);

        transform.SetAsLastSibling();

        startPosition = transform.position;

    }

    void OnMouseDrag()

    {

        if (!IsDragging) return;

        Vector3 mousePos = GetMouseWorldPosition();

        transform.position = new Vector3(mousePos.x, mousePos.y, transform.position.z);

    }

    void OnMouseUp()

    {

        if (!IsDragging) return;

        IsDragging = false;

        if (GameManager.Instance.currentState == GameManager.GameState.Intermission)

        {

            CheckIntermissionActions();

        }

        else

        {

            CheckBattleActions();

        }

    }

    void CheckBattleActions()

    {

        Collider2D[] hits = Physics2D.OverlapCircleAll(transform.position, 0.5f);

        foreach (var hit in hits)

        {

            HealZone healZone = hit.GetComponent<HealZone>();

            if (healZone != null && TryGetComponent(out HealCard healCard))

            {

                if (healCard.TryHeal())

                {

                    TurnSystem.Instance.SpendTurn();

                    return;

                }

            }

            if (TryGetComponent(out AmmoCard ammo) && hit.TryGetComponent(out WeaponCard weapon))

            {

                if (ammo.TryReloadWeapon(weapon))

                {

                    TurnSystem.Instance.SpendTurn();

                    return;

                }

            }

            Enemy enemy = hit.GetComponent<Enemy>();

            if (enemy != null)

            {

                if (TryGetComponent(out DynamiteCard dynamite))

                {

                    if (dynamite.TryUseDynamite(enemy))

                    {

                        return;

                    }

                }

                else if (TryGetComponent(out WeaponCard weaponCard))

                {

                    if (TryAttackEnemy(enemy))

                    {

                        TurnSystem.Instance.SpendTurn();

                        return;

                    }

                }

            }

        }

        StartCoroutine(ReturnToHand());

    }

    void CheckIntermissionActions()

    {

        Collider2D[] hits = Physics2D.OverlapCircleAll(transform.position, 0.5f);

        foreach (var hit in hits)

        {

            HealZone healZone = hit.GetComponent<HealZone>();

            if (healZone != null && TryGetComponent(out HealCard healCard))

            {

                if (healCard.TryHeal())

                {

                    IntermissionManager.Instance.SpendTurn();

                    return;

                }

            }

            if (TryGetComponent(out AmmoCard ammo) && hit.TryGetComponent(out WeaponCard weapon))

            {

                if (ammo.TryReloadWeapon(weapon))

                {

                    IntermissionManager.Instance.SpendTurn();

                    return;

                }

            }

            if (GetComponent<CardInHand>() != null)

            {

                foreach (var card in IntermissionManager.Instance.currentSelectionCards)

                {

                    if (hit.gameObject == card)

                    {

                        if (IntermissionManager.Instance.TryReplaceCard(transform, card))

                        {

                            return;

                        }

                    }

                }

            }

        }

        StartCoroutine(ReturnToHand());

    }

    IEnumerator ReturnToHand()

    {

        Vector3 startPos = transform.position;

        float elapsed = 0f;

        while (elapsed < returnSpeed)

        {

            transform.position = Vector3.Lerp(startPos, startPosition, elapsed/returnSpeed);

            elapsed += Time.deltaTime;

            yield return null;

        }

        playerHand?.AddCardToHand(transform);

    }

    Vector3 GetMouseWorldPosition()

    {

        if (mainCamera == null) mainCamera = Camera.main;

        Vector3 mousePos = Input.mousePosition;

        if (Input.touchCount > 0)

        {

            mousePos = Input.GetTouch(0).position;

        }

        mousePos.z = 10f;

        return mainCamera.ScreenToWorldPoint(mousePos);

    }

    bool TryAttackEnemy(Enemy enemy)

    {

        WeaponCard weapon = GetComponent<WeaponCard>();

        TurnSystem turnSystem = TurnSystem.Instance;

        if (!weapon.isMelee && weapon.currentAmmo <= 0)

        {

            Debug.Log("Нет патронов! Ход не тратится");

            StartCoroutine(ReturnToHand());

            return false;

        }

        if (weapon == null || !turnSystem.TryUseTurn())

        {

            StartCoroutine(ReturnToHand());

            return false;

        }

        bool attackSuccess = false;

        if (weapon.TryUseAmmo())

        {

            if (Random.value <= weapon.accuracy)

