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 аркады Vacuum Frenzy»
по ДПП ПП «Основы Gamedev и VR-разработки»

п/п №	Задание	Исполнитель	Рабочий график (план) выполнения
1	Изучение механик классической аркады и аналогичных игр для понимания ключевых элементов геймплея.	Зобнина Н. Д. Захарьина Т. В.	05.05.2025 - 20.05.2025
2	Создать рабочую версию игры в жанре казуальной аркады с уровнями, системой очков и сохранением прогресса.		20.05.2025 - 16.06.2025
3	Сделать выводы исходя из результатов работы.		16.06.2025 - 18.06.2025
4	Подготовка отчета и презентации.		18.06.2025 - 22.06.2025

Руководитель проекта
_старший преподаватель кафедры культурологии и дизайна Каратаев Алексей Антонович__

_____ /_____ / «___ » _____ 2025 г.
(подпись)

СОГЛАСОВАНО:

Руководитель ДПП ПП
заведующий кафедрой информатики Козлов Денис Юрьевич____

_____ /_____/ «____»_____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 группового итогового проекта по ДПП ПП

«Основы Gamedev и VR-разработки»

«Разработка аркады Vacuum Frenzy»

Исполнители:
_Зобнина Н. Д.__ *(Фамилия И.О.)*
Захарьина Т. В.
_____
_____
_____

«____» _____ 2025 г.

Руководитель проекта
__Каратаев_А. А.___
«____» _____ 2025 г.

г. Барнаул, 2025

1. Цель проекта

Целью данного проекта является создание казуальной аркадной игры, вдохновленной классическим Pacman, с использованием движка Unity. Проект направлен на изучение основ геймдизайна, программирования игровой логики и работы с искусственным интеллектом для противников.

2. Задачи проекта и исполнители

Для достижения поставленной цели необходимо решить следующие задачи:

1) Изучение механик классической аркады: анализ оригинального Pacman и аналогичных игр для понимания ключевых элементов геймплея.
2) Разработка игрового движка: создание системы управления персонажем, обработки столкновений и подсчета очков.
3) Реализация ИИ для врагов: программирование поведения противников с разными паттернами движения.
4) Тестирование и балансировка: проверка игрового процесса, исправление багов и настройка сложности.
5) Добавление визуальных и звуковых эффектов: улучшение игровой атмосферы с помощью анимаций и звуков.

Исполнители:

- Программист – разработка кода игры, реализация механик и ИИ.
- Геймдизайнер – проектирование уровней и балансировка геймплея.
- Художник – создание спрайтов и анимаций.

3. Актуальность и востребованность проекта

Разработка казуальной аркады, вдохновленной классическим Pacman, представляет собой значимый проект с точки зрения современных тенденций игровой индустрии. В условиях стремительного развития технологий и изменения потребительских предпочтений подобные игры сохраняют свою привлекательность благодаря сочетанию простоты геймплея и глубины игрового опыта. Сейчас игровой рынок демонстрирует устойчивый интерес, как к ретро-стилистике, так и к переосмыслению проверенных временем механик, что делает проект соответствующим актуальным запросам аудитории.

Классические аркадные игры, к которым относится Pacman, заложили фундамент для всей индустрии видеоигр. Их механики прошли испытание временем и продолжают использоваться в современных проектах. Популярность таких игр объясняется их доступностью - они не требуют длительного обучения, но при этом предлагают увлекательный игровой процесс. В последние годы наблюдается заметный рост интереса к ретро-играм, что проявляется в успехе ремейков и переизданий классических тайтлов. Например, такие проекты как Pac-Man Championship Edition DX и Pac-Man 99 доказали, что обновленные версии легендарной аркады могут успешно конкурировать на современном рынке.

С образовательной точки зрения разработка подобного проекта представляет особую ценность. Создание игры в стиле Pacman позволяет глубоко изучить ключевые аспекты игрового дизайна и программирования. В процессе работы осваиваются принципы построения игровой логики, обработки пользовательского ввода, управления искусственным интеллектом противников. Особое значение имеет работа с системой коллизий и физикой движка Unity, что составляет основу любого игрового проекта. При этом относительно

простая структура игры позволяет сосредоточиться на освоении фундаментальных концепций, которые в дальнейшем могут быть применены в более сложных разработках.

Перспективы развития проекта выходят за рамки создания законченного игрового продукта. Разработанные технологии и решения могут стать основой для дальнейших экспериментов и усовершенствований. В частности, система искусственного интеллекта, управляющего поведением врагов, может быть модернизирована с использованием более сложных алгоритмов поиска пути. Возможна реализация процедурной генерации уровней, что значительно расширит реиграбельность проекта. Интересным направлением развития могло бы стать добавление мультиплеерного режима, который позволил бы игрокам соревноваться друг с другом или объединяться для совместного прохождения.

Социальный аспект проекта также заслуживает внимания. Игры подобного типа обладают универсальной привлекательностью и могут служить своеобразным мостом между поколениями игроков. Они одинаково интересны как тем, кто помнит оригинальный Pacman, так и современным игрокам, что создает дополнительные возможности для популяризации проекта. Кроме того, простота и доступность геймплея делают игру потенциально привлекательной для самой широкой аудитории, включая людей, которые обычно не считают себя геймерами.

