Министерство науки и высшего образования Российской Федерации

ФГБОУ ВО «Алтайский государственный университет»

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

**ЗАДАНИЕ**

**на производственную практику, разбор кейса «Разработать пошаговую боевую систему» по ДПП ПП «Школа развития цифровых компетенций «Digital Up» (Цифровая кафедра АлтГУ)»**

|  |  |  |  |
| --- | --- | --- | --- |
| п/п  № | Задание | Исполнитель | Рабочий график (план) выполнения |
| 1 | Изучить задание, принципы создания и работы боевой системы, а также необходимых навыков для ее разработки. | Воронова Т.А. Попова Е.Е.  Фейгина А.А. | 17.05.2024 – 20.05.2024 |
| 2 | Разработать пошаговую боевую систему в Unity, с демонстрацией (заданием) определенных параметров, а именно, ограничением шагов, совершением определенных действий и генерации карт восстановления. | 21.05.2024 – 27.05.2024 |
| 3 | Сформировать выводы по результатам выполненной работы. | 28.05.2024 – 30.05.2024 |
| 4 | Подготовить отчет о работе. | 30.05.2024 – 31.05.2024 |

Руководитель проекта   
преподаватель СПО Осыкин Д. А.

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

(подпись)

СОГЛАСОВАНО:

Руководитель ДПП ПП

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

\_\_\_\_\_\_\_\_\_\_\_\_\_ /\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_/ «\_\_\_\_»\_\_\_\_\_\_\_\_\_\_2024 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 группового задания по производственной практике по ДПП ПП

«Основы Gamedev и VR-разработки»

**«Разработка пошаговой боевой системы»**

Исполнители:

Воронова Т. А.

Попова Е. Е.

Фейгина А. А.

«\_\_\_\_» \_\_\_\_\_\_\_\_ 2025 г.

Руководитель проекта

преподаватель СПО Осыкин Д. А.

«\_\_\_\_» \_\_\_\_\_\_\_\_ 2025 г.

г. Барнаул, 2025

# Цель проекта

Цель проекта – разработка пошаговой боевой системы для игры, которая включает элементы стратегии и управления отрядами. Система должна обеспечивать динамичный игровой процесс с возможностью выбора действий, использования карточной механики и определения победителя по истечении заданного количества ходов.

# Задачи проекта и исполнители

Для достижения поставленной цели требовалось решить ряд задач:

1. Изучить задание, принципы создания и работы боевой системы, а также необходимых навыков для ее разработки.
2. Разработать пошаговую боевую систему в Unity, с демонстрацией (заданием) определенных параметров, а именно, ограничением шагов, совершением определенных действий и генерации карт восстановления, а именно:

* Создать базовые классы для боевых персонажей и отрядов с необходимыми характеристиками.
* Реализовать систему ходов и ограничение по времени на сражение.
* Проработать механику действий в ходе (атака, передвижение, суперудар, выбор заклинаний).
* Разработать систему карточной колоды для влияния на характеристики отрядов.
* Провести тестирование и отладку проекта

1. Сформировать выводы по результатам выполненной работы.

# Актуальность и востребованность проекта

Актуальность данного проекта высока, так как концепция пошаговых боевых систем занимает важное место в игровом дизайне. Подобные механики используются как в настольных, так и в видеоиграх, что обеспечивает им широкую аудиторию. Построение такой системы предоставляет игрокам возможность развивать стратегическое мышление и тактические навыки, что является привлекательным как для казуальных, так и для хардкорных игроков. Кроме того, успешная реализация проекта может послужить основой для будущих расширений и дополнений, таких как новые режимы игры, персонажи и карточные эффекты.

# Общие сведения о проделанной работе

Этапы выполнения проекта:

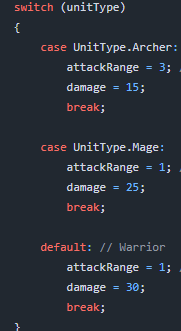
* Проанализировать требования и разработать концепцию системы (основные механики пошаговых боёв, карточных систем и взаимодействия персонажей; ограничение ходов, действия персонажей, систему карт и управление шкалой суперудара).
* Создать базовые классы и структуру данных.
* Разработать систему ходов и действий.
* Разработать интерфейс для выбора карт, отображения характеристик, анимаций и визуальных эффектов.
* Протестировать сценарии для проверки корректности работы системы.
* Сформировать выводы по работе.

Для разработки данной пошаговой боевой системы в Unity был выбран язык программирования C#, так как он является основным для движка Unity и обеспечивает богатый функционал для реализации игровых механик. В качестве среды разработки используется Unity 6000.0.42f1, что позволяет удобно создавать игровые объекты, управлять сценами и интегрировать визуальные элементы. Дополнительными библиотеками выступят встроенные возможности Unity, а также стандартные C# библиотеки для работы с коллекциями, случайными числами и таймерами.

Создание базовых игровых объектов включает в себя разработку классов для боевых персонажей, у которых есть свойства: здоровье, атака, защита, скорость и шкала суперудара. Эти свойства позволяют моделировать боевые действия и отслеживать состояние каждого персонажа. Также создается отдельный класс для отрядов, который объединяет нескольких персонажей и позволяет управлять их суммарными характеристиками, например, общим здоровьем. Такой подход помогает соблюдать требования по подсчёту общего здоровья отряда и упрощает управление группой.

Система ходов реализуется через счетчик, который увеличивается после каждого завершенного хода. После 10 ходов происходит финальный подсчет – сравниваются суммарные показатели здоровья обеих сторон, и определяется победитель или ничья. В процессе боя также реализуется генерация двух карт в начале каждого хода для каждого игрока. Эти карты содержат эффекты, которые могут изменить характеристики отряда, например, восстановить 15% здоровья всем членам, уменьшить защиту или увеличить силу атаки. Игроки выбирают одну из предложенных карт, и её эффект применяется к их отряду, что добавляет стратегический элемент в игру.

Механика действий в ходе включает три варианта: атака, передвижение и суперудар. Атака делится на три типа: физическая, стрельба и заклинания. У каждого игрока на поле находятся по 3 юнита с соответствующими навыками: воин, стрелок и маг.



