|  |
| --- |
| МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ  федеральное государственное автономное образовательное учреждение высшего образования  «САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ  АЭРОКОСМИЧЕСКОГО ПРИБОРОСТРОЕНИЯ» |

Кафедра \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(наименование)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| ОТЧЁТ ПО ПРАКТИКЕ  ЗАЩИЩЁН С ОЦЕНКОЙ  Руководитель |  | | | |
|  |  |  |  |  |
| должность, уч. степень, звание |  | подпись, дата |  | инициалы, фамилия |

ОТЧЁТ ПО ПРАКТИКЕ

|  |  |  |  |
| --- | --- | --- | --- |
| вид практики |  | |  |
| тип практики |  | |  |
| на тему индивидуального задания | |  |  |
|  | | | |
|  | | | |

|  |  |
| --- | --- |
| выполнен |  |
| фамилия, имя, отчество обучающегося в творительном падеже | |

|  |  |  |  |
| --- | --- | --- | --- |
| по направлению подготовки |  |  |  |
|  | код |  | наименование направления |
|  | | | |
| наименование направления | | | |
| направленности |  |  |  |
|  | код |  | наименование направленности |
|  | | | |
| наименование направленности | | | |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Обучающийся группы № |  |  |  |  |  |
|  | номер |  | подпись, дата |  | инициалы, фамилия |

Санкт–Петербург 2024

ИНДИВИДУАЛЬНОЕ ЗАДАНИЕ

на прохождение практики обучающегося направления подготовки/ специальности

1. Фамилия, имя, отчество обучающегося:

1. Группа:
2. Тема индивидуального задания:

1. Исходные данные:

1. Содержание отчетной документации:
   1. индивидуальное задание;
   2. отчёт, включающий в себя:
   * титульный лист;
   * материалы о выполнении индивидуального задания (содержание определяется кафедрой);
   * выводы по результатам практики;
   * список использованных источников.
2. Срок представления отчета на кафедру: «\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_\_\_20\_\_

Руководитель практики

должность, уч. степень, звание подпись, дата инициалы, фамилия

Задание принял к исполнению

обучающийся

дата подпись инициалы, фамилия

Санкт–Петербург 2024

СОДЕРЖАНИЕ

[ВВЕДЕНИЕ 4](#__RefHeading___Toc2117_3227342530)

[1 Тема и описание игры 7](#__RefHeading___Toc2119_3227342530)

[2 Пользовательская документация 8](#__RefHeading___Toc2127_3227342530)

[2.1 Установка и запуск 8](#__RefHeading___Toc2137_3227342530)

[2.2 Управление 8](#__RefHeading___Toc2137_32273425303)

[2.3 Выход из игры 9](#__RefHeading___Toc2137_322734253034)

[3 Техническая документация 10](#__RefHeading___Toc2125_3227342530)

[4 Тестирование программы 11](#__RefHeading___Toc2123_3227342530)

[5 Описание назначения и процесса загрузки проекта на GitHub 12](#__RefHeading___Toc2121_3227342530)

[ЗАКЛЮЧЕНИЕ 14](#__RefHeading___Toc2115_3227342530)

[СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 16](#__RefHeading___Toc2113_3227342530)

[ПРИЛОЖЕНИЕ 17](#__RefHeading___Toc2111_3227342530)

ВВЕДЕНИЕ

В рамках учебной практики, моё задание заключалось в создании видеоигры на предпочитаемом языке программирования. Я выбрал Unity и язык программирования C# для разработки игры в стиле Flappy Bird под названием «Floppa The Bird».

Цели выполнения данного проекта включают в себя следующее:

* + Закрепление и углубление теоретических знаний, полученных при изучении профессиональных дисциплин.
  + Приобретение практических навыков и компетенций в сфере профессиональной деятельности.
  + Освоение перспективных информационных технологий.
  + Приобретение опыта применения современной вычислительной техники для решения практических задач

В процессе работы предстоит улучшить свои профессиональные навыки, решая следующие задачи:

* + Изучение специальной литературы и другой научно-технической информации.
  + Сбор, обработка, анализ и систематизация научно-технической информации по теме (заданию).
  + Оформление результатов анализа информации по заданной теме и собственных исследований и разработок в виде отчета и технической документации.