4. Общие сведения о проделанной работе

4.1 Этапы выполнения проекта

Разработка казуальной аркадной игры в стиле Pacman проводилась в несколько последовательных этапов. Первоначально был проведен анализ классической механики оригинальной игры и современных аналогов, что позволило выделить ключевые игровые элементы, которые необходимо было реализовать. На следующем этапе была создана базовая архитектура проекта, включающая систему управления персонажем, карту поля и простейшую систему коллизий. Затем последовала реализация искусственного интеллекта для противников и системы подсчета очков. Завершающим этапом стала интеграция визуальных и звуковых эффектов, а также тестирование и балансировка игрового процесса.

4.2 Выбранные инструменты и технологии

В качестве основного инструмента разработки был выбран движок Unity, который предоставляет все необходимые средства для создания 2D-игр. Unity был выбран благодаря своей универсальности, кроссплатформенности и наличию мощного визуального редактора. Для программирования игровой логики использовался язык C#, который является стандартным для Unity и обладает оптимальным сочетанием производительности и удобства разработки.

Для разработки спрайтов использовалось приложение Ibis Paint, что позволило создать уникальные анимации персонажей и объектов. Для управления анимацией был реализован скрипт AnimatedSprites.cs, который переключает спрайты в зависимости от состояния игрового объекта. Звуковые эффекты были взяты из бесплатных открытых источников и интегрированы в

проект с помощью стандартной системы AudioSource в Unity, а AudioMixer дополнительно для возможности управления громкостью эффектов во время игры. Также использовался Tilemap для создания и редактирования игровых уровней и UI System для реализации интерфейса.

4.3 Реализация искусственного интеллекта

Для создания поведения противников были использованы несколько подходов. Основной алгоритм движения врагов реализован на основе конечных автоматов с тремя основными состояниями: преследование, разброс, «дом» и убегание (испуг). У босса есть дополнительные состояния стрельбы и взрыва. Каждый из этих режимов контролируется отдельным компонентом, что позволяет гибко переключаться между поведениями в зависимости от ситуации.

В состоянии преследования враги стремятся приблизиться к цели, выбирая направление движения, которое минимизирует расстояние до игрока. Для этого при прохождении через узлы навигации они оценивают доступные пути и выбирают наиболее оптимальный. В состоянии разброса - враги движутся в случайном направлении. Испуг активируется, когда игрок съедает рыбку, заставляя врагов убегать от игрока и делая их уязвимыми. Состояние "дом" отвечает за возврат противников в их исходную позицию после того, как они были съеден.

В режиме стрельбы босс периодически выпускает снаряды в сторону игрока и преследует его. Стрельба останавливается при испуге или других влияниях. Взрыв активируется после съедания всех игровых объектов.

Вся система построена так, чтобы переключение между состояниями происходило плавно и без конфликтов, обеспечивая естественное и разнообразное поведение врагов. Каждое состояние сопровождается

уникальной анимацией. Взаимодействие с игроком также учитывает состояние врага: при столкновении с игроком происходит либо уничтожение врага, либо съедение игрока.

4.4 Обоснование выбранных решений

Выбор Unity в качестве основного инструмента разработки был обусловлен несколькими факторами. Во-первых, движок предоставляет все необходимые инструменты для быстрого прототипирования и итеративной разработки. Во-вторых, встроенная система компонентов позволяет гибко настраивать поведение игровых объектов без необходимости написания сложного кода. Наконец, кроссплатформенность Unity дает возможность в будущем легко портировать игру на различные платформы.

Использование конечных автоматов для ИИ противников было выбрано как наиболее подходящее решение для аркадной игры. Этот подход обеспечивает предсказуемое поведение врагов, позволяет удобно и гибко переключаться между разными состояниями, что важно для балансировки игрового процесса, и в то же время позволяет создавать достаточно сложные модели поведения без чрезмерного усложнения кода.

4.5 Ход разработки

Работа над проектом началась с создания базовых игровых механик: движения главного персонажа, системы сбора точек и простейших коллизий. После отладки основных систем началась работа над реализацией искусственного интеллекта противников.

Параллельно велась работа над визуальной составляющей игры. Были созданы спрайты персонажей, элементы интерфейса и игрового окружения. Заключительным этапом стала интеграция звуковых эффектов и музыкального сопровождения, что значительно улучшило общее впечатление от игры.

Тестирование проводилось как в процессе разработки (альфа-тестирование), так и после завершения основных работ (бета-тестирование). Это позволило выявить и исправить множество проблем, связанных с балансом игры, производительностью и удобством управления. В результате был получен стабильный и сбалансированный игровой продукт, сохраняющий дух классической аркады.

5. Результаты проекта

В результате проведённой работы был успешно разработан и реализован полноценный игровой прототип казуальной аркады «Вакуумное безумие». Игра представляет собой современную интерпретацию классической игры Pacman с уникальным стилистическим оформлением.

