Отчет о выполнении практической работы по ДПП ПП

«Основы Gamedev и VR-разработки»

**«Кейс 1»**

Исполнители:  
Ваверко Илья Сергеевич

Введение

Проект представляет собой пошаговую тактическую систему, где игрок управляет юнитами, выполняя действия (перемещение, атака, специальные способности) с помощью карт. Основной геймплей строится на очереди ходов: юниты могут накапливать очки для мощных "супер-ударов", а карты дают случайные эффекты (лечение, усиление атаки или защиты). Управление реализовано через интерактивный выбор целей (мышь/клавиши), с визуализацией пути и зон атаки.

Система разделена на логические модули: Unit (данные юнита), Model (здоровье и действия), Controller (ввод игрока), Card (эффекты) и SuperHitBar (супер-удары). Дальше будут представлены скрипты с исчерпывающем пояснением.

Пошаговая система

using System.Collections;

using System.Collections.Generic;

using System;

using UnityEngine;

/// <summary>

/// Система управления раундами и очередью ходов в игре.

/// Отслеживает состояние боя, управляет сменой раундов и определяет победителя.

/// </summary>

public class RoundSystem : MonoBehaviour

{

/// <summary>

/// Singleton экземпляр системы раундов

/// </summary>

public static RoundSystem Instance { get; private set; }

#region SystemFields

// Статические поля для хранения состояния системы

private static int \_round; // Текущий раунд

private static int \_roundSteps; // Количество шагов в раунде

private static int \_currentStep = 0; // Текущий шаг в раунде

private static int \_endRound; // Финальный раунд

private static bool \_isEnemyAction; // Флаг очереди хода (true - враги, false - союзники)

private static int WinnerStatus; // Статус победителя (0 - союзники, 1 - враги, 2 - ничья)

// Группы юнитов

private static List<GameObject> enemyGroup = new List<GameObject>();

private static List<GameObject> allienGroup = new List<GameObject>();

#endregion

#region InspectorFields

// Настраиваемые параметры в инспекторе

[SerializeField, Tooltip("Начальный раунд")]

private int round = 1;

[SerializeField, Tooltip("Количество шагов в раунде")]

private int roundSteps = 2;

[SerializeField, Tooltip("Финальный раунд")]

private int endRound = 5;

#endregion

#region Events

// События системы

/// <summary>Событие окончания игры</summary>

public static Action onEndGame;

/// <summary>Событие изменения очереди хода</summary>

public static Action onChangedQueue;

/// <summary>Событие изменения раунда</summary>

public static Action onChangedRound;

/// <summary>Событие инициализации системы</summary>

public static Action onInitSystem;

/// <summary>Событие начала действий юнитов</summary>

public static Action OnStartAction;

#endregion

#region Properties

/// <summary>

/// Текущая очередь хода (true - враги, false - союзники)

/// </summary>

public static bool GetQueue

{

get { return \_isEnemyAction; }

}

/// <summary>

/// Текущий номер раунда

/// </summary>

public static int GetRound

{

get { return \_round; }

}

/// <summary>

/// Номер финального раунда

/// </summary>

public static int GetEndRound

{

get { return \_endRound; }

}

/// <summary>

/// Статус победителя (0 - союзники, 1 - враги, 2 - ничья)

/// </summary>

public static int GetWinner

{

get { return WinnerStatus; }

}

/// <summary>

/// Инициализирует статические поля значениями из инспектора

/// </summary>

private void initInspectorFields()

{

\_round = round;

\_roundSteps = roundSteps;

\_endRound = endRound;

}

/// <summary>

/// Список союзных юнитов

/// </summary>

public static List<GameObject> GetAllienGroup

{

get { return allienGroup; }

}

/// <summary>

/// Список вражеских юнитов

/// </summary>

public static List<GameObject> GetEnemyGroup

{

get { return enemyGroup; }

}

#endregion

private void Awake()

{

if (Instance == null)

{

Instance = this;

initInspectorFields();

DontDestroyOnLoad(gameObject);

RegisterExistingUnits();

onInitSystem?.Invoke();

}

else

{

Destroy(gameObject);

return;

}

}

private void Start()

{

Unit.onInitUnit += UpdateLists;

}

#region PrivateMethods

/// <summary>

/// Вычисляет суммарное HP группы юнитов

/// </summary>

/// <param name="Units">Список юнитов</param>

/// <returns>Суммарное HP группы</returns>

private float GetHP(List<GameObject> Units)

{

float HP = 0;

foreach (var unit in Units)

{

HP += unit.GetComponent<Model>().GetHP;

}

return HP;

}

/// <summary>

/// Добавляет союзного юнита в соответствующую группу

/// </summary>

/// <param name="allien">Объект союзного юнита</param>

private void UpdateAllienGroup(GameObject allien)

{

allienGroup.Add(allien);

}

/// <summary>

/// Добавляет вражеского юнита в соответствующую группу

/// </summary>

/// <param name="enemy">Объект вражеского юнита</param>

private void UpdateEnemyGroup(GameObject enemy)

{

enemyGroup.Add(enemy);

}

/// <summary>

/// Определяет победителя на основе оставшегося HP

/// </summary>

private void SetWinner()

{

var AlienHP = GetHP(allienGroup);

var EnemyHP = GetHP(enemyGroup);

Debug.Log(AlienHP);

Debug.Log(EnemyHP);

if (AlienHP > EnemyHP) WinnerStatus = 0;

else if (AlienHP < EnemyHP) WinnerStatus = 1;

else WinnerStatus = 2;

}

/// <summary>

/// Обновляет списки юнитов при инициализации нового юнита

/// </summary>

/// <param name="unit">Объект юнита</param>

/// <param name="isEnemy">Принадлежность к вражеской группе</param>

private void UpdateLists(GameObject unit, bool isEnemy)

{

OnStartAction += unit.GetComponent<Unit>().StartUnitActions;

onChangedQueue += unit.GetComponent<Unit>().QueueCheck;

if (isEnemy) UpdateEnemyGroup(unit);

else UpdateAllienGroup(unit);

Debug.Log($"Added {(isEnemy ? "Enemy" : "Allien")}: {unit.name}");

}

private void Update()

{

Debug.Log(allienGroup.Count);

}

/// <summary>

/// Регистрирует все существующие юниты на сцене

/// </summary>

private void RegisterExistingUnits()

{

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

foreach (Unit unit in units)

{

UpdateLists(unit.gameObject, unit.IsEnemy);

}

}

/// <summary>

/// Переход к следующему раунду

/// </summary>

private void ChangeRound()

{

if (\_round == \_endRound)

{

SetWinner();

onEndGame?.Invoke();

return;

}

\_round++;

}

/// <summary>

/// Переход к следующему шагу в раунде

/// </summary>

private void Step()

{

ChangeQueue();

\_currentStep++;

if (\_currentStep % \_roundSteps == 0)

{

ChangeRound();

\_currentStep = 0;

}

}

/// <summary>

/// Смена очереди хода между командами

/// </summary>

private void ChangeQueue()

{

\_isEnemyAction = !\_isEnemyAction;

}

/// <summary>

/// Запускает действия юнитов в текущем шаге

/// </summary>

public void StartActions()

{

Step();

OnStartAction?.Invoke();