Unity — это мощная и гибкая платформа для разработки игр и интерактивного контента. Она предоставляет полный набор инструментов и возможностей, которые позволяют создавать игры для различных платформ, включая ПК, мобильные устройства, игровые консоли и виртуальную реальность.

Основные особенности Unity:

* + Мультиплатформенность: Unity поддерживает экспорт игр на более чем 25 платформ, включая Windows, macOS, Linux, Android, iOS, PlayStation, Xbox, Nintendo Switch, и другие.
  + Интуитивно понятный интерфейс: Unity предлагает удобную среду разработки с графическим интерфейсом, который облегчает создание и настройку игровых объектов, сцен и анимаций.
  + Физический движок: Встроенный физический движок позволяет реалистично моделировать поведение объектов, включая столкновения, гравитацию и другие физические явления.
  + Гибкость скриптинга: Поддержка языка программирования C# обеспечивает мощные возможности для создания сложной игровой логики и взаимодействия между объектами.
  + Большое сообщество и ресурсы: Unity имеет обширное сообщество разработчиков, а также богатую базу обучающих материалов, документации и готовых компонентов, доступных в Unity Asset Store.
  + Поддержка 2D и 3D: Unity позволяет создавать как двухмерные, так и трехмерные игры, предоставляя инструменты для работы с различными типами графики и анимации.

C# — это современный, объектно-ориентированный язык программирования, разработанный корпорацией Microsoft. Он является основным языком для разработки на платформе .NET и широко используется в различных областях программирования, включая разработку настольных приложений, веб-приложений и игр.

Основные особенности C#:

* + Объектно-ориентированный подход: C# поддерживает основные принципы объектно-ориентированного программирования (ООП), такие как инкапсуляция, наследование и полиморфизм, что способствует модульности и повторному использованию кода.
  + Простота и удобочитаемость: Синтаксис C# разработан так, чтобы быть легким для чтения и понимания, что упрощает написание и сопровождение кода.
  + Типобезопасность: C# строго типизированный язык, что помогает предотвратить многие типы ошибок на этапе компиляции.
  + Межплатформенность: С появлением .NET Core и .NET 5/6, приложения на C# могут работать на различных операционных системах, включая Windows, macOS и Linux.

В контексте разработки игр на Unity, C# используется для написания скриптов, которые определяют поведение игровых объектов, взаимодействие с пользователем и логику игрового процесса. Сочетание возможностей Unity и языка C# позволяет создавать мощные, интерактивные и кроссплатформенные игры.

1. Тема и описание игры

Проект представляет из себя видеоигру в стиле Flappy Bird с оригинальным персонажем и уникальными элементами окружения.

Floppa The Bird — это захватывающая аркадная игра, где игрок управляет птицей по имени Флопа, летящей на фоне звездного неба и многоквартирных домов. Основная цель игры — пролететь как можно дальше, избегая столкновений с высотными зданиями и набирая максимальное количество очков.

Главный персонаж — птица, которая летает, взмахивая крыльями каждый раз, когда игрок нажимает на назначенную кнопку. Игрок управляет Флопой, избегая препятствий, возникающих на пути.

Фон игры представляет собой звездное небо и стену из многоквартирных домов, создающих атмосферу ночного города. Игрок должен маневрировать Флопой между зданиями, которые выступают в качестве препятствий.

Простое и интуитивно понятное управление: игрок нажимает на кнопку, чтобы заставить Флопу взмахивать крыльями и поддерживать его в полете.

Очки начисляются за каждое успешно преодоленное препятствие. Игра ведет учет набранных очков и позволяет игрокам устанавливать и бить собственные рекорды. На экране отображаются текущие очки и рекорд.

Floppa The Bird — это аркадная игра с потенциалом для дальнейшего развития и совершенствования. Она сочетает в себе простоту управления и захватывающий игровой процесс, делая ее привлекательной для широкого круга игроков. Разработка этой игры позволила мне не только применить свои теоретические знания на практике, но и приобрести ценные навыки в области программирования и игрового дизайна.

1. Пользовательская документация
   1. Установка и запуск

Для всех платформ игра представляет из себя директорию с различными файлами. В корне директории с игрой находится исполнительный файл, с помощью которого следует запустить игру.