Основные достижения проекта включают:

1) Реализация игровых механик.
   - Создана система управления главным персонажем (котом) с плавной анимацией движений
   - Разработано 5 уникальных уровней с постепенно возрастающей сложностью, в том числе уровень с боссом.
2) Искусственный интеллект противников.
   - Враги (пылесосы) демонстрируют различные модели поведения
   - Реализованы два типа ИИ: преследующий и убегающий

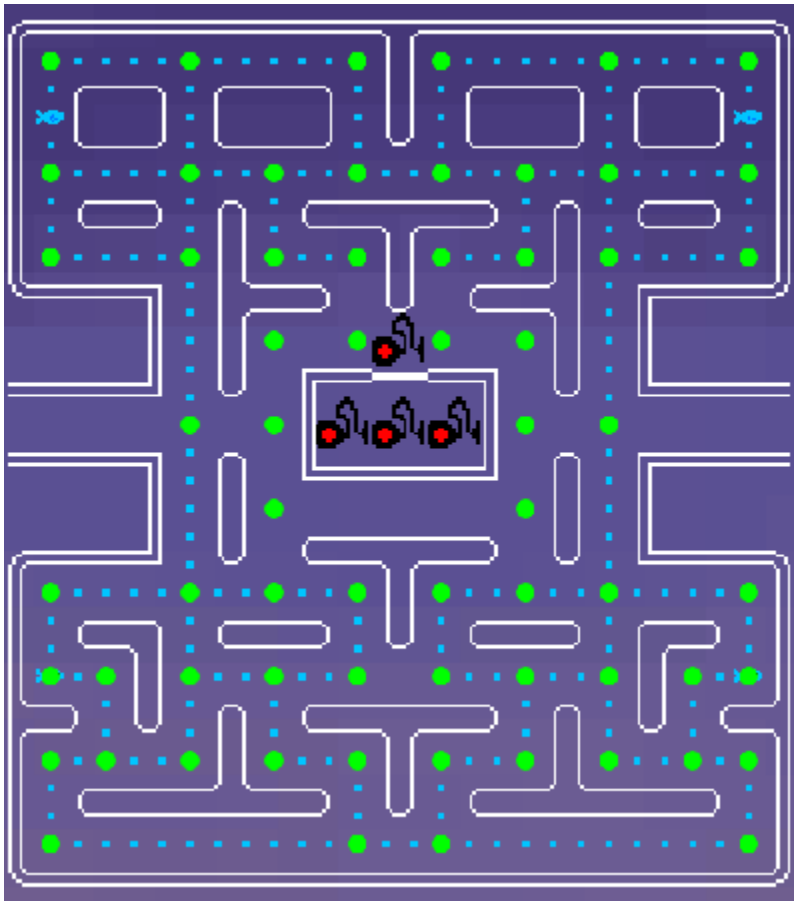

Рисунок 1. Узлы перемещения на игровом поле

3) Технические характеристики.
   - Стабильная работа на мобильных устройствах.
   - Оптимизированная производительность даже на слабом железе

4) Визуальное оформление.
   - Уникальный пиксель-арт стиль с ретро-элементами
   - Анимационные эффекты для игровых объектов

Рисунок 2. Анимация смерти игрока

5) Тестирование и балансировка.
   - Проведено более 50 тестовых прохождений
   - Оптимизирована сложность уровней

Проект показал хороший потенциал для дальнейшего развития. В ходе тестирования были выявлены несколько направлений для совершенствования:

1) Возможность добавления процедурной генерации уровней
2) Реализация системы редактирования карт
3) Введение новых типов противников с уникальным поведением
4) Разработка сетевого режима игры

Техническая база проекта позволяет относительно легко интегрировать новые функции без необходимости переработки основной архитектуры. Это открывает широкие перспективы для развития игры, как в качественном, так и в количественном отношении.