Передвижение реализуется через изменение позиции персонажа на сцене, а суперудар активируется только при полном заполнении шкалы, которая увеличивается при успешном нанесении урона противнику. Это стимулирует агрессивную игру и правильное управление ресурсами.

Карточная система реализуется следующим образом: в начале каждого хода генерируются две случайные карты из заранее подготовленного набора. Каждая карта содержит эффект, который может изменить характеристики отряда — например, восстановить здоровье, повысить атаку, снизить защиту или даже мгновенно передать ход противнику. Игрок выбирает одну из карт, и её эффект применяется к отряду или к полю боя. После этого ход продолжается, и начинается новый цикл.

Дальнейшие этапы разработки включают создание визуальных элементов интерфейса, анимаций и звуковых эффектов для повышения погружения. Также планируется тестирование системы на баланс и исправление возможных ошибок. В будущем возможно расширение механик, добавление новых типов карт, заклинаний и стратегических элементов, что сделает игру более глубокой и интересной.

В целом, данная система позволяет реализовать пошаговую боевую механику с ограничением ходов, стратегическими выборами и динамическими изменениями характеристик, что делает игровой процесс увлекательным и вариативным.

# Результаты проекта

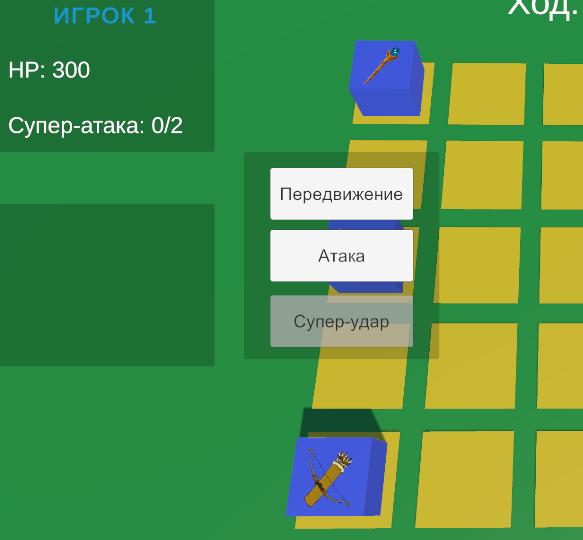
В результате разработки игры была создана комплексная система игрового процесса, которая охватывает множество действий и взаимодействий между боевыми персонажами.

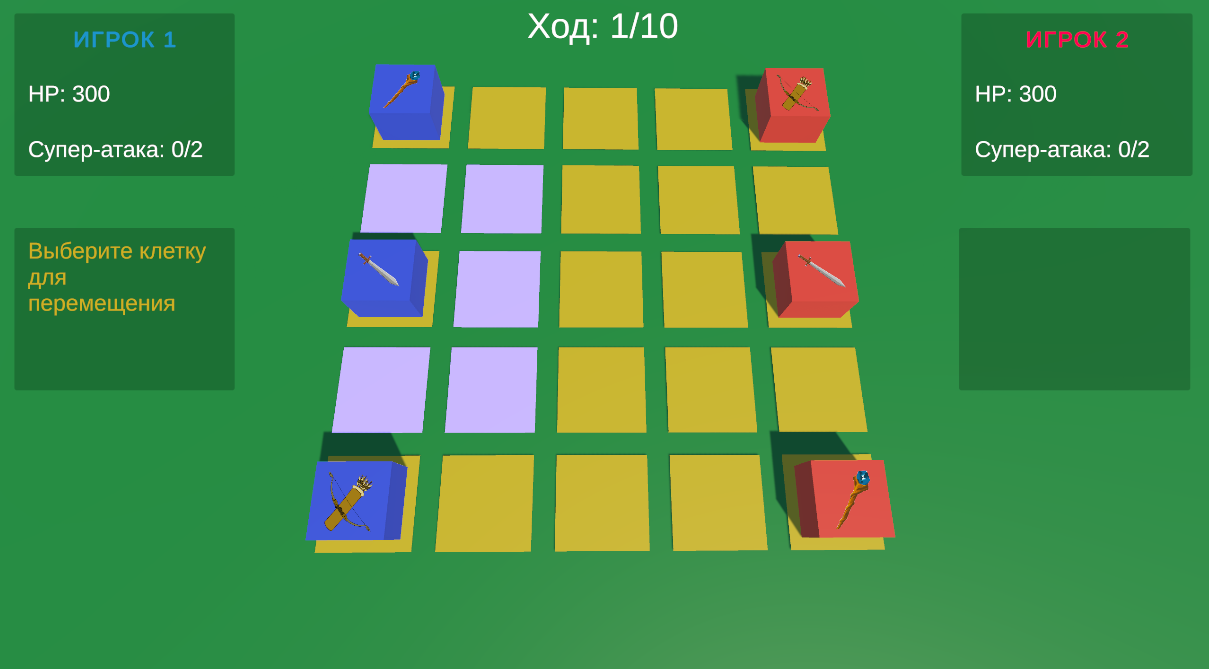
Рисунок демонстрирует систему ходов и счётчик, который увеличивается после завершения каждого хода. Считая до десяти, игроки должны взвешенно подходить к своим решениям, так как по истечении времени будет выявлен победитель. Рисунок также показывает, как на экране отображается текущее здоровье каждого отряда с помощью графических индикаторов.





На следующих рисунках показана механика действий хода (три возможных действия): атака, передвижение и суперудар. Каждый персонаж может выбрать только одно действие за ход. Можно увидеть, как персонаж движется на поле боя, воздействует на противника через физическую атаку или заклинание.