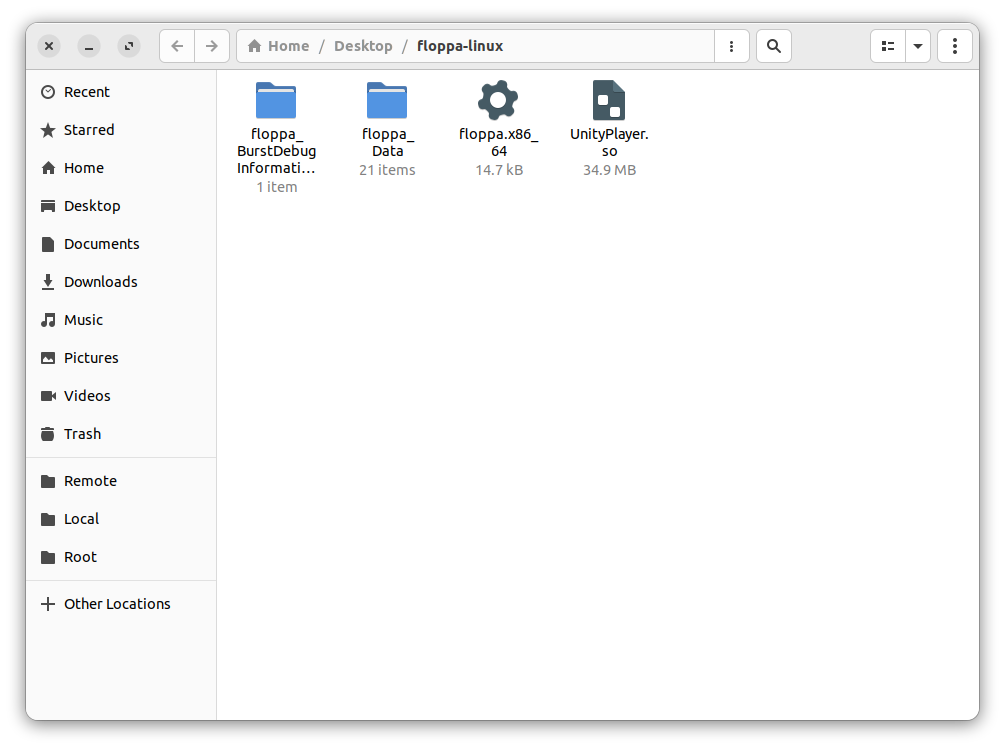


Рисунок 1 – Игра на Linux

* 1. Управление

Управление неизменяемое и представляет из себя одно действие — прыжок. Осуществляется это действие либо нажатием клавиши Space, либо щелчком левой кнопки мыши.



Рисунок 2 – Пример игры в процессе

* 1. Выход из игры

Для выхода из игры следует выйти в главное меню и нажать на кнопку «Exit».



Рисунок 3 – Главное меню игры

1. Техническая документация

Для работы с проектом необходим Unity Editor версии 2021.3.17f1, с другими версиями могут возникнуть проблемы (например несоответсвие версий использумеых пакетов).

Описание основных функций и методов приведены в таблице, оформленной в файле README репозитория проекта:



Рисунок 4 – Описание основных реализованных методов игровых объектов

1. Тестирование программы

Тестирование является критически важным этапом разработки любой видеоигры, включая Floppa The Bird. Оно позволяет выявить и устранить ошибки, убедиться в правильной работе игровых механик, а также улучшить общее качество игры. Тестирование игры включало в себя:

1. Проверка всех основных функций игры, таких как управление Флопой, генерация препятствий, подсчет очков и система рекордов.
2. Тестирование реакции игры на нажатия экрана или кнопки, проверка плавности полета игрока.
3. Убедиться, что препятствия появляются с правильной частотой и в нужных местах.
4. Проверка правильности начисления очков за преодоленные препятствия.
5. Проверка сохранения и отображения текущего и максимального набранного счета.
6. Проверка взаимодействия между различными компонентами игры, такими как игровой процесс, интерфейс пользователя и аудио-эффекты.
7. Проверка удобства и интуитивности игрового интерфейса, легкости навигации по меню и доступности информации для игрока.
8. Оценка общего игрового опыта.

В процессе тестирования, помимо пробных запусков приложения и экспериментальных проверок использовалась консоль разработчика, куда выводятся возникающие ошибки и другая отладочная информация.