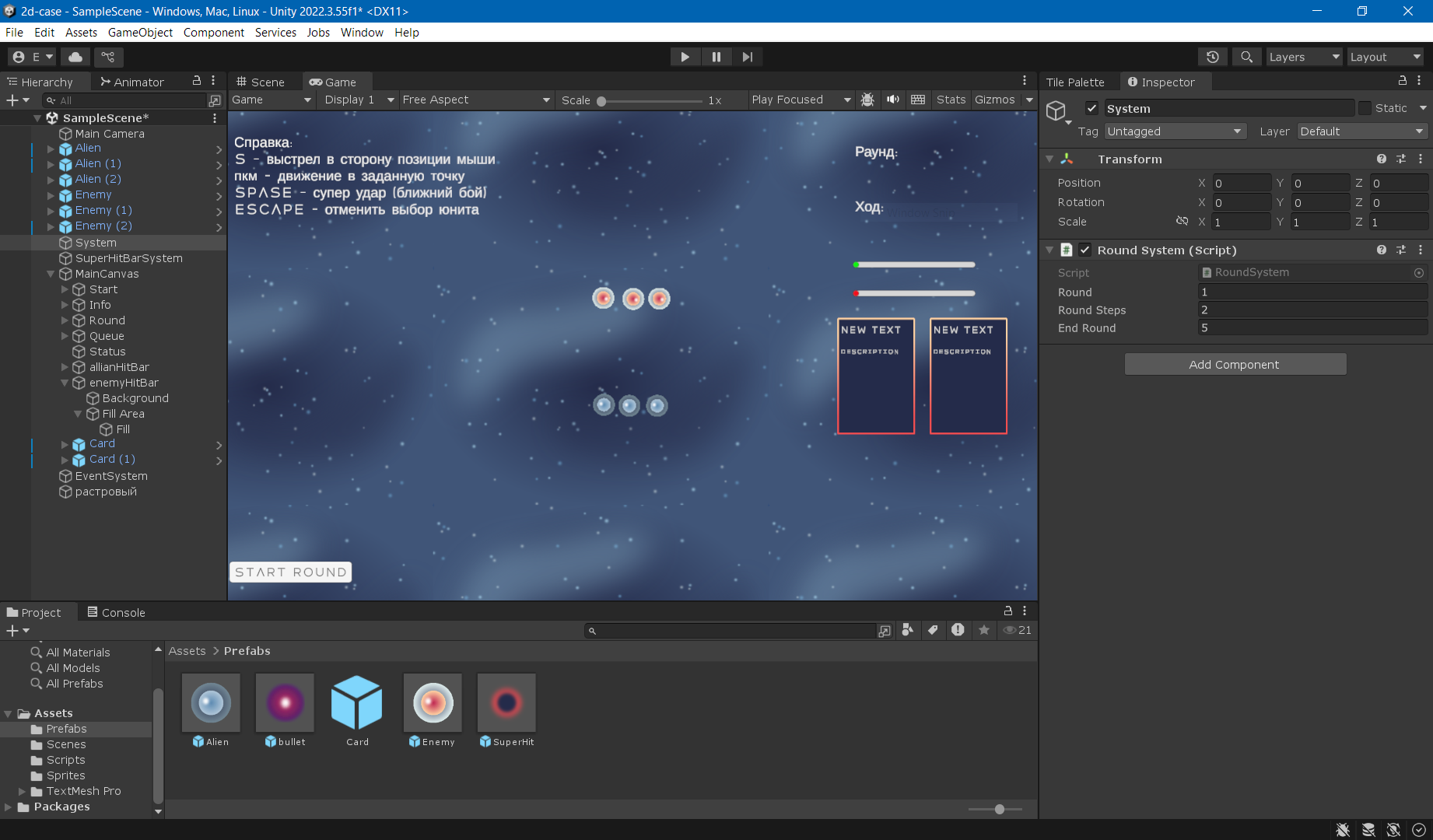
onChangedRound?.Invoke();

onChangedQueue?.Invoke();

}

#endregion

}



Юнит

using UnityEngine;

using UnityEngine.Events;

using System;

/// <summary>

/// Класс, представляющий базовую единицу (юнит) в игре.

/// Управляет состоянием юнита, его действиями и взаимодействием с системой очереди ходов.

/// </summary>

public class Unit : MonoBehaviour

{

[SerializeField] private bool \_isEnemy = false; // Флаг, определяющий принадлежность юнита к вражеской команде

[SerializeField] private uint ActionCounts = 3; // Количество доступных действий за ход

[SerializeField] private Controller Controller; // Ссылка на контроллер управления юнитом

[SerializeField] private UnityEvent onUnitActionsStart; // Событие, вызываемое при начале действий юнита

/// <summary>

/// Принадлежность юнита к вражеской команде

/// </summary>

public bool IsEnemy

{

get { return \_isEnemy; }

}

/// <summary>

/// Статическое событие инициализации юнита

/// Параметры:

/// GameObject - ссылка на объект юнита

/// bool - принадлежность к вражеской команде

/// </summary>

public static Action<GameObject, bool> onInitUnit;

private void Awake()

{

Init();

QueueCheck();

}

/// <summary>

/// Уменьшает количество доступных действий юнита.

/// Если действия закончились - отключает контроллер и сбрасывает счетчик.

/// </summary>

public void ChangeFreeActions()

{

ActionCounts--;

if (ActionCounts == 0)

{

Controller.enabled = false;

ActionCounts = 3;

}

Debug.Log($"Name:{gameObject.name} actions free {ActionCounts}");

}

/// <summary>

/// Проверяет, совпадает ли текущая очередь хода с принадлежностью юнита.

/// Включает/выключает контроллер в зависимости от результата проверки.

/// </summary>

public void QueueCheck()

{

Debug.Log($"CheckQueue:{gameObject.name}");

if (RoundSystem.GetQueue == \_isEnemy)

{

Controller.enabled = true;

}

else

{

Controller.enabled = false;

}

}

/// <summary>

/// Инициализирует юнит, вызывая событие onInitUnit

/// </summary>

private void Init()

{

onInitUnit?.Invoke(gameObject, \_isEnemy);

Debug.Log($"Initiation {gameObject.name}");

}

/// <summary>

/// Запускает действия юнита, переключая состояние контроллера и вызывая событие onUnitActionsStart

/// </summary>

public void StartUnitActions()

{

Controller.ChangeControllerStatus();

onUnitActionsStart?.Invoke();

Controller.ChangeControllerStatus();

}

private void OnDisable()

{

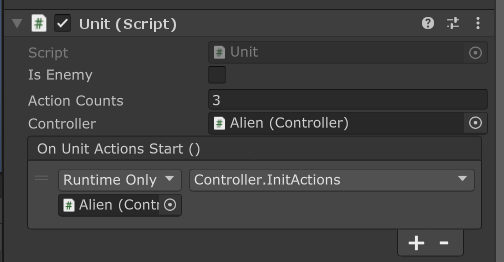
// Отписываемся от событий при отключении объекта

RoundSystem.onChangedQueue -= QueueCheck;

RoundSystem.OnStartAction -= StartUnitActions;

}

}



Модель

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

/// <summary>

/// Основной класс модели юнита, управляющий его характеристиками и поведением

/// </summary>

public class Model : MonoBehaviour

{

// Основные характеристики юнита

[SerializeField] float Health = 200; // Текущее здоровье юнита

[SerializeField] float Speed = 15; // Скорость перемещения

[SerializeField] float Defence = 20; // Защита (уменьшает получаемый урон)