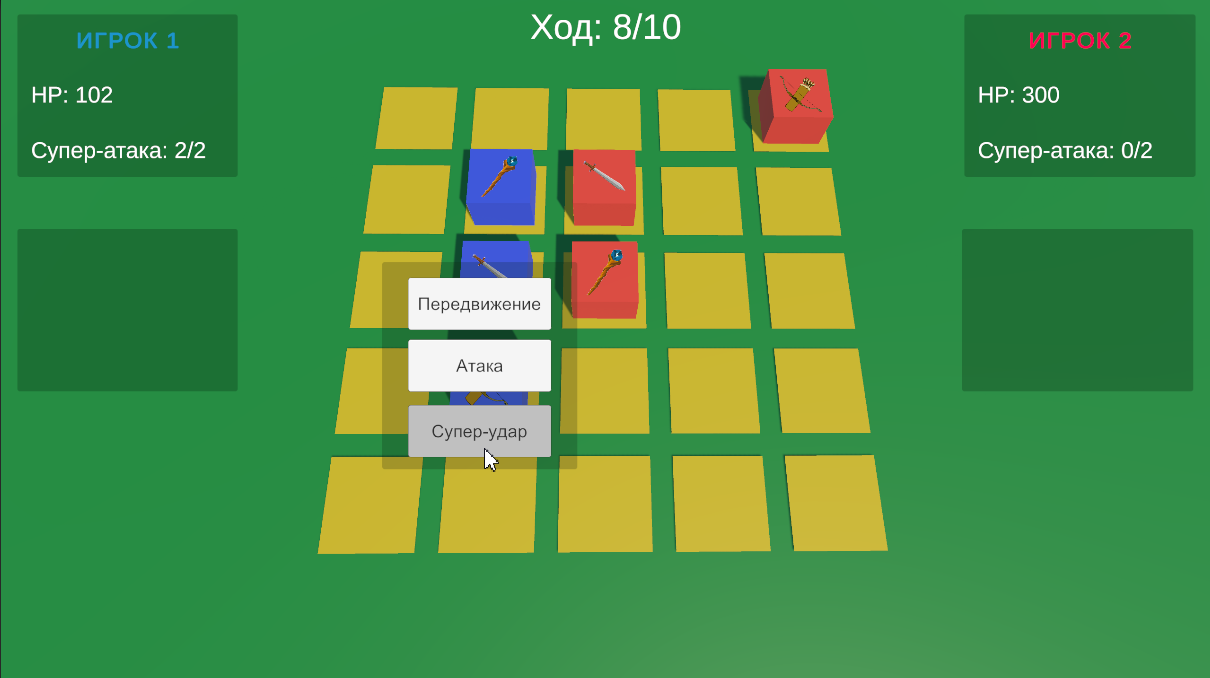
Далее показана иллюстрация двух карт, сгенерированных для каждого игрока в начале хода. На одной карте изображён эффект восстановления здоровья, а на другой – эффект повышения атаки. Также показан интерфейс выбора карты, где игроки могут кликнуть на желаемую карту для активации её эффекта.



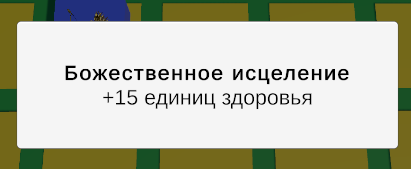
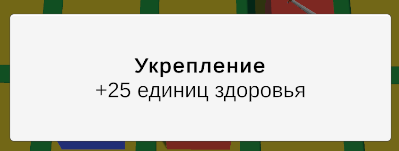
На следующих рисунках представлена сцена боя с участием персонажей. Можно увидеть, как персонаж атакует противника, нанося урон и применяя заклинание, что отражается соответствующим образом в игре.



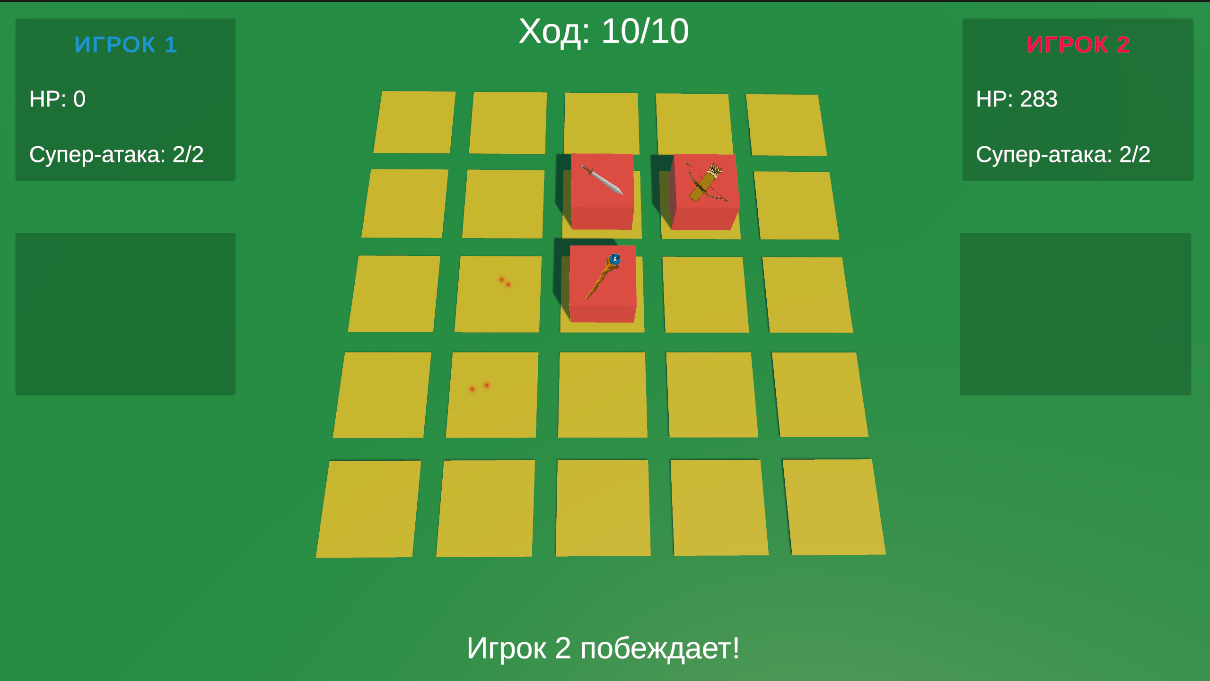
Далее показано применение суперудара после активации соответствующей кнопки.



На следующем рисунке изображен процесс применения карты, восстанавливающей здоровье. Игрок активирует карту, и соответствующий эффект отображается на здоровье каждого персонажа отряда.