            {

                int damage = Random.Range(weapon.minDamage, weapon.maxDamage + 1);

                enemy.TakeDamage(damage);

                Debug.Log($"Нанесено {damage} урона!");

                attackSuccess = true;

            }

            else

            {

                Debug.Log("Промах!");

                enemy.PlayDodgeEffect();

                turnSystem.SpendTurn();

            }

        }

        StartCoroutine(ReturnToHand());

        return attackSuccess;

    }

}

GameManager – управление состояниями игры

using UnityEngine;

public class GameManager : MonoBehaviour

{

    public static GameManager Instance;

    public enum GameState { Battle, Intermission }

    public GameState currentState = GameState.Battle;

    void Awake()

    {

        if (Instance == null) Instance = this;

        else Destroy(gameObject);

    }

    public void StartIntermission()

    {

        currentState = GameState.Intermission;

        IntermissionManager.Instance.StartIntermission();

    }

    public void StartBattle()

    {

        currentState = GameState.Battle;

        EnemyManager.Instance.SpawnNewEnemy();

        TurnSystem.Instance.ResetTurns();

    }

}

PlayerHealth – контроль здоровья игрока

using UnityEngine;

using TMPro;

public class PlayerHealth : MonoBehaviour

{

    [Header("Настройки")]

    public int maxHP = 100;

    public int currentHP = 10;

    [Header("Ссылки")]

    public TextMeshProUGUI healthText;

    void Start()

    {

        currentHP = Mathf.Clamp(currentHP, 1, maxHP);

        UpdateHealthUI();

    }

public void TakeDamage(int damage)

{

    currentHP -= damage;

    currentHP = Mathf.Max(currentHP, 0); // Гарантируем неотрицательное HP

    UpdateHealthUI();

    if (currentHP <= 0 && !GameOverManager.Instance.gameOverPanel.activeSelf)

    {

        GameOverManager.Instance.ShowGameOver(GameStats.Instance.enemiesKilled);

    }

}

    public void Heal(int amount)

    {

        currentHP = Mathf.Min(currentHP + amount, maxHP);

        UpdateHealthUI();

    }

    void UpdateHealthUI()

    {

        if (healthText != null)

            healthText.text = $"{currentHP}/{maxHP}";

    }

}

TurnSystem – Управление игровыми ходами

using UnityEngine;

using TMPro;

using System.Collections;

public class TurnSystem : MonoBehaviour

{

    public static TurnSystem Instance;

    [Header("Настройки")]

    public int maxTurns = 3;

    public int currentTurns;

    public float enemyAttackDelay = 1f;

    [Header("Ссылки")]

    public TextMeshProUGUI turnsText;

    public Enemy enemy;

    void Awake()

    {

        if (Instance == null) Instance = this;

        else Destroy(gameObject);

    }

    void Start()

    {

        ResetTurns();

    }

    public void ResetTurns()

    {

        currentTurns = maxTurns;

        UpdateTurnsUI();

    }

    public bool TryUseTurn()

    {

        return currentTurns > 0;

    }

    public void SpendTurn()

    {

        if (currentTurns <= 0) return;

        currentTurns--;

        UpdateTurnsUI();

        if (currentTurns <= 0 && GameManager.Instance.currentState == GameManager.GameState.Battle)

        {

            StartCoroutine(EnemyAttackCoroutine());

        }

    }

    IEnumerator EnemyAttackCoroutine()

    {

        yield return new WaitForSeconds(enemyAttackDelay);

        DynamiteCard[] dynamites = FindObjectsOfType<DynamiteCard>();

        foreach (var dynamite in dynamites)

        {

            if (dynamite != null)

                dynamite.CheckForExplosion();

        }

        if (enemy != null)

        {

            enemy.AttackPlayer();

        }

        ResetTurns();

    }

    public void UpdateTurnsUI()

    {

        if (turnsText != null)

        {

            if (GameManager.Instance.currentState == GameManager.GameState.Intermission)

                turnsText.text = $"Ходы между боями: {IntermissionManager.Instance.currentIntermissionTurns}";

            else

                turnsText.text = $"Ходы: {currentTurns}/{maxTurns}";

        }

    }

}

PlayerHand – размещение карт в руке, добавление и удаление карт

using UnityEngine;

using System.Collections.Generic;

public class PlayerHand : MonoBehaviour

{

    public static PlayerHand Instance;

    [Header("Настройки")]

    public float cardSpacing = 2f;

    public float baseCardScale = 1.78f;

    public float moveSpeed = 10f;