[SerializeField] float AttackForce = 15; // Сила атаки

[SerializeField] float RotationSpeed = 10f; // Скорость поворота

// Ссылки на трансформы и компоненты

[SerializeField] Transform ShootFrom; // Точка выстрела

[SerializeField] Transform CompTransf; // Ссылка на трансформ объекта

[SerializeField] SuperHitBar hitBar; // Ссылка на шкалу супер-удара

// Префабы

[SerializeField] GameObject SuperHitPref; // Префаб области супер-удара

[SerializeField] GameObject Bullet; // Префаб пули

// Флаги состояний

bool isActionsStart = false; // Флаг разрешения действий

bool isSuperHitStart = false; // Флаг активации супер-удара

bool isShoot = false; // Флаг необходимости выстрела

bool isMoving = false; // Флаг движения

bool \_isActionComplete = true; // Флаг завершения текущего действия

// Целевые позиции

Vector2 TargetPosition; // Целевая позиция для движения

Vector2 BulletTargetPosition; // Целевая позиция для выстрела

/// <summary>

/// Текущее здоровье юнита (не может быть меньше 0)

/// </summary>

public float GetHP

{

get

{

if (Health > 0) return Health;

else return 0;

}

}

/// <summary>

/// Восстанавливает здоровье юнита

/// </summary>

/// <param name="count">Количество восстанавливаемого здоровья</param>

public void Heal(float count)

{

Health += count;

}

/// <summary>

/// Модифицирует защиту юнита

/// </summary>

/// <param name="count">Значение изменения защиты</param>

public void ModifyDefense(float count)

{

Defence += count;

}

/// <summary>

/// Модифицирует силу атаки юнита

/// </summary>

/// <param name="count">Значение изменения силы атаки</param>

public void ModifyAttack(float count)

{

AttackForce += count;

}

/// <summary>

/// Наносит урон юниту с учетом защиты

/// </summary>

/// <param name="Damage">Исходный урон</param>

public void setDamage(float Damage)

{

Health = Health - (Damage - Defence);

SuperHitBar.AddPoint(gameObject.GetComponent<Unit>().IsEnemy);

}

private void Awake()

{

CompTransf = GetComponent<Transform>();

}

/// <summary>

/// Изменяет статус готовности юнита

/// </summary>

private void ChangeReadyStatus()

{

\_isActionComplete = !\_isActionComplete;

}

/// <summary>

/// Статус готовности юнита к новым действиям

/// </summary>

public bool GetReadyStatus

{

get { return \_isActionComplete; }

}

/// <summary>

/// Устанавливает целевую позицию для движения

/// </summary>

/// <param name="newTarget">Новая целевая позиция</param>

public void SetTarget(Vector2 newTarget)

{

if (TargetPosition == newTarget) return;

TargetPosition = newTarget;

isMoving = true;

}

/// <summary>

/// Устанавливает целевую позицию для выстрела

/// </summary>

/// <param name="newTarget">Новая целевая позиция</param>

public void SetBulletTarget(Vector2 newTarget)

{

BulletTargetPosition = newTarget;

isShoot = true;

}

/// <summary>

/// Активирует супер-удар

/// </summary>

public void SetSuperHit()

{

isSuperHitStart = true;

}

/// <summary>

/// Разрешает выполнение действий

/// </summary>

public void StartActions()

{

isActionsStart = true;

}

/// <summary>

/// Запрещает выполнение действий

/// </summary>

public void StopActions()

{

isActionsStart = false;

}

/// <summary>

/// Поворачивает юнита к целевой позиции

/// </summary>

/// <param name="targetPosition">Целевая позиция</param>

private void RotateTowardsTarget(Vector2 targetPosition)

{

Vector2 direction = targetPosition - (Vector2)CompTransf.position;

float angle = Mathf.Atan2(direction.y, direction.x) \* Mathf.Rad2Deg;

Quaternion targetRotation = Quaternion.AngleAxis(angle, Vector3.forward);

CompTransf.rotation = Quaternion.Slerp(

CompTransf.rotation,

targetRotation,

RotationSpeed \* Time.deltaTime);

}

/// <summary>

/// Осуществляет шаг движения к цели

/// </summary>

private void MovementStep()

{

if (!isMoving) return;

\_isActionComplete = false;

Vector2 toTarget = TargetPosition - (Vector2)CompTransf.position;

float distance = toTarget.magnitude;

RotateTowardsTarget(TargetPosition);

if (distance <= 1)

{

isMoving = false;

ChangeReadyStatus();

return;

}

Vector2 currentDirection = toTarget.normalized;

CompTransf.position += (Vector3)(currentDirection \* Speed \* Time.deltaTime);

}

/// <summary>

/// Выполняет выстрел

/// </summary>

private void Shooting()

{

ChangeReadyStatus();

if (isShoot)

{

var bullet = Instantiate(Bullet, ShootFrom);

var bullet\_conf = bullet.GetComponent<bullet>();

bullet\_conf.AttackForce = AttackForce;

bullet\_conf.isEnemy = gameObject.GetComponent<Unit>().IsEnemy;

bullet.GetComponent<Rigidbody2D>().AddForce(

(BulletTargetPosition - (Vector2)ShootFrom.position).normalized \* 25,

ForceMode2D.Impulse);

Destroy(bullet, 30f);

isShoot = false;

}

ChangeReadyStatus();

}

/// <summary>

/// Активирует супер-удар

/// </summary>

private void SuperHit()

{

ChangeReadyStatus();

if (isSuperHitStart)

{

var superHitArea = Instantiate(SuperHitPref, CompTransf.position, CompTransf.rotation);

Destroy(superHitArea, 5f);

isSuperHitStart = false;

}

ChangeReadyStatus();

}

/// <summary>

/// Обрабатывает смерть юнита

/// </summary>

public void Death()

{

if (Health <= 0)

{

gameObject.SetActive(false);

}

}

private void Update()

{

if (isActionsStart)

{

MovementStep();