Далее продемонстрирован подсчет оставшихся очков здоровья после завершения 10-ти ходов. На экране показаны обобщенные данные о здоровье каждого отряда, а также итоговая информация, указывающая победителя или ничью.



На основе собранных данных в ходе тестирования игровых механик можно сделать выводы о том, что игроки могут развивать свои навыки в стратегии и тактике, выбирая оптимальные действия и карты для достижения победы. Изменение характеристик через карты и возможность комбинировать действия персонажей создаёт уникальные игровые ситуации, что делает каждую партию неповторимой и играет важную роль в конечном результате. Благодаря визуальным эффектам и динамичным анимациям игроки могут лучше погрузиться в игру, а также развивать свои навыки в стратегии и тактике.

В будущем данная концепция может быть усовершенствована для разработки более сложных игр, а также адаптирована для других проектных направлений, таких как обучающие симуляторы или многопользовательские платформы с возможностью взаимодействия между игроками в реальном времени. Для этого можно будет расширить 3D-среду игры, чтобы привлечь больше пользователей и улучшить впечатления от игрового процесса.

**Приложение 1.**Программный код

ActionMenu.cs

using UnityEngine;

using UnityEngine.UI;

using UnityEngine.EventSystems;

public class ActionMenu : MonoBehaviour, IPointerEnterHandler, IPointerExitHandler

{

[SerializeField] private GameObject menuPanel;

[SerializeField] private Button moveButton;

[SerializeField] private Button attackButton;

[SerializeField] private Button superButton;

private UnitSelector unitSelector;

private bool isPointerOverMenu = false;

void Awake()

{

unitSelector = FindObjectOfType<UnitSelector>();

menuPanel.SetActive(false);

moveButton.onClick.AddListener(OnMoveClicked);

attackButton.onClick.AddListener(OnAttackClicked);

superButton.onClick.AddListener(OnSuperClicked);

}

public void OnPointerEnter(PointerEventData eventData)

{

isPointerOverMenu = true;

}

public void OnPointerExit(PointerEventData eventData)

{

isPointerOverMenu = false;

}

public bool IsPointerOverMenu() => isPointerOverMenu;

public void ShowMenu(Unit unit, Vector2 screenPosition)

{

menuPanel.transform.position = screenPosition;

menuPanel.SetActive(true);

superButton.interactable = TurnManager.Instance.CanUseSuper(unit.owner);

}

public void HideMenu()

{

menuPanel.SetActive(false);

}

private void OnMoveClicked()

{

unitSelector.StartMovement(unitSelector.SelectedUnit);

HideMenu();

}

private void OnAttackClicked()

{

unitSelector.StartAttack(unitSelector.SelectedUnit);

HideMenu();

}

private void OnSuperClicked()

{

unitSelector.StartSuperAttack(unitSelector.SelectedUnit);

HideMenu();

}

}

CardSystem.cs

using UnityEngine;

using UnityEngine.UI;

using System.Collections.Generic;

using TMPro;

public class CardSystem : MonoBehaviour

{

[System.Serializable]

public class CardEffect

{

public string cardName;

[TextArea] public string description;

public System.Action<Unit.UnitOwner> ApplyEffect;

}

[Header("UI References")]

[SerializeField] private GameObject cardPanel;

[SerializeField] private Button Card1;

[SerializeField] private Button Card2;

[SerializeField] private TextMeshProUGUI Card1Text;

[SerializeField] private TextMeshProUGUI Card2Text;

private List<CardEffect> allCards = new List<CardEffect>();

private CardEffect[] currentCards = new CardEffect[2];

private Unit.UnitOwner currentPlayer;

private void Awake()

{

InitializeAllCards();

cardPanel.SetActive(false);

Card1.onClick.AddListener(() => OnCardSelected(0));

Card2.onClick.AddListener(() => OnCardSelected(1));

}

private void InitializeAllCards()

{

allCards.Add(new CardEffect

{

cardName = "Божественное исцеление",

description = "+15 единиц здоровья",

ApplyEffect = (owner) => {

foreach (var unit in FindUnitsByOwner(owner))

{

unit.TakeDamage(-Mathf.RoundToInt(unit.CurrentHealth + 15));

}

}

});

allCards.Add(new CardEffect

{

cardName = "Ярость",

description = "+30% урона,\nно -15% здоровья",

ApplyEffect = (owner) => {

foreach (var unit in FindUnitsByOwner(owner))

{

unit.damage = Mathf.RoundToInt(unit.damage \* 1.3f);

unit.TakeDamage(Mathf.RoundToInt(unit.CurrentHealth \* 0.15f));

}

}

});

allCards.Add(new CardEffect

{

cardName = "Укрепление",

description = "+25 единиц здоровья",

ApplyEffect = (owner) => {

foreach (var unit in FindUnitsByOwner(owner))

{

unit.TakeDamage(-Mathf.RoundToInt(unit.CurrentHealth + 25));

}

}

});

allCards.Add(new CardEffect

{

cardName = "Стальная воля",

description = "Мечник получает +40% урона, другие юниты -10%",

ApplyEffect = (owner) => {

foreach (var unit in FindUnitsByOwner(owner))

{

if (unit.unitType == Unit.UnitType.Warrior)

unit.damage = Mathf.RoundToInt(unit.damage \* 1.4f);

else

unit.damage = Mathf.RoundToInt(unit.damage \* 0.9f);

}

}

});

allCards.Add(new CardEffect

{

cardName = "Магическая вспышка",

description = "Маг получает +50% урона, другие юниты -15%",

ApplyEffect = (owner) => {

foreach (var unit in FindUnitsByOwner(owner))

{

if (unit.unitType == Unit.UnitType.Mage)

unit.damage = Mathf.RoundToInt(unit.damage \* 1.5f);

else

unit.damage = Mathf.RoundToInt(unit.damage \* 0.85f);

}

}

});

allCards.Add(new CardEffect

{

cardName = "Равновесие",

description = "Выравнивает здоровье всех\nюнитов до среднего значения",

ApplyEffect = (owner) => {

var units = FindUnitsByOwner(owner);

int totalHealth = 0;

foreach (var unit in units) totalHealth += unit.CurrentHealth;

int averageHealth = units.Count > 0 ? totalHealth / units.Count : 0;

foreach (var unit in units)