Приложение 1

Рис. 3 Главное меню

Рис. 4 Настройки

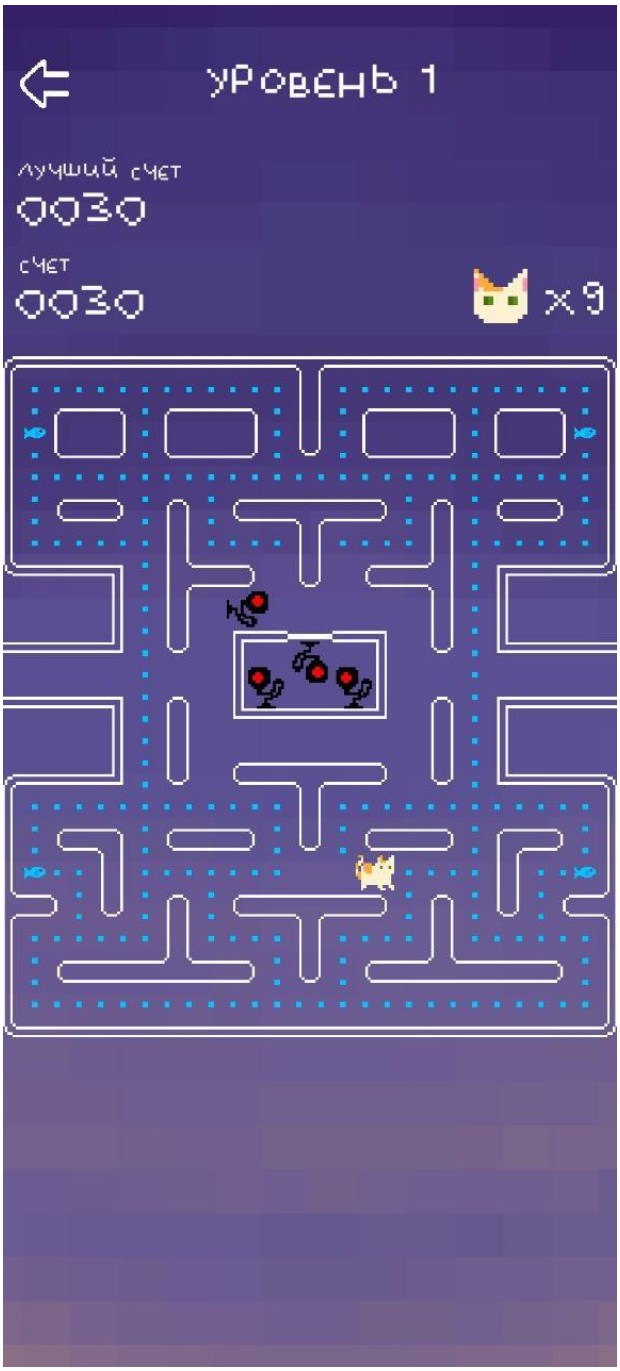

Рис. 5 Уровень

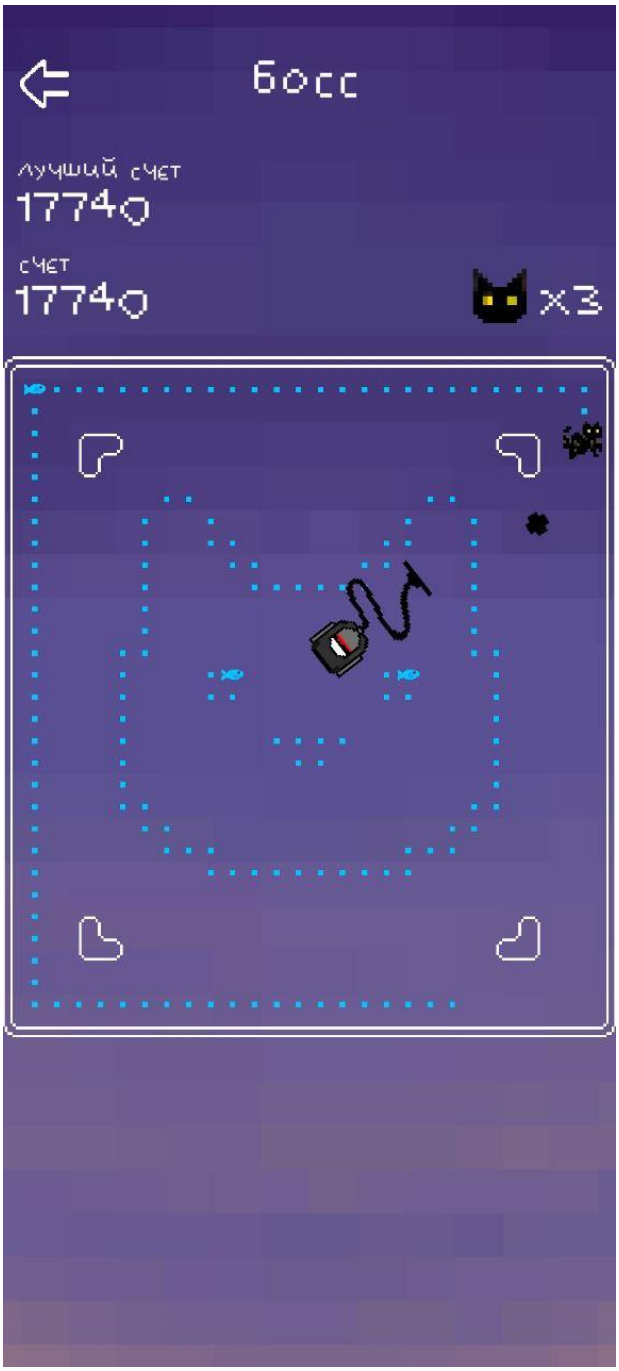

Рис. 6 Уровень с боссом

Приложение 2

Основные фрагменты программного кода на C#

Скрипт GameManager.cs. Реализует основную работу игры

```
using UnityEngine;
using TMPro;
using UnityEngine.InputSystem;
using UnityEngine.SceneManagement;

[DefaultExecutionOrder(-100)]
public class GameManager : MonoBehaviour
{
    public static GameManager Instance { get; private set; }

    public enum GameMode { Normal, Endless }
    public GameMode CurrentGameMode { get; private set; } = GameMode.Normal;

    private Vacuum[] vacuums;
    protected Cat cat;
    private Transform pellets;
    private TextMeshProUGUI gameOverText;
    private TextMeshProUGUI scoreText;
    private TextMeshProUGUI livesText;
    private TextMeshProUGUI highScoreText;
    public AudioSource audioSource;
    public AudioClip pelletEatClip;
    public AudioClip powerPelletEatClip;
    public AudioClip catEatenClip;
    public AudioClip catWinClip;
    public AudioClip vacuumEatenClip;
    private bool waitingForRestart = false;

    public int highScore { get; private set; }
    public int score { get; private set; } = 0;
    public int lives { get; private set; } = 9;

    private int vacuumMultiplier = 1;

    private void Awake()
    {
        if (Instance != null && Instance != this)
        {
            Destroy(gameObject);
            return;
        }

        Instance = this;
        DontDestroyOnLoad(gameObject);
        highScore = PlayerPrefs.GetInt("HighScore", 0);
    }

    private void OnEnable() => SceneManager.sceneLoaded += OnSceneLoaded;
```

```
    private void OnDisable() => SceneManager.sceneLoaded -= OnSceneLoaded;

    private void OnSceneLoaded(Scene scene, LoadSceneMode mode)
    {
        string name = scene.name.ToLower();

        if (name.Contains("endless"))
            SetGameMode(GameMode.Endless);
        else
            SetGameMode(GameMode.Normal);

        vacuums = FindObjectsOfType<Vacuum>();

        pellets = GameObject.Find("Pellets")?.transform;
        gameOverText =
GameObject.Find("GameOverText")?.GetComponent<TextMeshProUGUI>();
        scoreText =
GameObject.Find("ScoreText")?.GetComponent<TextMeshProUGUI>();
        livesText =
GameObject.Find("LivesText")?.GetComponent<TextMeshProUGUI>();
        highScoreText =
GameObject.Find("HighScoreText")?.GetComponent<TextMeshProUGUI>();

        UpdateUI();
        NewRound();
        Time.timeScale = 1f;
        waitingForRestart = false;
    }

    private void Update()
    {
        if (lives <= 0 && gameOverText != null && gameOverText.enabled &&
!waitingForRestart)
        {
            if ((Keyboard.current?.anyKey.wasPressedThisFrame ?? false) ||
                (Touchscreen.current?.primaryTouch.press.wasPressedThisFrame ??
false) ||
                (Mouse.current?.leftButton.wasPressedThisFrame ?? false))
            {
                waitingForRestart = true;
                RestartGame();
            }
        }
    }

    private void RestartGame()
    {
        NewGame();

        if (CurrentGameMode == GameMode.Endless)
        {
            SceneManager.LoadScene(SceneManager.GetActiveScene().buildIndex);
        }
        else
        {
            SceneManager.LoadScene(1);
        }
    }

    private void NewGame()
```

```
{
    SetScore(0);
    SetLives(9);
    vacuumMultiplier = 1;
}

protected void NewRound()
{
    if (gameOverText != null)
        gameOverText.enabled = false;

    if (pellets != null)
    {
        foreach (Transform pellet in pellets)
            pellet.gameObject.SetActive(true);
    }

    ResetState();
}

private void ResetState()
{
    foreach (Vacuum vacuum in vacuums)
        vacuum.ResetState();

    if (cat != null)
        cat.ResetState();

    VacuumShoot.AllowShooting = true;
}

private void SetLives(int value)
{
    lives = value;
    if (livesText != null)
        livesText.text = "x" + lives.ToString();
}

protected void SetScore(int value)
{
    score = value;
    if (scoreText != null)