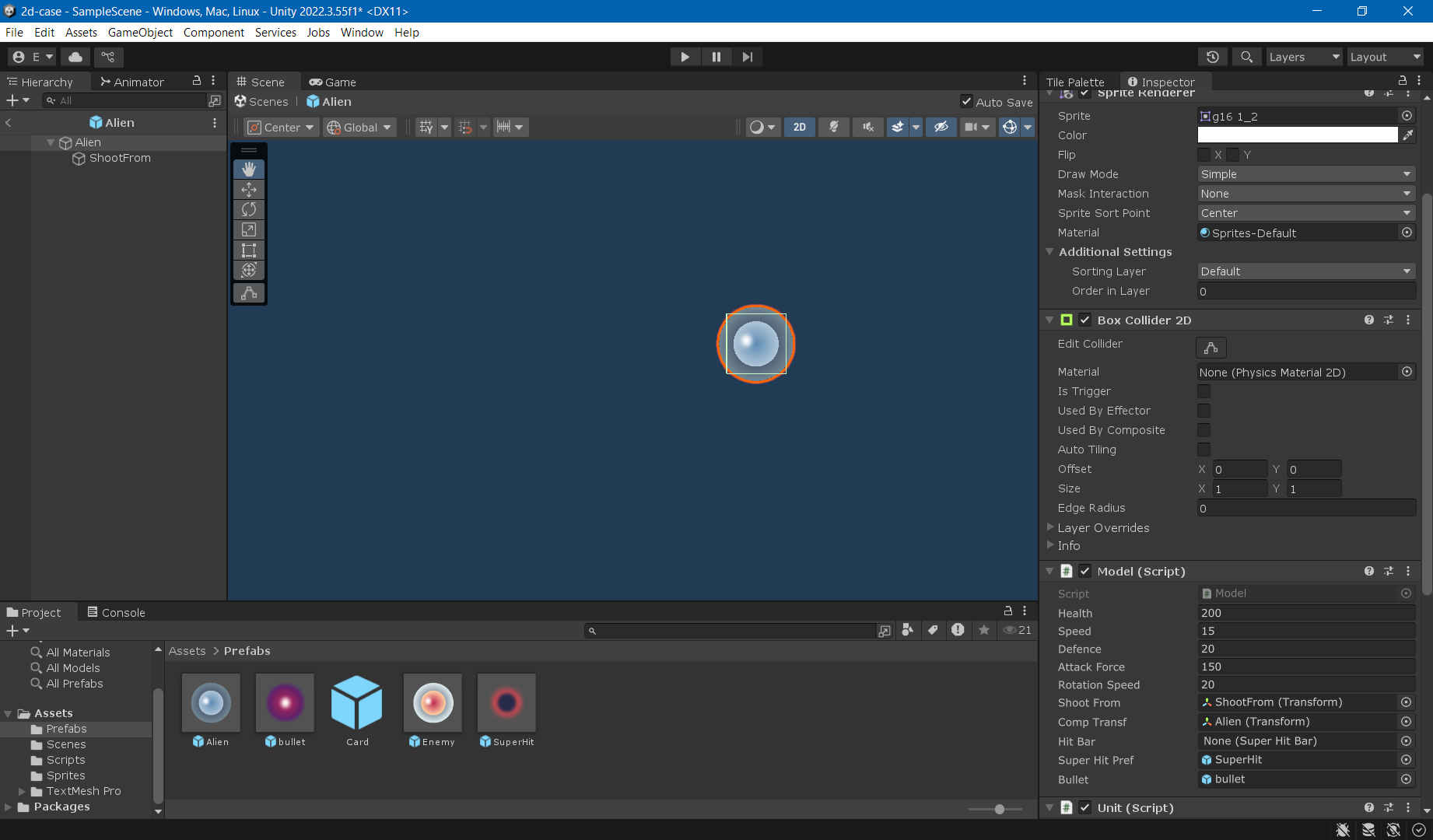
Shooting();

SuperHit();

Death();

}

}

}  


Контроллер

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

using System;

/// <summary>

/// Контроллер управления юнитом, обрабатывающий ввод игрока и управляющий действиями юнита.

/// Реализует систему очереди действий, визуализацию пути и специальные способности.

/// </summary>

public class Controller : MonoBehaviour

{

// Состояния контроллера

private bool isChosen = false; // Флаг выбора юнита

private bool isOff = false; // Флаг отключения контроллера

// Ссылки на компоненты

[SerializeField] private Model myModel; // Модель юнита

[SerializeField] private Unit myUnit; // Данные юнита

private SuperHitBar hitbar; // Шкала супер-удара

// Система действий

private List<Action> actions = new List<Action>(); // Очередь действий юнита

// Визуализация пути

private List<Vector2> pathPoints = new List<Vector2>(); // Точки пути

private GameObject pathLineObject; // Объект линии пути

private LineRenderer pathLineRenderer; // Рендерер линии пути

// События

Action actionChange; // Событие изменения действия

Action onEndActions; // Событие завершения всех действий

private void Awake()

{

// Инициализация компонентов

myModel = GetComponent<Model>();

myUnit = GetComponent<Unit>();

// Настройка события изменения действия

actionChange = () => myUnit.ChangeFreeActions();

// Создание визуализации пути

pathLineObject = new GameObject("PathLine");

pathLineRenderer = pathLineObject.AddComponent<LineRenderer>();

pathLineRenderer.startWidth = 0.1f;

pathLineRenderer.endWidth = 0.1f;

pathLineRenderer.material = new Material(Shader.Find("Sprites/Default"));

pathLineRenderer.startColor = Color.cyan;

pathLineRenderer.endColor = Color.cyan;

pathLineObject.SetActive(false);

}

/// <summary>

/// Обрабатывает клик по юниту, выбирая его и начина визуализацию пути

/// </summary>

private void OnMouseDown()

{

isChosen = true;

ClearPath();

pathPoints.Add(transform.position);

UpdatePathLine();

pathLineObject.SetActive(true);

}

/// <summary>

/// Переключает состояние активности контроллера

/// </summary>

public void ChangeControllerStatus()

{

isOff = !isOff;

}

private void Update()

{

if (isChosen && !isOff)

{

// Обработка движения (ПКМ)

if (Input.GetMouseButtonDown(1))

{

Vector2 touch = Camera.main.ScreenToWorldPoint(Input.mousePosition);

actions.Add(() => myModel.SetTarget(touch));

pathPoints.Add(touch);

UpdatePathLine();

actionChange?.Invoke();

}

// Обработка выстрела (клавиша S)

if (Input.GetKeyDown(KeyCode.S))

{

Vector2 touch = Camera.main.ScreenToWorldPoint(Input.mousePosition);

actions.Add(() => myModel.SetBulletTarget(touch));

DrawTemporaryLine(touch, Color.red, 1f);

actionChange?.Invoke();

}

// Обработка супер-удара (клавиша Space)

if (Input.GetKeyDown(KeyCode.Space) && SuperHitBar.IsHavePoints(myUnit.IsEnemy))

{

actions.Add(() => myModel.SetSuperHit());

DrawCircleAroundObject(1f, Color.yellow, 1f);

actionChange?.Invoke();

}

// Отмена выбора (клавиша Escape)

if (Input.GetKeyDown(KeyCode.Escape))

{

isChosen = false;

ClearPath();

}

}

}

/// <summary>

/// Обновляет визуализацию пути

/// </summary>

private void UpdatePathLine()

{

pathLineRenderer.positionCount = pathPoints.Count;

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

{

pathLineRenderer.SetPosition(i, pathPoints[i]);

}

}

/// <summary>

/// Рисует временную линию к цели

/// </summary>

/// <param name="target">Конечная точка линии</param>

/// <param name="color">Цвет линии</param>

/// <param name="duration">Время отображения в секундах</param>

private void DrawTemporaryLine(Vector2 target, Color color, float duration)

{

GameObject lineObj = new GameObject("TempLine");

LineRenderer lr = lineObj.AddComponent<LineRenderer>();

lr.startWidth = 0.1f;

lr.endWidth = 0.1f;

lr.positionCount = 2;

lr.material = new Material(Shader.Find("Sprites/Default"));

lr.startColor = color;

lr.endColor = color;

lr.SetPosition(0, transform.position);

lr.SetPosition(1, target);

Destroy(lineObj, duration);

}

/// <summary>

/// Рисует временную окружность вокруг объекта

/// </summary>

/// <param name="radius">Радиус окружности</param>

/// <param name="color">Цвет окружности</param>

/// <param name="duration">Время отображения в секундах</param>

private void DrawCircleAroundObject(float radius, Color color, float duration)

{

GameObject circleObj = new GameObject("TempCircle");