{

unit.TakeDamage(-averageHealth);

}

}

});

}

private List<Unit> FindUnitsByOwner(Unit.UnitOwner owner)

{

Unit[] allUnits = FindObjectsOfType<Unit>();

List<Unit> playerUnits = new List<Unit>();

foreach (Unit unit in allUnits)

{

if (unit.owner == owner)

{

playerUnits.Add(unit);

}

}

return playerUnits;

}

public void StartCardSelection(Unit.UnitOwner player)

{

currentPlayer = player;

GenerateRandomCards();

ShowCardUI();

}

private void GenerateRandomCards()

{

int firstIndex = Random.Range(0, allCards.Count);

int secondIndex;

do

{

secondIndex = Random.Range(0, allCards.Count);

} while (secondIndex == firstIndex);

currentCards[0] = allCards[firstIndex];

currentCards[1] = allCards[secondIndex];

Card1Text.text = $"<b>{currentCards[0].cardName}</b>\n{currentCards[0].description}";

Card2Text.text = $"<b>{currentCards[1].cardName}</b>\n{currentCards[1].description}";

}

private void ShowCardUI()

{

cardPanel.SetActive(true);

Card1.interactable = true;

Card2.interactable = true;

}

private void HideCardUI()

{

cardPanel.SetActive(false);

}

private void OnCardSelected(int cardIndex)

{

if (cardIndex < 0 || cardIndex >= currentCards.Length) return;

Card1.interactable = false;

Card2.interactable = false;

currentCards[cardIndex].ApplyEffect(currentPlayer);

Debug.Log($"{currentPlayer} выбрал карту: {currentCards[cardIndex].cardName}");

Invoke("HideCardUI", 0.5f);

Debug.Log("Выбор карт завершен, можно продолжать игру");

}

}

Unit.cs

using UnityEngine;

public class Unit : MonoBehaviour

{

public enum UnitOwner { Player1, Player2 }

public enum UnitType { Warrior, Archer, Mage }

[Header("Basic Settings")]

public UnitOwner owner;

public UnitType unitType;

public int moveRange = 1;

[Header("Combat Stats")]

public int attackRange;

public int damage;

[SerializeField] private int \_baseHealth = 100;

[SerializeField] private int \_maxHealth = 100;

[SerializeField] private int \_currentHealth;

public int CurrentHealth => \_currentHealth;

public int MaxHealth { get; private set; }

void Start()

{

InitializeUnitStats();

MaxHealth = \_baseHealth;

\_currentHealth = MaxHealth;

}

private void InitializeUnitStats()

{

switch (unitType)

{

case UnitType.Archer:

attackRange = 3;

damage = 15;

break;

case UnitType.Mage:

attackRange = 1;

damage = 25;

break;

default:

attackRange = 1;

damage = 30;

break;

}

\_maxHealth = \_baseHealth;

}

public void TakeDamage(int amount)

{

\_currentHealth = Mathf.Clamp(\_currentHealth - amount, 0, MaxHealth);

if (TurnManager.Instance != null && UIManager.Instance != null)

{

int p1Health = TurnManager.Instance.CalculateTotalHealth(UnitOwner.Player1);

int p2Health = TurnManager.Instance.CalculateTotalHealth(UnitOwner.Player2);

UIManager.Instance.UpdateHealthDisplay(p1Health, p2Health);

}

if (\_currentHealth <= 0)

{

Die();

}

}

private void Die()

{

if (TurnManager.Instance != null && UIManager.Instance != null)

{

int p1Health = TurnManager.Instance.CalculateTotalHealth(UnitOwner.Player1);

int p2Health = TurnManager.Instance.CalculateTotalHealth(UnitOwner.Player2);

UIManager.Instance.UpdateHealthDisplay(p1Health, p2Health);

}

Destroy(gameObject);

}

public bool CanAttack(Vector3 targetPosition)

{

float distance = Vector3.Distance(

new Vector3(transform.position.x, 0, transform.position.z),

new Vector3(targetPosition.x, 0, targetPosition.z)

);

return distance <= attackRange \* 5f;

}

}

TurnManager.cs

using UnityEngine;

using TMPro;

public class TurnManager : MonoBehaviour

{

public static TurnManager Instance { get; private set; }

public enum PlayerTurn { Player1, Player2 }

public PlayerTurn CurrentTurn { get; private set; } = PlayerTurn.Player1;

[SerializeField] private UnitSelector unitSelector;

[SerializeField] private TileHighlighter tileHighlighter;

[SerializeField] private TMP\_Text victoryText;

[SerializeField] private CardSystem сardSystem;

private int turnCount = 1;

private const int maxTurns = 10;

private int actionsPerformed = 0;

private int player1SuperCharges = 0;

private int player2SuperCharges = 0;

void Awake()

{

if (Instance == null)

{

Instance = this;

}

else

{

Destroy(gameObject);

}

if (victoryText != null)

victoryText.gameObject.SetActive(false);

}

public void StartFirstTurn()

{

if (UIManager.Instance == null)

{

Debug.LogError("UIManager не инициализирован!");

return;

}

CurrentTurn = PlayerTurn.Player1;

turnCount = 1;

actionsPerformed = 0;

player1SuperCharges = 0;

player2SuperCharges = 0;

UIManager.Instance.UpdateTurnInfo(turnCount);

UIManager.Instance.UpdateActionCount(actionsPerformed);

UIManager.Instance.UpdateSuperMeter(0, 0);

UIManager.Instance.ShowHint(CurrentTurn, "Первый ход! Выберите юнита.");

Debug.Log("Начало игры - ход Player1");

}

void Start()

{

UIManager.Instance.InitializeUI();

CurrentTurn = PlayerTurn.Player1;

turnCount = 1;

actionsPerformed = 0;

UIManager.Instance.UpdateTurnInfo(turnCount);

UIManager.Instance.UpdateActionCount(actionsPerformed);

UIManager.Instance.UpdateSuperMeter(0, 0);

UIManager.Instance.ShowHint(CurrentTurn, "Первый ход! Выберите юнита.");

Debug.Log($"Сейчас ход: {CurrentTurn}");