        scoreText.text = score.ToString().PadLeft(4, '0');

    if (score > highScore)
    {
        highScore = score;
        PlayerPrefs.SetInt("HighScore", highScore);
        PlayerPrefs.Save();
        if (highScoreText != null)
            highScoreText.text = highScore.ToString().PadLeft(4, '0');
    }
}

private void UpdateUI()
{
    SetScore(score);
    SetLives(lives);
    if (highScoreText != null)
        highScoreText.text = highScore.ToString().PadLeft(4, '0');
```

```
    }

    public void CatEaten()
    {
        if (cat != null)
        {
            Projectile.DestroyAllProjectiles();
            cat.DeathSequence();
            if (audioSource != null && catEatenClip != null)
                audioSource.PlayOneShot(catEatenClip);
        }

        SetLives(lives - 1);

        VacuumShoot bossShoot = FindObjectOfType<VacuumShoot>();
        if (bossShoot != null && bossShoot.vacuum != null &&
bossShoot.vacuum.isBoss)
        {
            VacuumShoot.AllowShooting = false;
        }

        if (lives > 0)
        {
            Invoke(nameof(ResetState), 3f);
        }
        else
        {
            GameOver();
        }
    }

    private void GameOver()
    {
        if (gameOverText != null)
            gameOverText.enabled = true;

        foreach (Vacuum vacuum in vacuums)
            vacuum.gameObject.SetActive(false);

        if (cat != null)
            cat.gameObject.SetActive(false);
    }

    public void VacuumEaten(Vacuum vacuum)
    {
        int points = vacuum.points * vacuumMultiplier;
        SetScore(score + points);
        vacuumMultiplier++;

        if (audioSource != null && vacuumEatenClip != null)
            audioSource.PlayOneShot(vacuumEatenClip);
    }

    public virtual void PelletEaten(Pellet pellet)
    {
        pellet.gameObject.SetActive(false);
        SetScore(score + pellet.points);

        if (audioSource != null && pelletEatClip != null)
            audioSource.PlayOneShot(pelletEatClip);
```

```
    if (!HasRemainingPellets())
    {
        if (cat != null)
        {
            cat.WinSequence();
            if (audioSource != null && catWinClip != null)
                audioSource.PlayOneShot(catWinClip);
        }

        if (CurrentGameMode == GameMode.Endless)
            Invoke(nameof(NewRound), 3f);
        else
            Invoke(nameof(LoadNextLevel), 3f);
    }
}

public void PowerPelletEaten(PowerPellet pellet)
{
    foreach (Vacuum vacuum in vacuums)
        vacuum.frightened.Enable(pellet.duration);

    if (audioSource != null && powerPelletEatClip != null)
        audioSource.PlayOneShot(powerPelletEatClip);

    PelletEaten(pellet);
    CancelInvoke(nameof(ResetVacuumMultiplier));
    Invoke(nameof(ResetVacuumMultiplier), pellet.duration);
}

protected bool HasRemainingPellets()
{
    if (pellets == null) return false;

    foreach (Transform pellet in pellets)
    {
        if (pellet.gameObject.activeSelf)
            return true;
    }

    return false;
}

private void ResetVacuumMultiplier()
{
    vacuumMultiplier = 1;
}

protected virtual void LoadNextLevel()
{
    if (CurrentGameMode == GameMode.Endless)
    {
        NewRound(); // В Endless просто новая волна
        return;
    }

    int currentIndex = SceneManager.GetActiveScene().buildIndex;
    int nextIndex = currentIndex + 1;

    if (nextIndex < SceneManager.sceneCountInBuildSettings)
```

```
        {
            SceneManager.LoadScene(nextIndex);
        }
        else
        {
            SceneManager.LoadScene(0);
            NewGame();
        }
    }

    public bool AllPelletsEaten() => !HasRemainingPellets();

    public void ResetHighScore()
    {
        PlayerPrefs.DeleteKey("HighScore");
        highScore = 0;
        UpdateUI();

        MenuHighScoreDisplay menuDisplay =
FindObjectOfType<MenuHighScoreDisplay>();
        if (menuDisplay != null)
            menuDisplay.UpdateHighScoreDisplay();
    }

    public void RegisterCat(Cat catInstance)
    {
        cat = catInstance;
        cat.ResetState();
    }

    public void SetGameMode(GameMode mode)
    {
        CurrentGameMode = mode;
    }
}
```