LineRenderer lr = circleObj.AddComponent<LineRenderer>();

lr.startWidth = 0.1f;

lr.endWidth = 0.1f;

lr.material = new Material(Shader.Find("Sprites/Default"));

lr.startColor = color;

lr.endColor = color;

int segments = 36;

lr.positionCount = segments + 1;

Vector3[] points = new Vector3[segments + 1];

for (int i = 0; i < segments + 1; i++)

{

float angle = Mathf.Deg2Rad \* (i \* 360f / segments);

points[i] = transform.position + new Vector3(Mathf.Sin(angle) \* radius, Mathf.Cos(angle) \* radius, 0);

}

lr.SetPositions(points);

Destroy(circleObj, duration);

}

/// <summary>

/// Очищает текущий путь и его визуализацию

/// </summary>

private void ClearPath()

{

pathPoints.Clear();

pathLineRenderer.positionCount = 0;

pathLineObject.SetActive(false);

}

/// <summary>

/// Корутина выполнения очереди действий

/// </summary>

private IEnumerator InitActionsCoroutine()

{

bool exit = false;

int i = 0;

while (!exit && actions.Count != 0)

{

if (myModel.GetReadyStatus)

{

actions[i]?.Invoke();

i++;

if (i == actions.Count)

{

exit = true;

onEndActions?.Invoke();

isChosen = false;

actions.Clear();

ClearPath();

}

}

yield return new WaitForSeconds(1);

}

}

/// <summary>

/// Запускает выполнение очереди действий

/// </summary>

public void InitActions()

{

myModel.StartActions();

StartCoroutine(InitActionsCoroutine());

}

private void OnDisable()

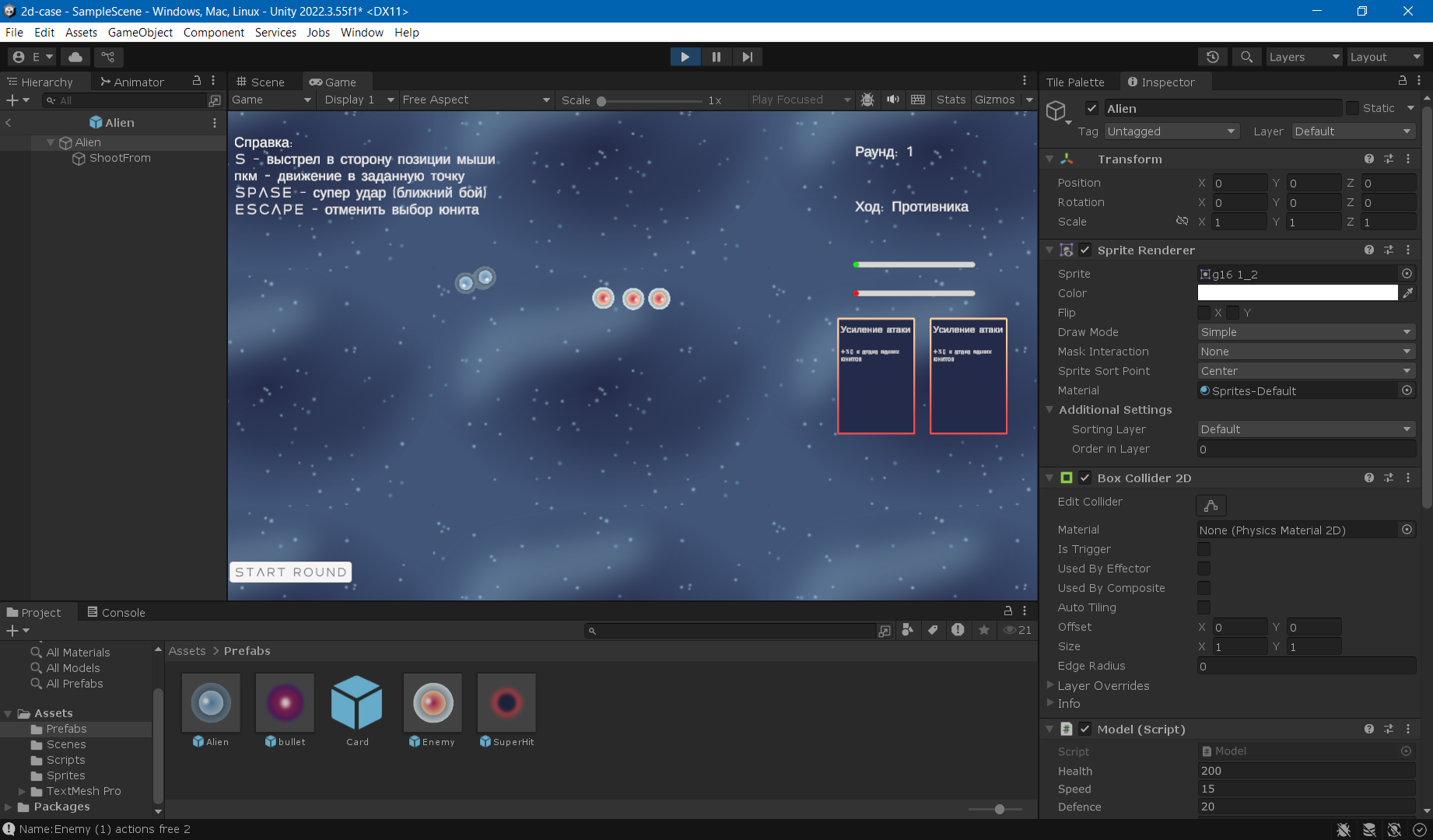
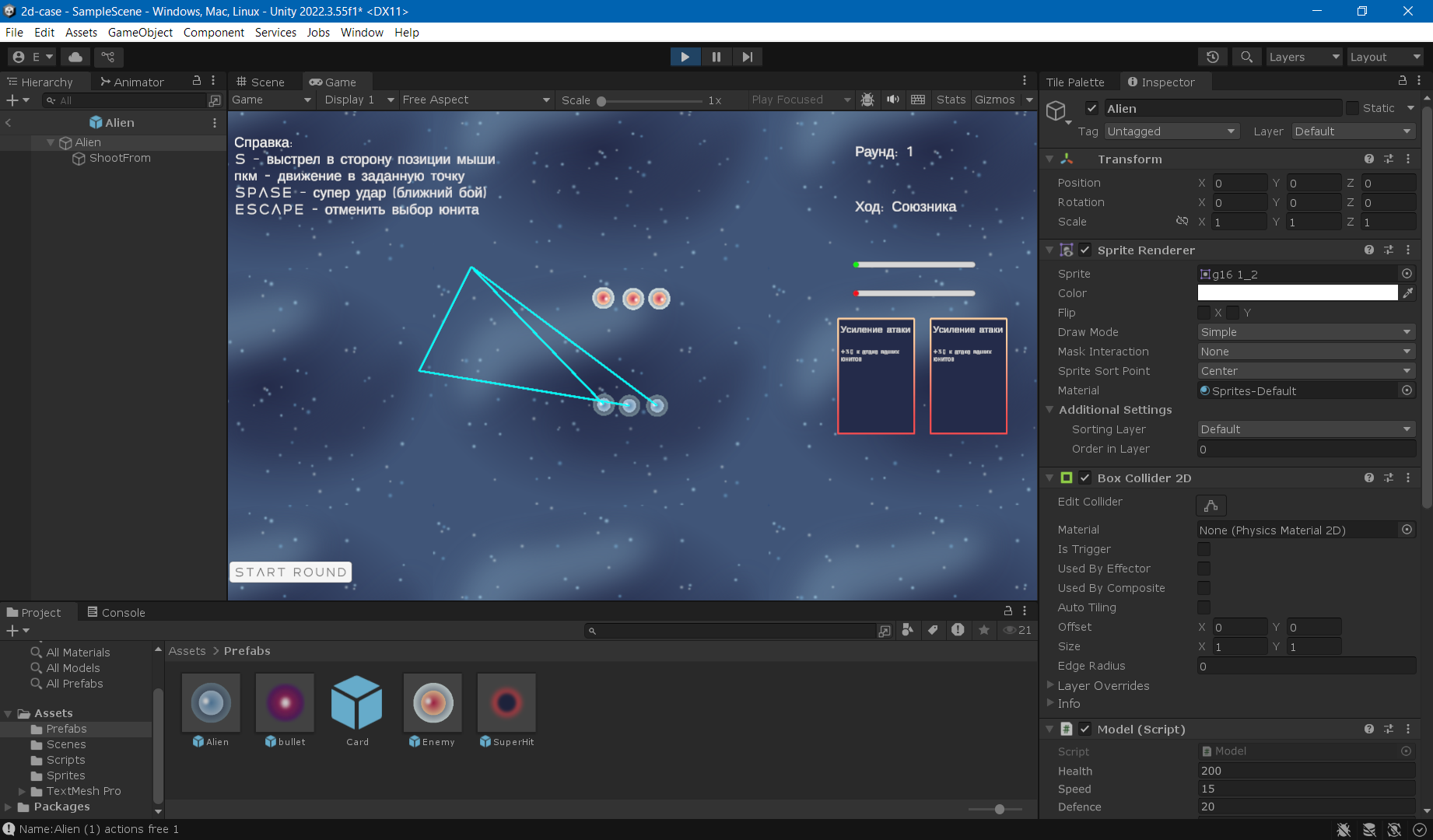
{