сardSystem.StartCardSelection(CurrentTurn == PlayerTurn.Player1 ? Unit.UnitOwner.Player1 : Unit.UnitOwner.Player2);

player1SuperCharges = 0;

player2SuperCharges = 0;

UIManager.Instance.UpdateSuperMeter(0, 0);

}

public void EndTurn()

{

UnitSelector selector = FindObjectOfType<UnitSelector>();

if (selector != null) selector.DeselectUnit();

CurrentTurn = (CurrentTurn == PlayerTurn.Player1) ? PlayerTurn.Player2 : PlayerTurn.Player1;

actionsPerformed = 0;

UIManager.Instance.UpdateActionCount(actionsPerformed);

if (CurrentTurn == PlayerTurn.Player1)

{

turnCount++;

if (turnCount > maxTurns)

{

EndGame();

return;

}

}

UIManager.Instance.UpdateTurnInfo(turnCount);

UIManager.Instance.ShowHint(CurrentTurn, "Ваш ход! Выберите юнита.");

tileHighlighter.ClearHighlights();

unitSelector.DeselectUnit();

Debug.Log($"Ход переключен на: {CurrentTurn}");

сardSystem.StartCardSelection(CurrentTurn == PlayerTurn.Player1 ? Unit.UnitOwner.Player1 : Unit.UnitOwner.Player2);

}

public void ActionPerformed()

{

actionsPerformed++;

UIManager.Instance.UpdateActionCount(actionsPerformed);

if (actionsPerformed >= 1)

{

EndTurn();

}

}

public void RegisterAttack(Unit.UnitOwner owner)

{

if (owner == Unit.UnitOwner.Player1)

{

player1SuperCharges++;

if (player1SuperCharges > 2) player1SuperCharges = 2;

Debug.Log($"Player1 charges: {player1SuperCharges}");

}

else

{

player2SuperCharges++;

if (player2SuperCharges > 2) player2SuperCharges = 2;

Debug.Log($"Player2 charges: {player2SuperCharges}");

}

UIManager.Instance.UpdateSuperMeter(player1SuperCharges, player2SuperCharges);

}

public bool CanUseSuper(Unit.UnitOwner owner)

{

bool canUse = owner == Unit.UnitOwner.Player1 ?

player1SuperCharges >= 2 :

player2SuperCharges >= 2;

Debug.Log($"Can use super: {canUse}");

return canUse;

}

public void UseSuper(Unit.UnitOwner owner)

{

if (owner == Unit.UnitOwner.Player1)

{

player1SuperCharges = 0;

}

else

{

player2SuperCharges = 0;

}

UIManager.Instance.UpdateSuperMeter(player1SuperCharges, player2SuperCharges);

Debug.Log($"Super used by {owner}");

}

private void EndGame()

{

int p1Health = CalculateTotalHealth(Unit.UnitOwner.Player1);

int p2Health = CalculateTotalHealth(Unit.UnitOwner.Player2);

string result = p1Health > p2Health ? "Игрок 1 побеждает!" :

p2Health > p1Health ? "Игрок 2 побеждает!" : "Ничья!";

if (victoryText != null)

{

victoryText.text = result;

victoryText.gameObject.SetActive(true);

victoryText.alignment = TextAlignmentOptions.Center;

victoryText.fontSize = 48;

}

this.enabled = false;

unitSelector.enabled = false;

Time.timeScale = 0;

}

public int CalculateTotalHealth(Unit.UnitOwner owner)

{

Unit[] units = FindObjectsOfType<Unit>();

int total = 0;

foreach (Unit unit in units)

{

if (unit.owner == owner && unit.CurrentHealth > 0)

{

total += unit.CurrentHealth;

}

}

return total;

}

public void CompleteAction()

{

ActionPerformed();

tileHighlighter.ClearHighlights();

unitSelector.DeselectUnit();

}

}

UIManager

using UnityEngine;

using TMPro;

using System.Collections;

using static TurnManager;

public class UIManager : MonoBehaviour

{

public static UIManager Instance;

[Header("Player 1 UI")]

public TextMeshProUGUI player1HealthText;

public TextMeshProUGUI player1SuperText;

public TextMeshProUGUI player1HintText;

[Header("Player 2 UI")]

public TextMeshProUGUI player2HealthText;

public TextMeshProUGUI player2SuperText;

public TextMeshProUGUI player2HintText;

[Header("Common UI")]

public TextMeshProUGUI turnCountText;

public TextMeshProUGUI actionCountText;

void Awake()

{

if (Instance == null)

{

Instance = this;

DontDestroyOnLoad(gameObject);

}

else

{

Destroy(gameObject);

}

}

public void InitializeUI()

{

UpdateHealthDisplay(300, 300);

UpdateSuperMeter(0, 0);

UpdateTurnInfo(1);

UpdateActionCount(0);

ClearHints();

}

public void UpdateHealthDisplay(int p1Health, int p2Health)

{

player1HealthText.text = $"HP: {p1Health}";

player2HealthText.text = $"HP: {p2Health}";

}

public void UpdateSuperMeter(int p1Charges, int p2Charges)

{

player1SuperText.text = $"Супер-атака: {p1Charges}/2";

player2SuperText.text = $"Супер-атака: {p2Charges}/2";

}

public void UpdateTurnInfo(int turnCount)

{

turnCountText.text = $"Ход: {turnCount}/10";

}

public void UpdateActionCount(int actionsPerformed)

{

actionCountText.text = $"Действия: {actionsPerformed}/1";

}

public void ShowHint(TurnManager.PlayerTurn player, string message)

{

if (player == TurnManager.PlayerTurn.Player1)

{

player1HintText.text = message;

StartCoroutine(ClearHintAfterDelay(player1HintText, 3f));

}

else

{

player2HintText.text = message;

StartCoroutine(ClearHintAfterDelay(player2HintText, 3f));

}

}

public void ClearHints()

{

player1HintText.text = "";

player2HintText.text = "";

}

private IEnumerator ClearHintAfterDelay(TextMeshProUGUI hintText, float delay)

{

yield return new WaitForSeconds(delay);

hintText.text = "";

}

public void ShowContextMessage(string message)