Тестирование — это итеративный процесс, включающий повторение этапов до тех пор, пока не будут устранены все критические ошибки и недочеты. Благодаря тестированию, удалось исправить основное количествео ошибок в программном коде Floppa The Bird.

Короткое видео с демонстрацией механик игры доступно по ссылке: https://youtu.be/tzV2l3ffRrI.

1. Описание назначения и процесса загрузки проекта на GitHub

GitHub — это веб-сервис для хостинга и совместной разработки программного обеспечения с использованием системы контроля версий Git. Назначение загрузки проекта на GitHub включает несколько ключевых аспектов:

1. Совместная работа: GitHub позволяет нескольким разработчикам работать над одним проектом одновременно, облегчая координацию и управление изменениями в коде.
2. Контроль версий: использование Git обеспечивает хранение истории изменений, возможность возврата к предыдущим версиям, отслеживание и объединение различных веток разработки.
3. Безопасность и резервное копирование: хранение проекта на удаленном сервере защищает его от потери данных и обеспечивает доступность из любой точки мира.
4. Публичные и приватные репозитории: GitHub позволяет создавать как публичные репозитории для открытого исходного кода, так и приватные репозитории для закрытых проектов.
5. Интеграции и автоматизация: поддержка интеграции с различными инструментами и сервисами CI/CD, такими как GitHub Actions, для автоматизации тестирования, сборки и развертывания приложений.

Процесс загрузки заключался в следущем:

1. Создание удаленного репозитория на сайте GitHub
2. Инициализация локального репозитория с помощью git init
3. Добавление файлов в индекс и коммит: git add . и git commit -m "Initial commit"
4. Подключение к удаленному репозиторию: git remote add origin <URL репозитория>
5. Загрузка текущей ветки в удаленный репозиторий: git push -u origin master