Скрипт Cat.cs. Реализует управление игроком

```
using UnityEngine;
using UnityEngine.InputSystem;
using UnityEngine.UI;

[RequireComponent(typeof(Movement))]
public class Cat : MonoBehaviour
{
    [SerializeField]
    private AnimatedSprite deathSequence;
    public AnimatedSprite winSequence;
    private SpriteRenderer spriteRenderer;
    private CircleCollider2D circleCollider;
    private Movement movement;

    private Vector2 startTouchPos;
    private Vector2 endTouchPos;
    private bool swipeHandled = false;
    public Rigidbody2D rb;
    public AnimatedSprite walking;


    private void Awake()
    {
        this.movement = GetComponent<Movement>();
        spriteRenderer = GetComponent<SpriteRenderer>();
        this.circleCollider = GetComponent<CircleCollider2D>();
    }
    void Start()
    {
        GameManager.Instance?.RegisterCat(this);

        Vacuum[] vacuums = FindObjectsOfType<Vacuum>();
        foreach (var vacuum in vacuums)
        {
            vacuum.SetTarget(transform);
        }
    }

    private void Update()
    {
#if UNITY_EDITOR || UNITY_STANDALONE
        var keyboard = Keyboard.current;

        if (keyboard != null)
        {
            if (keyboard.wKey.wasPressedThisFrame ||
keyboard.upArrowKey.wasPressedThisFrame)
            {
                movement.SetDirection(Vector2.up);
            }
            else if (keyboard.sKey.wasPressedThisFrame ||
keyboard.downArrowKey.wasPressedThisFrame)
            {
                movement.SetDirection(Vector2.down);
            }
            else if (keyboard.aKey.wasPressedThisFrame ||
keyboard.leftArrowKey.wasPressedThisFrame)
            {
```

```
                movement.SetDirection(Vector2.left);
            }
            else if (keyboard.dKey.wasPressedThisFrame ||
keyboard.rightArrowKey.wasPressedThisFrame)
            {
                movement.SetDirection(Vector2.right);
            }
        }
#endif

        var touchscreen = Touchscreen.current;
        if (touchscreen != null && touchscreen.touches.Count > 0)
        {
            var touch = touchscreen.primaryTouch;

            if (touch.press.wasPressedThisFrame)
            {
                startTouchPos = touch.position.ReadValue();
                swipeHandled = false;
            }
            else if (touch.press.wasReleasedThisFrame && !swipeHandled)
            {
                Vector2 endTouchPos = touch.position.ReadValue();
                Vector2 delta = endTouchPos - startTouchPos;

                if (delta.magnitude > 50f)
                {
                    swipeHandled = true;

                    if (Mathf.Abs(delta.x) > Mathf.Abs(delta.y))
                    {
                        movement.SetDirection(delta.x > 0 ? Vector2.right :
Vector2.left);
                    }
                    else
                    {
                        movement.SetDirection(delta.y > 0 ? Vector2.up :
Vector2.down);
                    }
                }
            }
        }

        if (movement.direction.x < 0)
        {
            walking.transform.localScale = new Vector3(-1, 1, 1);
        }
        else if (movement.direction.x > 0)
        {
            walking.transform.localScale = new Vector3(1, 1, 1);
        }
    }

    public void ResetState()
    {
        deathSequence.gameObject.SetActive(false);
        winSequence.gameObject.SetActive(false);
        enabled = true;
        spriteRenderer.enabled = true;
        circleCollider.enabled = true;
```

```
        deathSequence.enabled = false;
        winSequence.enabled = false;
        movement.ResetState();
        gameObject.SetActive(true);
    }

    public void DeathSequence()
    {
        deathSequence.gameObject.SetActive(true);
        enabled = false;
        spriteRenderer.enabled = false;
        circleCollider.enabled = false;
        movement.enabled = false;
        deathSequence.enabled = true;
        deathSequence.Restart();
    }

    public void WinSequence()
    {
        winSequence.gameObject.SetActive(true);
        enabled = false;
        spriteRenderer.enabled = false;
        circleCollider.enabled = false;
        movement.enabled = false;
        winSequence.enabled = true;
        winSequence.Restart();
    }
}
```