{

if (TurnManager.Instance.CurrentTurn == PlayerTurn.Player1)

player1HintText.text = message;

else

player2HintText.text = message;

}

}

using UnityEngine;

using System.Collections.Generic;

public class TileHighlighter : MonoBehaviour

{

public static TileHighlighter Instance;

public Material highlightMaterial;

private List<GameObject> highlightedTiles = new List<GameObject>();

private Unit selectedUnit;

void Awake()

{

if (Instance == null)

{

Instance = this;

}

else

{

Destroy(gameObject);

}

}

public void SetSelectedUnit(Unit unit)

{

selectedUnit = unit;

}

public void HighlightTiles(Vector3 unitPos, int range)

{

ClearHighlights();

int currentX = Mathf.RoundToInt(unitPos.x / 5) \* 5;

int currentZ = Mathf.RoundToInt(unitPos.z / 5) \* 5;

Unit[] allUnits = FindObjectsOfType<Unit>();

for (int x = currentX - range \* 5; x <= currentX + range \* 5; x += 5)

{

for (int z = currentZ - range \* 5; z <= currentZ + range \* 5; z += 5)

{

if (x == currentX && z == currentZ) continue;

string tileName = $"Tile\_{x}\_{z}";

GameObject tile = GameObject.Find(tileName);

if (tile != null)

{

bool isOccupied = false;

Vector3 tileCenter = new Vector3(x, 0, z);

foreach (Unit unit in allUnits)

{

if (unit != selectedUnit &&

Vector3.Distance(unit.transform.position, tileCenter) < 2.5f)

{

isOccupied = true;

break;

}

}

if (!isOccupied)

{

tile.GetComponent<Renderer>().material = highlightMaterial;

highlightedTiles.Add(tile);

}

}

}

}

}

public void ClearHighlights()

{

foreach (var tile in highlightedTiles)

{

if (tile != null)

{

tile.GetComponent<Renderer>().material = Resources.Load<Material>("DefaultTileMaterial");

}

}

highlightedTiles.Clear();

}

}

using UnityEngine;

using UnityEngine.InputSystem;

using System.Collections;

using UnityEngine.EventSystems;

public class UnitSelector : MonoBehaviour

{

private Unit selectedUnit;

private Camera mainCamera;

[Header("Grid Settings")]

[SerializeField] private float gridSize = 5f;

[SerializeField] private float moveSpeed = 10f;

[Header("References")]

[SerializeField] private TurnManager turnManager;

[SerializeField] private TileHighlighter tileHighlighter;

[SerializeField] private ParticleSystem AttackEffect;

[SerializeField] private ActionMenu actionMenu;

private bool isSelectingTarget = false;

private bool isSuperAttack = false;

private Vector3 moveTargetPosition;

public Unit SelectedUnit => selectedUnit;

void Start()

{

mainCamera = Camera.main;

if (turnManager == null) turnManager = FindObjectOfType<TurnManager>();

if (tileHighlighter == null) tileHighlighter = FindObjectOfType<TileHighlighter>();

}

void Update()

{

HandleUnitSelection();

HandleTargetSelection();

}

public void DeselectUnit()

{

selectedUnit = null;

tileHighlighter.ClearHighlights();

isSelectingTarget = false;

isSuperAttack = false;

}

public void StartMovement(Unit unit)

{

if (unit == null || unit.owner.ToString() != turnManager.CurrentTurn.ToString())

{

Debug.LogWarning("Попытка начать движение с недействительным юнитом");

return;

}

if (selectedUnit != null && selectedUnit != unit)

{

tileHighlighter.ClearHighlights();

}

selectedUnit = unit;

tileHighlighter.HighlightTiles(unit.transform.position, unit.moveRange);

UIManager.Instance.ShowHint(TurnManager.Instance.CurrentTurn, "Выберите клетку для перемещения");

}

public void StartAttack(Unit unit)

{

selectedUnit = unit;

isSelectingTarget = true;

isSuperAttack = false;

if (unit.unitType == Unit.UnitType.Mage)

{

PerformMageAttack();

return;

}

UIManager.Instance.ShowHint(TurnManager.Instance.CurrentTurn, "Выберите цель для атаки");

}

public void StartSuperAttack(Unit unit)

{

if (!TurnManager.Instance.CanUseSuper(unit.owner))

{

UIManager.Instance.ShowHint(TurnManager.Instance.CurrentTurn, "Недостаточно зарядов!");

return;

}

selectedUnit = unit;

if (unit.unitType == Unit.UnitType.Mage)

{

PerformMageSuperAttack();

return;

}

isSelectingTarget = true;

isSuperAttack = true;

UIManager.Instance.ShowHint(TurnManager.Instance.CurrentTurn, "Выберите цель для супер-атаки");

}

private void PerformMageSuperAttack()

{

bool enemiesFound = false;

Vector3 magePos = selectedUnit.transform.position;

int centerX = Mathf.RoundToInt(magePos.x / 5) \* 5;

int centerZ = Mathf.RoundToInt(magePos.z / 5) \* 5;

for (int xOffset = -1; xOffset <= 1; xOffset++)

{

for (int zOffset = -1; zOffset <= 1; zOffset++)

{

if (xOffset == 0 && zOffset == 0) continue;

int targetX = centerX + xOffset \* 5;

int targetZ = centerZ + zOffset \* 5;

Collider[] colliders = Physics.OverlapBox(

new Vector3(targetX, 0, targetZ),

new Vector3(2.5f, 2.5f, 2.5f)

);

foreach (var collider in colliders)

{

Unit enemy = collider.GetComponent<Unit>();

if (enemy != null && enemy.owner != selectedUnit.owner)

{

Instantiate(AttackEffect, enemy.transform.position, Quaternion.identity);

enemy.TakeDamage(selectedUnit.damage \* 2);

enemiesFound = true;

}

}

}

}

if (enemiesFound)

{

TurnManager.Instance.RegisterAttack(selectedUnit.owner);

TurnManager.Instance.UseSuper(selectedUnit.owner);

TurnManager.Instance.ActionPerformed();

}

else

{

UIManager.Instance.ShowHint(TurnManager.Instance.CurrentTurn, "Нет врагов рядом!");