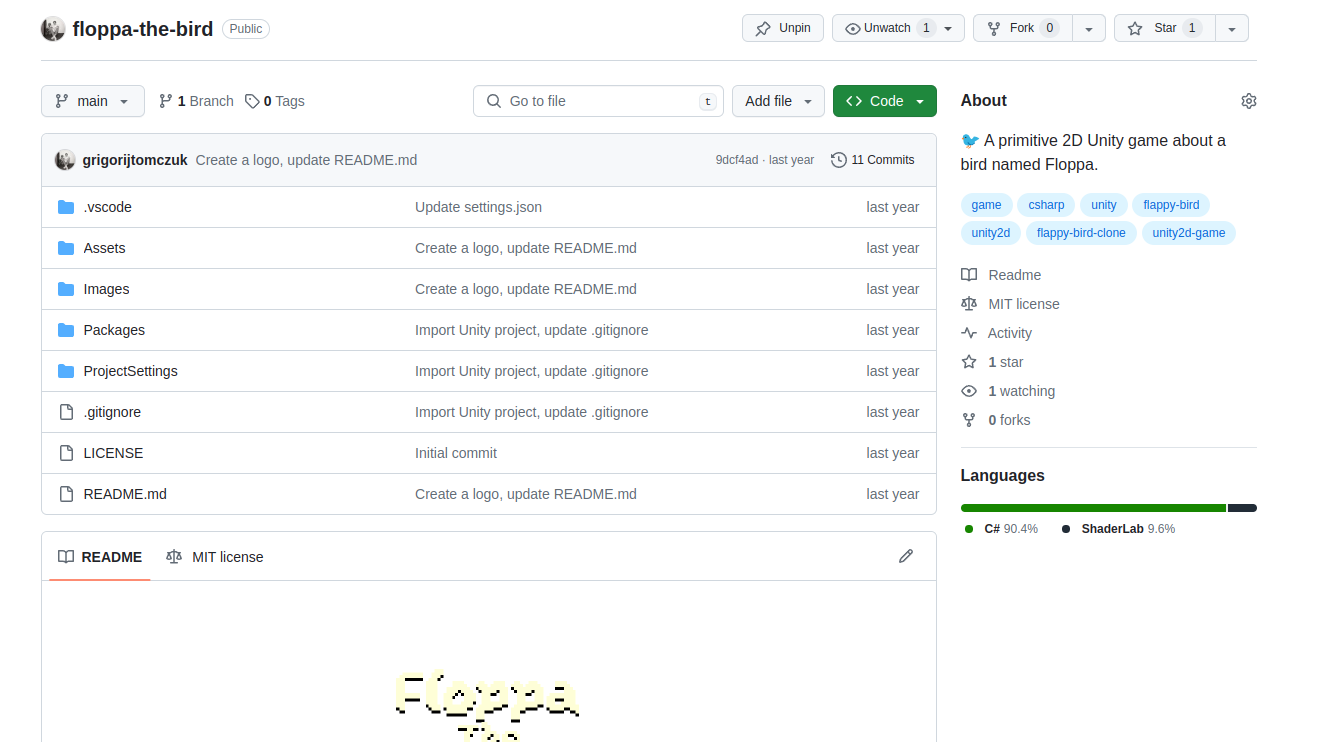


Рисунок 5 – Репозиторий (https://github.com/grigorijtomczuk/floppa-the-bird)

ЗАКЛЮЧЕНИЕ

В ходе учебной практики я выполнил задание по созданию видеоигры, используя платформу Unity и язык программирования C#. Разработанная мной игра Floppa The Bird представляет собой увлекательную аркаду в стиле Flappy Bird, где игрок управляет птицей, летящей на фоне звездного неба и многоквартирных домов, избегая столкновений с препятствиями.

Цели учебной практики были достигнуты. Я смог закрепить и углубить теоретические знания, полученные в ходе изучения профессиональных дисциплин. Разработка игры позволила мне приобрести практические навыки программирования и использования современных инструментов разработки, а также познакомиться с новыми информационными технологиями и методами проектирования.

В процессе работы над проектом я выполнил следующие задачи:

* + Разработал концепцию и дизайн игры, учитывая элементы игрового процесса и визуальные особенности.
  + Реализовал основные механики игры, такие как управление персонажем, генерация препятствий, подсчет очков и система рекордов.
  + Проанализировал специальную литературу и другие научно-технические источники для поиска оптимальных решений и лучших практик в разработке видеоигр.
  + Систематизировал и оформил результаты своей работы в виде отчета и технической документации.

Практика оказалась полезной и продуктивной, способствовала формированию профессионально значимых качеств и укреплению интереса к сфере информационных технологий и программирования. Полученные знания и опыт будут незаменимы в дальнейшей учебной и профессиональной деятельности.

Проект Floppa The Bird имеет значительный потенциал для дальнейшего развития. В перспективе игру можно дорабатывать, вводя новые режимы и уровни сложности, добавляя уникальные препятствия и бонусы, а также расширяя функционал для большего разнообразия игрового процесса. Это позволит не только улучшить игровой опыт, но и привлечь больше игроков, предоставив им более интересные и сложные задачи.

В заключение стоит отметить, что выполнение данного проекта не только подтвердило важность теоретических знаний, но и продемонстрировало необходимость практического применения современных технологий для решения реальных задач.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Unity Technologies. Документация Unity [Электронный ресурс]. — URL: https://docs.unity3d.com/Manual/index.html (дата обращения: 02.05.2024).
2. Карпухин, А. В. Unity в действии. Мультимедийное приложение для начинающих / А. В. Карпухин. — Москва: Диалектика, 2019. — 320 с.
3. Microsoft. Документация по C# [Электронный ресурс]. — URL: https://docs.microsoft.com/ru-ru/dotnet/csharp/ (дата обращения: 11.05.2024).
4. Gamedev.ru. Форум разработчиков игр [Электронный ресурс]. — URL: https://gamedev.ru/ (дата обращения: 30.05.2024).
5. Чернявский, В. И. Unity 2019. Создание игр с нуля / В. И. Чернявский. — Санкт-Петербург: Питер, 2020. — 352 с.
6. Астанин, М. В. Разработка игр на Unity для начинающих / М. В. Астанин. — Санкт-Петербург: Питер, 2018. — 256 с.
7. Линдсей, К. Разработка игр на C# и Unity / К. Линдсей. — Москва: Бином. Лаборатория знаний, 2019. — 368 с.
8. Habr. Статьи по разработке игр на Unity [Электронный ресурс]. — URL: https://habr.com/ru/hub/gamedev/ (дата обращения: 24.05.2024).