Скрипт Vacuum.cs. Реализует работу врагов.

```
using UnityEngine;

public class Vacuum : MonoBehaviour
{
    public Movement movement { get; private set; }
    public VacuumHome home { get; private set; }
    public VacuumScatter scatter { get; private set; }
    public VacuumChase chase { get; private set; }
    public VacuumFrightened frightened { get; private set; }
    public VacuumShoot shooting { get; private set; }

    public VacuumBehavior initialBehavior;
    [HideInInspector] public Transform target;
    private bool isPaused;
    public bool isBoss = false;
    public bool isExploded = false;

    public int points = 200;

    private void Awake()
    {
        this.movement = GetComponent<Movement>();
        this.home = GetComponent<VacuumHome>();
        this.scatter = GetComponent<VacuumScatter>();
        this.chase = GetComponent<VacuumChase>();
        this.frightened = GetComponent<VacuumFrightened>();

        if (CompareTag("Boss"))
        {
            this.shooting = GetComponent<VacuumShoot>();
        }
    }

    private void Start()
    {
        ResetState();
    }

    private void FixedUpdate()
    {
        UpdateRotation();
    }

    public void UpdateRotation()
    {
        if (movement.direction != Vector2.zero)
        {
            float angle = Mathf.Atan2(movement.direction.y,
movement.direction.x) * Mathf.Rad2Deg;
            transform.rotation = Quaternion.AngleAxis(angle, Vector3.forward);
        }
    }

    public void ForceUpdateRotation()
    {
        UpdateRotation();
    }
```

```
public void ResetState()
{
    this.gameObject.SetActive(true);
    this.movement.ResetState();

    this.frightened.Disable();
    this.chase.Disable();
    if (isExploded) return;
    else this.scatter.Enable();

    if (!CompareTag("Boss"))
    {
        this.scatter.Enable();

        if (this.home != null && this.home != this.initialBehavior)
        {
            this.home.Disable();
        }

        if (this.initialBehavior != null)
        {
            this.initialBehavior.Enable();
        }
    }
    else
    {
        transform.position = this.movement.startingPosition;
        this.scatter.Enable();
    }

    if (this.initialBehavior != null)
    {
        this.initialBehavior.Enable();
    }

    if (this.shooting != null)
    {
        this.shooting.Disable();
    }

    ForceUpdateRotation();
}

private void OnCollisionEnter2D(Collision2D collision)
{
    if (collision.gameObject.layer == LayerMask.NameToLayer("Cat"))
    {
        if (frightened.enabled)
        {
            GameManager.Instance.VacuumEaten(this);
        }
        else
        {
            GameManager.Instance.CatEaten();
        }
    }
}

public void SetTarget(Transform catTransform)
```

```
    {
        this.target = catTransform;
    }

    public void DisableAllBehaviors()
    {
        if (scatter != null) scatter.enabled = false;
        if (chase != null) chase.enabled = false;
        if (frightened != null) frightened.enabled = false;
        if (shooting != null) shooting.enabled = false;
    }
}
```

Скрипт VacuumBehavior.cs. Хранит методы для наследников: VacuumChase.cs, VacuumScatter.cs, VacuumFrightened.cs, VacuumHome.cs, VacuumShoot.cs

```
using UnityEngine;

[RequireComponent (typeof(Vacuum))]
public abstract class VacuumBehavior : MonoBehaviour
{
    public Vacuum vacuum {  get; private set; }
    public float duration;

    private void Awake()
    {
        this.vacuum = GetComponent<Vacuum> ();
        this.enabled = false;
    }

    public void Enable()
    {
         Enable(this.duration);
    }

    public virtual void Enable(float duration)
    {
        if (vacuum.isExploded)
            return;

        this.enabled = true;

        CancelInvoke();
        Invoke(nameof(Disable), duration);
    }

    public virtual void Disable()
    {
        this.enabled = false;

        CancelInvoke();
    }

}
```