// Отписываемся от событий при отключении

actionChange -= myUnit.ChangeFreeActions;

ClearPath();

}

} 

Система суперудара  
using System.Collections;

using System.Collections.Generic;

using UnityEngine;

using System;

/// <summary>

/// Класс для управления системой накопления очков "Super Hit".

/// Позволяет накапливать очки для двух сторон (враг/союзник) и определять,

/// когда накоплено достаточно очков для специального действия.

/// </summary>

public class SuperHitBar : MonoBehaviour

{

// Статические поля для хранения значений очков

private static int \_enemyPoints = 0; // Текущие очки врага

private static int \_allianPoints = 0; // Текущие очки союзника

private static int \_maxPoints = 100; // Максимальное количество очков для специального действия

private static int \_step = 25; // Шаг приращения очков

// Сериализуемые поля для настройки в инспекторе Unity

[SerializeField] private int inspectorEnemyPoints = 0; // Начальные очки врага (настраивается в инспекторе)

[SerializeField] private int inspectorAllianPoints = 0; // Начальные очки союзника (настраивается в инспекторе)

[SerializeField] private int inspectorMaxPoints = 100; // Макс. очки (настраивается в инспекторе)

[SerializeField] private int inspectorStep = 25; // Шаг приращения (настраивается в инспекторе)

/// <summary>

/// Событие, вызываемое при изменении количества очков любой из сторон

/// </summary>

public static Action onChangedPoints;

/// <summary>

/// Возвращает текущий шаг приращения очков

/// </summary>

public static int GetStep

{

get { return \_step; }

}

/// <summary>

/// Получает текущее количество очков для указанной стороны

/// </summary>

/// <param name="IsEnemy">Если true - возвращает очки врага, иначе - очки союзника</param>

/// <returns>Текущее количество очков</returns>

public static int GetPoints(bool IsEnemy)

{

if (IsEnemy) return \_enemyPoints;

return \_allianPoints;

}

/// <summary>

/// Возвращает максимальное количество очков, необходимое для специального действия

/// </summary>

public static int GetMaxPoints

{

get { return \_maxPoints; }

}

private void Awake()

{

// Инициализация статических полей значениями из инспектора

\_enemyPoints = inspectorEnemyPoints;

\_allianPoints = inspectorAllianPoints;

\_maxPoints = inspectorMaxPoints;

\_step = inspectorStep;

}

/// <summary>

/// Добавляет очки указанной стороне (обратите внимание на инвертированную логику)

/// </summary>

/// <param name="IsEnemy">Если true - добавляет очки союзнику, иначе - врагу</param>

public static void AddPoint(bool IsEnemy)

{

if (IsEnemy) \_allianPoints += \_step;

else \_enemyPoints += \_step;

onChangedPoints?.Invoke();

}

/// <summary>

/// Проверяет, накоплено ли достаточно очков для специального действия.

/// Если да - снимает необходимое количество очков и возвращает true.

/// </summary>

/// <param name="IsEnemy">Если true - проверяет очки врага, иначе - союзника</param>

/// <returns>True, если очков достаточно для специального действия</returns>

public static bool IsHavePoints(bool IsEnemy)

{

if (IsEnemy)

{

if (\_enemyPoints >= \_maxPoints)

{

\_enemyPoints -= \_maxPoints;

onChangedPoints?.Invoke();

return true;

}

}

else

{

if (\_allianPoints >= \_maxPoints)

{

\_allianPoints -= \_maxPoints;

onChangedPoints?.Invoke();

return true;