ПРИЛОЖЕНИЕ

Managers/LogicManager.cs

using System;  
using UnityEngine;  
using UnityEngine.UI;  
using UnityEngine.SceneManagement;  
  
#if UNITY\_EDITOR  
  
using UnityEditor;  
  
#endif  
  
public class **LogicManager** : MonoBehaviour  
{  
 *// \* Remember to serialize fields in custom LogicManagerEditor class as well*  
[SerializeField]  
 private Preset **preset**;  
  
 [SerializeField]  
 private Text **scoreText**;  
  
 [SerializeField]  
 private Text **highScoreText**;  
  
 [SerializeField]  
 private GameObject **hud**;  
  
 [SerializeField]  
 private GameObject **gameOverScreen**;  
  
 private enum Preset { None, **TitleScreen**, **MainScene** };  
  
 private AudioManager audioManager;  
  
 private int playerScore;  
 private int playerHighScore;  
 private static string currentSceneName;  
 private string previousSceneName;  
  
#if UNITY\_EDITOR  
  
 [CustomEditor(typeof(LogicManager))]  
 public class **LogicManagerEditor** : Editor  
 {  
 public override void OnInspectorGUI()  
 {  
 LogicManager logicManager = (LogicManager)target;  
  
 serializedObject.Update();  
  
 SerializePropertyField("preset");  
   
 if (logicManager.preset == Preset.**TitleScreen**)  
 {  
 EditorGUILayout.Space();  
 SerializePropertyField("highScoreText");  
 }  
  
 if (logicManager.preset == Preset.**MainScene**)  
 {  
 EditorGUILayout.Space();  
 SerializePropertyField("scoreText");  
 SerializePropertyField("highScoreText");  
 SerializePropertyField("hud");  
 SerializePropertyField("gameOverScreen");  
 }  
  
 serializedObject.ApplyModifiedProperties();  
  
 void SerializePropertyField(string propertyPath)  
 {  
 EditorGUILayout.PropertyField(serializedObject.FindProperty(propertyPath));  
 }  
 }  
 }  
  
#endif  
  
 void **Start**()  
 {  
 audioManager = GameObject.FindGameObjectWithTag("Audio").GetComponent<AudioManager>();  
  
 *// Store current scene name in a static variable to have an access to it in the subsequent manager instances*  
previousSceneName = currentSceneName;  
 currentSceneName = SceneManager.GetActiveScene().name;  
  
 if (currentSceneName == "TitleScreen")  
 {  
 audioManager.PlayMusic("titleScreen");  
 }  
  
 *// Check additionally if the scene switched from another; if so - replay music; do nothing otherwise (keep music playing)*  
if (currentSceneName == "Main" && previousSceneName != currentSceneName)  
 {  
 audioManager.PlayMusic("chiptune");  
 }  
   
 if (preset == Preset.**TitleScreen**)  
 {  
 playerHighScore = PlayerPrefs.GetInt("playerHighScore", 0);  
 highScoreText.text = playerHighScore.ToString();  
 }  
  
 if (preset == Preset.**MainScene**)  
 {  
 playerScore = 0;  
 playerHighScore = PlayerPrefs.GetInt("playerHighScore", 0);  
  
 scoreText.text = playerScore.ToString();  
 highScoreText.text = playerHighScore.ToString();  
 }  
 }  
  
 public void AddScore(int scoreToAdd)  
 {  
 audioManager.PlaySound("score");  
  
 playerScore += scoreToAdd;  
 scoreText.text = playerScore.ToString();  
  
 if (playerScore > playerHighScore)  
 {  
 playerHighScore = playerScore;  
 PlayerPrefs.SetInt("playerHighScore", playerHighScore);  
 highScoreText.text = playerHighScore.ToString();  
 }  
 }  
   
 public void ResetHighScore()  
 {  
 playerHighScore = 0;  
 PlayerPrefs.SetInt("playerHighScore", playerHighScore);  
 highScoreText.text = playerHighScore.ToString();  
 }  
  
 public void SwitchToTitleScreen()  
 {  
 SceneManager.LoadScene("TitleScreen");  
 }  
  