    [Header("Начальные наборы")]

    public List<StartingHand> startingHands;

    public List<Transform> cardsInHand = new List<Transform>();

    public GameObject cardToReplace;

    void Awake()

    {

        if (Instance == null) Instance = this;

        else Destroy(gameObject);

    }

    void Start()

    {

        DrawStartingHand();

    }

    public void DrawStartingHand()

    {

        if (startingHands.Count == 0) return;

        StartingHand selectedHand = startingHands[Random.Range(0, startingHands.Count)];

        if (selectedHand.card1 != null) CreateCard(selectedHand.card1);

        if (selectedHand.card2 != null) CreateCard(selectedHand.card2);

        if (selectedHand.card3 != null) CreateCard(selectedHand.card3);

        if (selectedHand.card4 != null) CreateCard(selectedHand.card4);

        if (selectedHand.card5 != null) CreateCard(selectedHand.card5);

    }

    void CreateCard(GameObject cardPrefab)

    {

        GameObject newCard = Instantiate(cardPrefab, transform);

        AddCardToHand(newCard.transform);

    }

    void Update()

    {

        UpdateCardPositions();

    }

    public void UpdateCardPositions()

    {

        if (cardsInHand.Count == 0) return;

        float totalWidth = (cardsInHand.Count - 1) \* cardSpacing;

        float startX = -totalWidth / 2;

        for (int i = 0; i < cardsInHand.Count; i++)

        {

            if (cardsInHand[i] == null) continue;

            Vector3 targetPos = transform.position + new Vector3(startX + i \* cardSpacing, 0, 0);

            cardsInHand[i].position = Vector3.Lerp(

                cardsInHand[i].position,

                targetPos,

                moveSpeed \* Time.deltaTime

            );

            cardsInHand[i].localScale = Vector3.Lerp(

                cardsInHand[i].localScale,

                Vector3.one \* baseCardScale,

                moveSpeed \* Time.deltaTime

            );

        }

    }

    public void AddCardToHand(Transform card)

    {

        if (!cardsInHand.Contains(card))

        {

            cardsInHand.Add(card);

            card.SetParent(transform);

            if (!card.GetComponent<CardInHand>())

                card.gameObject.AddComponent<CardInHand>();

            DraggableCard draggable = card.GetComponent<DraggableCard>();

            if (draggable != null)

            {

                draggable.InitializeComponents();

                draggable.enabled = true;

            }

        }

    }

    public void RemoveCardFromHand(Transform card)

    {

        if (cardsInHand.Contains(card))

        {

            cardsInHand.Remove(card);

        }

    }

    public void ReplaceCard(Transform oldCard, GameObject newCard)

    {

        RemoveCardFromHand(oldCard);

        Destroy(oldCard.gameObject);

        AddCardToHand(newCard.transform);

    }

    public void DrawNewHand(bool keepExisting)

    {

        if (!keepExisting)

        {

            foreach (Transform card in cardsInHand.ToArray())

            {

                if (card != null) Destroy(card.gameObject);

            }

            cardsInHand.Clear();

        }

        if (GameManager.Instance.currentState == GameManager.GameState.Battle) return;

        int cardsToAdd = 5 - cardsInHand.Count;

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

        {

            if (IntermissionManager.Instance.cardPools.Count > 0)

            {

                CardPool randomPool = IntermissionManager.Instance.cardPools[

                    Random.Range(0, IntermissionManager.Instance.cardPools.Count)

                ];

                GameObject randomCard = randomPool.card1;

                if (randomCard != null) CreateCard(randomCard);

            }

        }

    }

}

[System.Serializable]

public class StartingHand

{

    public GameObject card1;

    public GameObject card2;

    public GameObject card3;

    public GameObject card4;

    public GameObject card5;

}

Приложение 3

Презентация проекта

https://docs.google.com/presentation/d/1cqJj3aEHWVKhqsJRb1OVnDd6YSFLcvOX/edit?usp=sharing&ouid=106429858777891067460&rtpof=true&sd=true

Приложение 4

Видео-презентация и демонстрация игры

https://drive.google.com/file/d/1z2OMasC6JUwj9V1gqNIR\_TPS-HsiiKew/view?usp=sharing

https://drive.google.com/file/d/1H9uyacKcdXcp76KM58HsIljFp\_bYP7EJ/view?usp=sharing