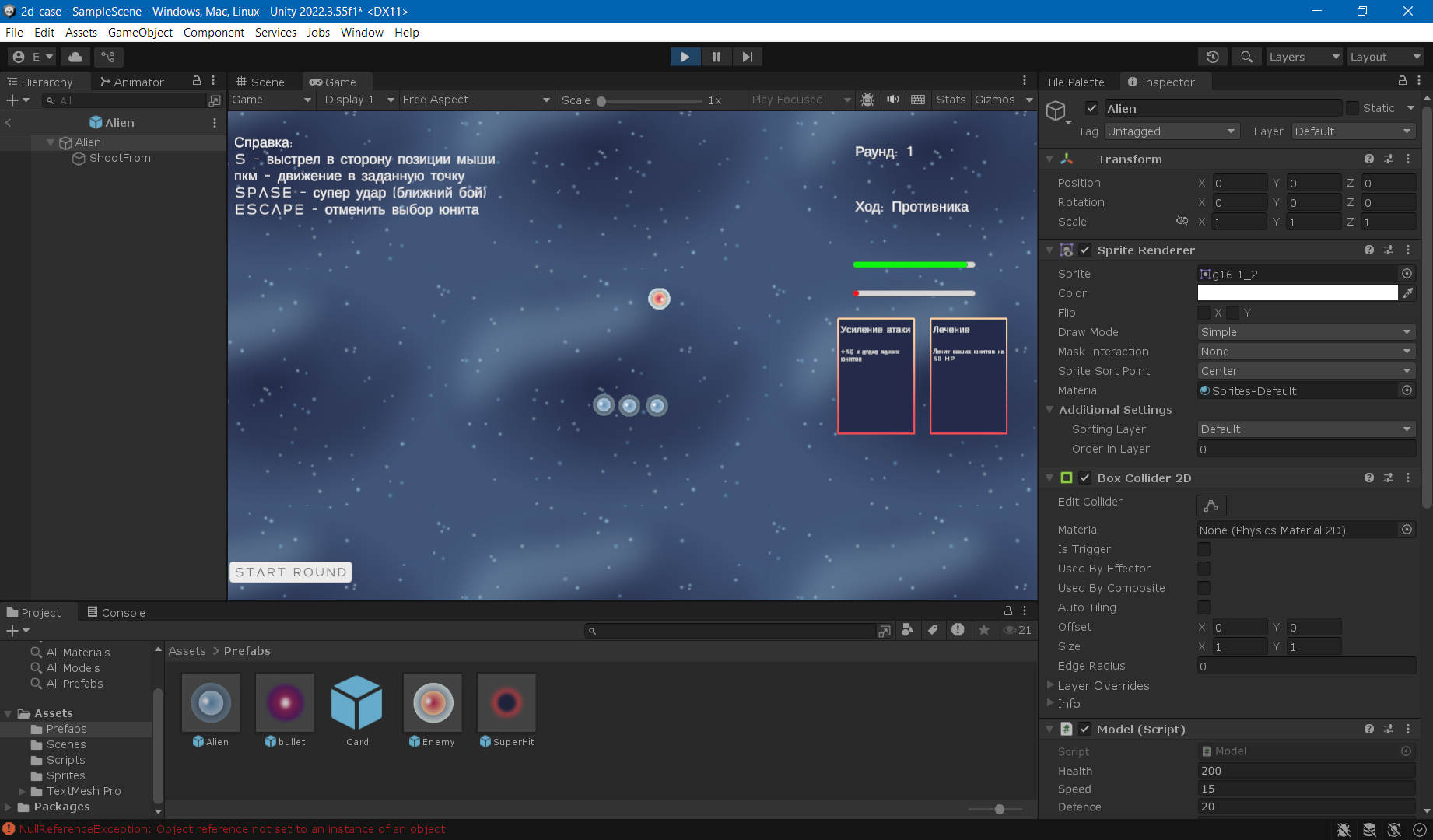
}

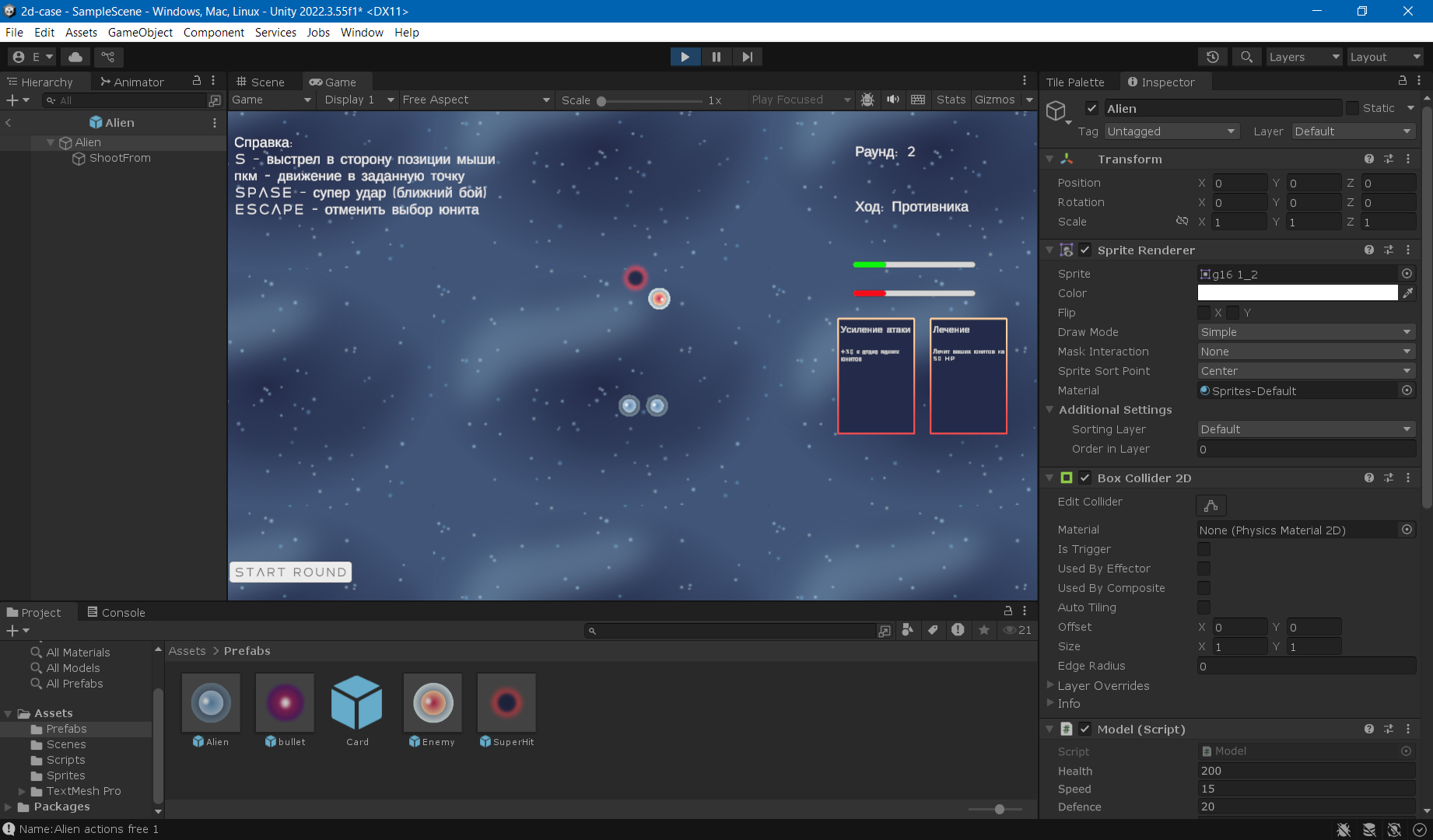
}

return false;

}

}





Карты

using System.Collections.Generic;

using UnityEngine;

using System;

using TMPro;

/// <summary>

/// Класс карты, реализующий систему карточных эффектов и их применение к группам юнитов

/// </summary>

public class Card : MonoBehaviour

{

[SerializeField] private TextMeshProUGUI Name; // Текстовое поле для отображения названия карты

[SerializeField] private TextMeshProUGUI Description; // Текстовое поле для отображения описания карты

private CardOptions card; // Текущие параметры карты

private System.Random randint = new System.Random(); // Генератор случайных чисел

/// <summary>

/// Событие, вызываемое при выборе карты

/// </summary>

public static Action onCardSelected;

/// <summary>

/// Структура, описывающая параметры карты

/// </summary>

public struct CardOptions

{

public string Name; // Название карты

public string Description; // Описание эффекта карты

public List<Action> Effects; // Список действий, выполняемых картой

}

/// <summary>

/// Статический класс, содержащий возможные эффекты карт

/// </summary>

public static class CardEffects

{

/// <summary>

/// Лечение группы юнитов

/// </summary>

/// <param name="count">Количество восстанавливаемого здоровья</param>

/// <param name="group">Список юнитов для лечения</param>

public static void ModifyHeal(float count, List<GameObject> group)

{

foreach (GameObject unit in group)

{

if (unit.TryGetComponent(out Model model))

{

model.Heal(count);