 public void SwitchToMainScene()  
 {  
 SceneManager.LoadScene("Main");  
 }  
  
 [ContextMenu("Restart Scene")]  
 public void RestartScene()  
 {  
 SceneManager.LoadScene(currentSceneName);  
 }  
  
 public void SetGameOver()  
 {  
 hud.SetActive(false);  
 gameOverScreen.SetActive(true);  
 }  
  
 public void ExitGame()  
 {  
 Application.Quit();  
 }  
}

Managers/AudioManager.cs

using System.Collections.Generic;  
using UnityEngine;  
  
public class **AudioManager** : MonoBehaviour  
{  
 [System.Serializable]  
 public class Sound  
 {  
 public string **name**;  
 public AudioClip **clip**;  
 }  
  
 *// Need multiple audio sources for sounds because of the "variable pitch" feature (to avoid adrupt pitch override)*  
[SerializeField]  
 private AudioSource **soundSourceA**;  
  
 [SerializeField]  
 private AudioSource **soundSourceB**;  
  
 [SerializeField]  
 private AudioSource **musicSource**;  
  
 [SerializeField]  
 private List<Sound> **sounds**;  
  
 [SerializeField]  
 private List<Sound> **tracks**;  
  
 private static AudioManager objectInstance;  
  
 void **Awake**()  
 {  
 if (objectInstance == null)  
 {  
 objectInstance = this;  
 DontDestroyOnLoad(this);  
 }  
 else  
 {  
 Destroy(gameObject);  
 }  
 }  
  
 public void PlaySound(string name)  
 {  
 Sound sound = sounds.Find(x => x.name == name);  
  
 if (sound == null)  
 {  
 Debug.Log($"Failed to play \"{name}\" sound.");  
 }  
 else if (sound.name == "flap")  
 {  
 soundSourceB.pitch = Random.Range(0.9f, 1.1f);  
 soundSourceB.PlayOneShot(sound.clip);  
 }  
 else  
 {  
 soundSourceA.pitch = Random.Range(0.9f, 1.1f);  
 soundSourceA.PlayOneShot(sound.clip);  
 }  
 }  
  
 public void PlayMusic(string name)  
 {  
 Sound track = tracks.Find(x => x.name == name);  
  
 if (track == null)  
 {  
 Debug.Log($"Failed to play \"{name}\" track.");  
 }  
 else  
 {  
 musicSource.clip = track.clip;  
 musicSource.Play();  
 }  
 }  
}

Generic/BirdController.cs

using UnityEngine;

public class **BirdController** : MonoBehaviour

{

[SerializeField]

private float **flapForce**;

private LogicManager logicManager;

private AudioManager audioManager;

private Rigidbody2D rb;

private SpriteRenderer sprite;

private Animator anim;

private bool isAlive = true;

private float deadZoneY = 3.55f;

void **Start**()

{

logicManager = GameObject.FindGameObjectWithTag("Logic").GetComponent<LogicManager>();

audioManager = GameObject.FindGameObjectWithTag("Audio").GetComponent<AudioManager>();

rb = GetComponent<Rigidbody2D>();

sprite = GetComponent<SpriteRenderer>();

anim = GetComponent<Animator>();

}

void **Update**()

{

if ((Input.GetKeyDown(KeyCode.**Space**) || Input.GetKeyDown(KeyCode.**Mouse0**)) && isAlive)

{

Flap();

}

if (transform.position.y >= deadZoneY || transform.position.y <= -deadZoneY)

{

Die();

}

}

void Flap()

{

audioManager.PlaySound("flap");

anim.Play("BirdFlap");

rb.velocity = Vector2.up \* flapForce;

}

void Die()

{

isAlive = false;

sprite.flipY = true;

logicManager.SetGameOver();

}

void **OnCollisionEnter2D**(Collision2D collision)

{

audioManager.PlaySound("hit");

Die();

}

}

Generic/PipeController.cs

using UnityEngine;  
  
public class **PipeController** : MonoBehaviour  
{  
 [SerializeField]  
 private float **moveSpeed**;  
  
 private float deadZoneX = -8;  
  
 void **Update**()  
 {  
 transform.position += Vector3.left \* moveSpeed \* Time.deltaTime;  
 if (transform.position.x <= deadZoneX)  
 {  
 Destroy(gameObject);  
 }  
 }  
}