Скрипт Movement.cs. Реализует движение врагов по игровому полю.

```
using UnityEngine;

[RequireComponent(typeof(Rigidbody2D))]
public class Movement : MonoBehaviour
{
    public float speed = 8.0f;
    public float speedMultiplier = 1.0f;
    public Vector2 initialDirection;
    public LayerMask obstacleLayer;

    public bool isBoss = false;

    public new Rigidbody2D rigidbody { get; private set; }
    public Vector2 direction { get; private set; }
    public Vector2 nextDirection { get; private set; }
    public Vector3 startingPosition { get; private set; }

    private void Awake()
    {
        this.rigidbody = GetComponent<Rigidbody2D>();
        this.startingPosition = this.transform.position;
    }

    private void Start()
    {
        ResetState();
    }

    public void ResetState()
    {
        this.speedMultiplier = 1.0f;
        this.direction = this.initialDirection;
        this.nextDirection = Vector2.zero;
        this.transform.position = this.startingPosition;
        this.rigidbody.isKinematic = false;
        this.enabled = true;
    }

    private void Update()
    {
        if (nextDirection != Vector2.zero)
        {
            SetDirection(nextDirection);
        }
    }

    private void FixedUpdate()
    {
        if (Time.timeScale == 0f) return;

        if (this.nextDirection != Vector2.zero)
        {
            TrySetDirection(this.nextDirection);
        }

        Vector2 position = this.rigidbody.position;
        Vector2 translation = this.direction * this.speed * this.speedMultiplier
* Time.fixedDeltaTime;
```

```
        this.rigidbody.MovePosition(position + translation);
    }

    public void SetDirection(Vector2 direction, bool forced = false)
    {
        if (isBoss || forced || !Occupied(direction))
        {
            this.direction = direction;
            this.nextDirection = Vector2.zero;
        }
        else
        {
            this.nextDirection = direction;
        }
    }

    private void TrySetDirection(Vector2 direction)
    {
        if (isBoss)
        {
            this.direction = direction;
            this.nextDirection = Vector2.zero;
            return;
        }

        Vector2 position = this.rigidbody.position;
        Vector2 offsetPosition = position + direction * 0.5f;

        RaycastHit2D hit = Physics2D.BoxCast(offsetPosition, Vector2.one *
0.75f, 0f, direction, 0.05f, this.obstacleLayer);

        if (hit.collider == null)
        {
            this.direction = direction;
            this.nextDirection = Vector2.zero;
        }
    }

    public bool Occupied(Vector2 direction)
    {
        if (isBoss) return false;

        Vector2 position = this.rigidbody.position;
        RaycastHit2D hit = Physics2D.BoxCast(position, Vector2.one * 0.75f, 0f,
direction, 1.5f, this.obstacleLayer);
        return hit.collider != null;
    }
}
```

Скрипт AnimatedSprite.cs. Реализует анимацию объектов с помощью простого переключения спрайтов

```
using UnityEngine;

[RequireComponent(typeof(SpriteRenderer))]
public class AnimatedSprite : MonoBehaviour
{
    public SpriteRenderer spriteRenderer { get; private set; }
    public Sprite[] sprites;
    public float animationTime = 0.25f;
    public int animationFrame { get; private set; }
    public bool loop = true;

    private void Awake()
    {
        this.spriteRenderer = GetComponent<SpriteRenderer>();
    }

    private void Start()
    {
        InvokeRepeating(nameof(Advance), this.animationTime,
this.animationTime);
    }

    private void Advance()
    {
        if (!this.spriteRenderer.enabled)
        {
            return;
        }

        this.animationFrame++;

        if (this.animationFrame >= this.sprites.Length && this.loop)
        {
            this.animationFrame = 0;
        }

        if (this.animationFrame >= 0 && this.animationFrame <
this.sprites.Length)
        {
            this.spriteRenderer.sprite = this.sprites[this.animationFrame];
        }
    }

    public void Restart()
    {
        this.animationFrame = -1;
        Advance();
    }
    public void Enable()
    {
        this.enabled = true;
        this.spriteRenderer.enabled = true;
        this.gameObject.SetActive(true);
```

```
    }

    public void Disable()
    {
        this.enabled = false;
        this.spriteRenderer.enabled = false;
        this.gameObject.SetActive(false);
    }
}
```

Приложение 3

Ссылка на презентацию проекта:
https://drive.google.com/file/d/197YJ0J0rE-S2HuRsBHc-IxmLockfeoEj/view?usp=sharing

Ссылка на видео презентацию:
https://drive.google.com/file/d/1XRrc-JlVpmACKKmlDwVP-Ou1SFMw2Mkh/view?usp=sharing

Ссылка на видео демонстрацию геймплея:
https://drive.google.com/file/d/1ldUhZXbWFNeO3-Ny6FkAkNYZjZOfVwVB/view?usp=sharing

Ссылка на репозиторий:
https://github.com/tanya-zakharina/Vacuum-frenzy