}

DeselectUnit();

}

private void PerformMageAttack()

{

bool enemiesFound = false;

Vector3 magePos = selectedUnit.transform.position;

int centerX = Mathf.RoundToInt(magePos.x / 5) \* 5;

int centerZ = Mathf.RoundToInt(magePos.z / 5) \* 5;

for (int xOffset = -1; xOffset <= 1; xOffset++)

{

for (int zOffset = -1; zOffset <= 1; zOffset++)

{

if (xOffset == 0 && zOffset == 0) continue;

int targetX = centerX + xOffset \* 5;

int targetZ = centerZ + zOffset \* 5;

Collider[] colliders = Physics.OverlapBox(

new Vector3(targetX, 0, targetZ),

new Vector3(2.5f, 2.5f, 2.5f)

);

foreach (var collider in colliders)

{

Unit enemy = collider.GetComponent<Unit>();

if (enemy != null && enemy.owner != selectedUnit.owner)

{

Instantiate(AttackEffect, enemy.transform.position, Quaternion.identity);

enemy.TakeDamage(selectedUnit.damage);

enemiesFound = true;

}

}

}

}

if (enemiesFound)

{

TurnManager.Instance.RegisterAttack(selectedUnit.owner);

TurnManager.Instance.ActionPerformed();

}

else

{

UIManager.Instance.ShowHint(TurnManager.Instance.CurrentTurn, "Нет врагов рядом!");

}

DeselectUnit();

}

private void HandleUnitSelection()

{

if (EventSystem.current.IsPointerOverGameObject()) return;

if (isSelectingTarget) return;

if (Mouse.current.leftButton.wasPressedThisFrame)

{

Ray ray = mainCamera.ScreenPointToRay(Mouse.current.position.ReadValue());

if (Physics.Raycast(ray, out RaycastHit hit))

{

Unit unit = hit.collider.GetComponent<Unit>();

if (unit != null && unit.owner.ToString() == turnManager.CurrentTurn.ToString())

{

selectedUnit = unit;

actionMenu.ShowMenu(unit, Mouse.current.position.ReadValue());

}

else

{

actionMenu.HideMenu();

if (selectedUnit != null && hit.collider.CompareTag("Tile"))

{

moveTargetPosition = GetTileCenterPosition(hit.point);

int distance = CalculateGridDistance(selectedUnit.transform.position, moveTargetPosition);

if (distance <= selectedUnit.moveRange)

{

StartCoroutine(MoveUnitCoroutine(moveTargetPosition));

TurnManager.Instance.ActionPerformed();

}

else

{

UIManager.Instance.ShowHint(TurnManager.Instance.CurrentTurn, "Слишком далеко!");

}

}

}

}

else

{

actionMenu.HideMenu();

}

}

}

private void HandleTargetSelection()

{

if (!isSelectingTarget) return;

if (Mouse.current.leftButton.wasPressedThisFrame)

{

if (EventSystem.current.IsPointerOverGameObject()) return;

Ray ray = mainCamera.ScreenPointToRay(Mouse.current.position.ReadValue());

if (Physics.Raycast(ray, out RaycastHit hit))

{

Unit clickedUnit = hit.collider.GetComponent<Unit>();

if (clickedUnit != null && clickedUnit.owner == selectedUnit.owner)

{

actionMenu.ShowMenu(clickedUnit, Mouse.current.position.ReadValue());

isSelectingTarget = false;

isSuperAttack = false;

return;

}

Unit target = hit.collider.GetComponent<Unit>();

if (target != null && target.owner != selectedUnit.owner)

{

if (selectedUnit.CanAttack(target.transform.position))

{

if (isSuperAttack)

PerformSuperAttack(target);

else

PerformRegularAttack(target);

}

else

{

UIManager.Instance.ShowHint(TurnManager.Instance.CurrentTurn, "Цель вне досягаемости!");

}

}

}

}

}

private void PerformRegularAttack(Unit target)

{

switch (selectedUnit.unitType)

{

case Unit.UnitType.Archer:

case Unit.UnitType.Warrior:

target.TakeDamage(selectedUnit.damage);

break;

}

Instantiate(AttackEffect, target.transform.position, Quaternion.identity);

TurnManager.Instance.RegisterAttack(selectedUnit.owner);

TurnManager.Instance.ActionPerformed();

DeselectUnit();

}

private void PerformSuperAttack(Unit target)

{

int superDamage = selectedUnit.damage \* 2;

target.TakeDamage(superDamage);

Instantiate(AttackEffect, target.transform.position, Quaternion.identity);

TurnManager.Instance.UseSuper(selectedUnit.owner);

TurnManager.Instance.ActionPerformed();

DeselectUnit();

}

private Vector3 GetTileCenterPosition(Vector3 worldPosition)

{

return new Vector3(

Mathf.Round(worldPosition.x / gridSize) \* gridSize,

0,

Mathf.Round(worldPosition.z / gridSize) \* gridSize

);

}

private int CalculateGridDistance(Vector3 from, Vector3 to)

{

int fromX = Mathf.RoundToInt(from.x / gridSize);

int fromZ = Mathf.RoundToInt(from.z / gridSize);

int toX = Mathf.RoundToInt(to.x / gridSize);

int toZ = Mathf.RoundToInt(to.z / gridSize);

return Mathf.Max(

Mathf.Abs(fromX - toX),

Mathf.Abs(fromZ - toZ)

);

}

private IEnumerator MoveUnitCoroutine(Vector3 targetPos)

{

Unit unit = selectedUnit;

if (unit == null)

yield break;

Transform unitTransform = unit.transform;

Vector3 startPos = unitTransform.position;

targetPos.y = startPos.y;

while (Vector3.Distance(unitTransform.position, targetPos) > 0.1f)

{

if (unit == null)

yield break;

unitTransform.position = Vector3.MoveTowards(

unitTransform.position,

targetPos,

moveSpeed \* Time.deltaTime

);

yield return null;

}

if (unit != null)

{

unitTransform.position = targetPos;

tileHighlighter.ClearHighlights();

}

}

}

**Приложение 2.**

Ссылка на файл программы с ассетами: <https://github.com/Voronova1/practice>