}

}

}

/// <summary>

/// Увеличение защиты группы юнитов

/// </summary>

/// <param name="defenseBoost">Бонус к защите</param>

/// <param name="group">Список юнитов для усиления</param>

public static void ModifyDefense(float defenseBoost, List<GameObject> group)

{

foreach (GameObject unit in group)

{

if (unit.TryGetComponent(out Model model))

{

model.ModifyDefense(defenseBoost);

}

}

}

/// <summary>

/// Увеличение атаки группы юнитов

/// </summary>

/// <param name="attackBoost">Бонус к атаке</param>

/// <param name="group">Список юнитов для усиления</param>

public static void ModifyAttack(float attackBoost, List<GameObject> group)

{

foreach (GameObject unit in group)

{

if (unit.TryGetComponent(out Model model))

{

model.ModifyAttack(attackBoost);

}

}

}

}

/// <summary>

/// Массив возможных карт с их эффектами

/// </summary>

public CardOptions[] Options = new CardOptions[]

{

new CardOptions()

{

Name = "Лечение",

Description = "Лечит ваших юнитов на 50 HP",

Effects = new List<Action>

{

() =>

{

var targetGroup = RoundSystem.GetQueue ? RoundSystem.GetEnemyGroup : RoundSystem.GetAllienGroup;

CardEffects.ModifyHeal(50, targetGroup);

}

}

},

new CardOptions()

{

Name = "Усиление защиты",

Description = "+40 к защите ваших юнитов",

Effects = new List<Action>

{

() =>

{

var targetGroup = RoundSystem.GetQueue ? RoundSystem.GetEnemyGroup : RoundSystem.GetAllienGroup;

CardEffects.ModifyDefense(40, targetGroup);

}

}

},

new CardOptions()

{

Name = "Усиление атаки",

Description = "+30 к атаке ваших юнитов",

Effects = new List<Action>

{

() =>

{

var targetGroup = RoundSystem.GetQueue ? RoundSystem.GetEnemyGroup : RoundSystem.GetAllienGroup;

CardEffects.ModifyAttack(30, targetGroup);

}

}

}

};

/// <summary>

/// При активации карты обновляет текстовые поля

/// </summary>

private void OnEnable()

{

Name.text = card.Name;

Description.text = card.Description;

}

/// <summary>

/// При деактивации карты выбирает случайный новый эффект

/// </summary>

private void OnDisable()

{

card = Options[randint.Next(3)];

}

/// <summary>

/// Деактивирует карту

/// </summary>

public void Disable()

{

gameObject.SetActive(false);

}

/// <summary>

/// Активирует карту

/// </summary>

public void Enable()

{

gameObject.SetActive(true);

}

/// <summary>

/// Инициализация карты при создании

/// </summary>

private void Awake()

{

onCardSelected += Disable;

card = Options[randint.Next(3)];

Name.text = card.Name;

Description.text = card.Description;

}

/// <summary>

/// Применяет эффекты текущей карты

/// </summary>

public void UseEffect()

{

foreach (var effect in card.Effects)

{

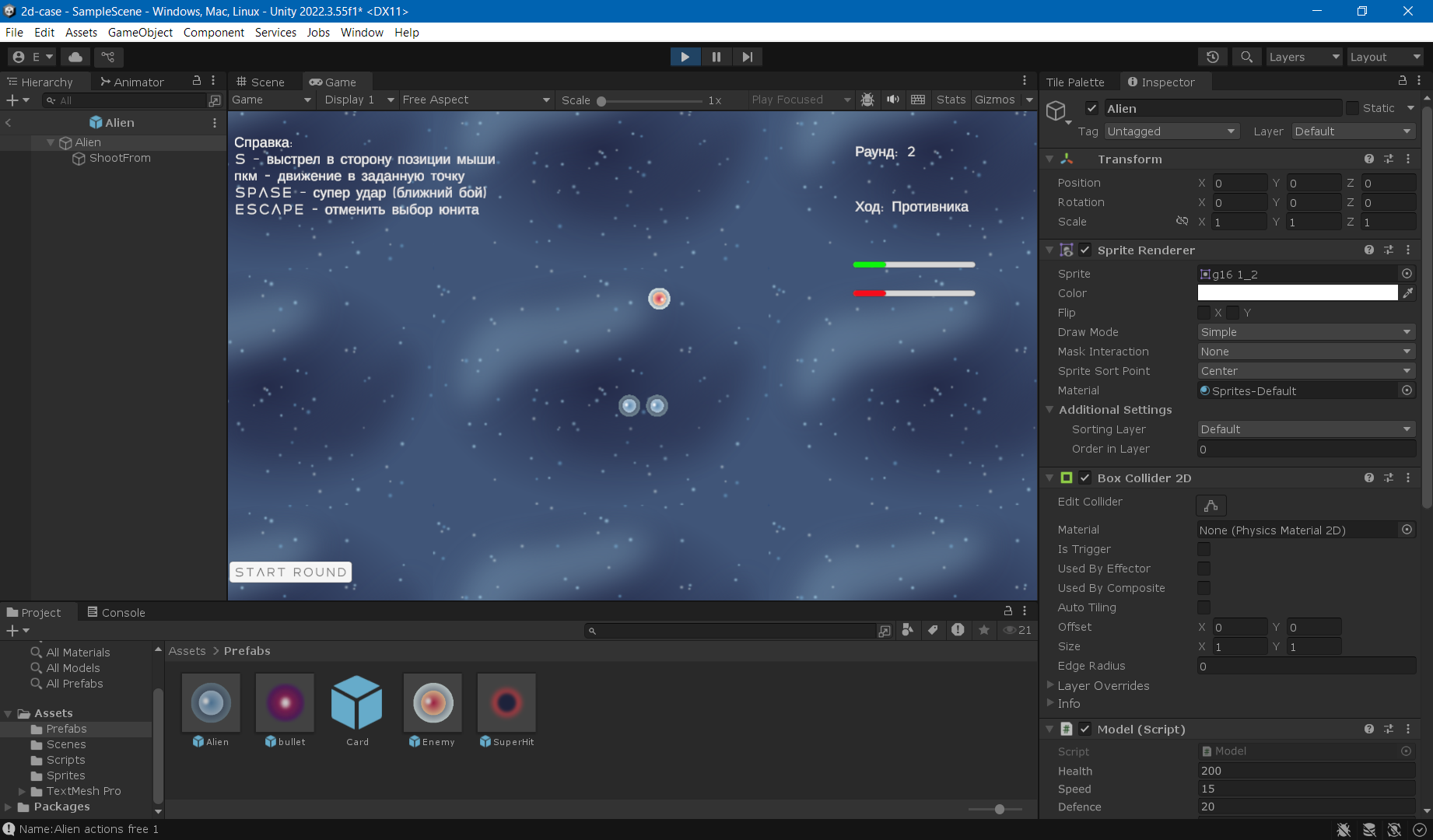
effect?.Invoke();

}

onCardSelected?.Invoke();

}

}



Дополнительно

* Ссылка на репозиторий: [Ersoil/Case](https://github.com/Ersoil/Case)
* Ссылка на демонстрационное видео: [мышка при записи сбилась.mp4](https://1drv.ms/v/c/1fd28d83461ec201/EcTg9W0R5jpGihrpFGQsBtcBY6GVjFbIraBK_xXZXpmNOA?e=7LpELl)