Generic/PipeGenerator.cs

using UnityEngine;  
  
public class **PipeGenerator** : MonoBehaviour  
{  
 [SerializeField]  
 private GameObject **pipes**;  
  
 [SerializeField]  
 private float **spawnRate**;  
  
 [SerializeField]  
 private float **heightOffset**;  
  
 private float timer = 0f;  
  
 void **Start**()  
 {  
 SpawnPipes(); *// Initial spawn*  
}  
  
 void **Update**()  
 {  
 if (timer > spawnRate)  
 {  
 SpawnPipes();  
 timer = 0f;  
 }  
 else  
 {  
 timer += Time.deltaTime;  
 }  
 }  
  
 void SpawnPipes()  
 {  
 float lowestY = transform.position.y - heightOffset;  
 float highestY = transform.position.y + heightOffset;  
 Instantiate(pipes, new Vector3(transform.position.x, Random.Range(lowestY, highestY), transform.position.z), transform.rotation);  
 }  
}

Generic/PipeTrigger.cs

using UnityEngine;  
  
public class **PipeTrigger** : MonoBehaviour  
{  
 private LogicManager logic;  
  
 void **Start**()  
 {  
 logic = GameObject.FindGameObjectWithTag("Logic").GetComponent<LogicManager>();  
 }  
  
 void **OnTriggerEnter2D**(Collider2D collision)  
 {  
 if (collision.gameObject.layer == 6)  
 {  
 logic.AddScore(1);  
 }  
 }  
}

Generic/BackgroundController.cs

using UnityEngine;  
  
public class **BackgroundController** : MonoBehaviour  
{  
 [SerializeField]  
 private GameObject **backgroundLeft**;  
  
 [SerializeField]  
 private GameObject **backgroundRight**;  
  
 [SerializeField, Range(-1f, 1f)]  
 private float **scrollSpeed**;  
  
 private float maxOffsetX = 12.65f;  
  
 void **Update**()  
 {  
 backgroundLeft.transform.position += new Vector3(-scrollSpeed \* Time.deltaTime \* 10f, 0f, 0f);  
 backgroundRight.transform.position += new Vector3(-scrollSpeed \* Time.deltaTime \* 10f, 0f, 0f);  
  
 if (backgroundLeft.transform.position.x <= -maxOffsetX)  
 {  
 backgroundLeft.transform.position = new Vector3(maxOffsetX, 0, 0);  
 }  
  
 if (backgroundRight.transform.position.x <= -maxOffsetX)  
 {  
 backgroundRight.transform.position = new Vector3(maxOffsetX, 0, 0);  
 }  
 }  
}

PostFX/ParticleAttachment.cs

*// Credits: richardkettlewell https://forum.unity.com/threads/lwrp-using-2d-lights-in-a-particle-system-emitter.718847/#post-5554201*  
using System.Collections.Generic;  
using UnityEngine;  
  
[RequireComponent(typeof(ParticleSystem))]  
public class **ParticleAttachment** : MonoBehaviour  
{  
 [SerializeField]  
 private GameObject **prefab**;  
  
 private ParticleSystem \_particleSystem;  
 private List<GameObject> instances = new List<GameObject>();  
 private ParticleSystem.Particle[] particles;  
  
 void **Start**()  
 {  
 \_particleSystem = GetComponent<ParticleSystem>();  
 particles = new ParticleSystem.Particle[\_particleSystem.main.maxParticles];  
 }  
  
 void **LateUpdate**()  
 {  
 int count = \_particleSystem.GetParticles(particles);  
  
 while (instances.Count < count)  
 instances.Add(Instantiate(prefab, \_particleSystem.transform));  
  
 bool worldSpace = (\_particleSystem.main.simulationSpace == ParticleSystemSimulationSpace.**World**);  
 for (int i = 0; i < instances.Count; i++)  
 {  
 if (i >= count)  
 {  
 instances[i].SetActive(false);  
 }  
 else  
 {  
 if (worldSpace)  
 {  
 instances[i].transform.position = particles[i].position;  
 }  
 else  
 {  
 instances[i].transform.localPosition = particles[i].position;  
 instances[i].SetActive(true);  
 }  
 }  
 }  
 }  
}

UI/ButtonWrapper.cs

using UnityEngine;  
using UnityEngine.UI;  
  
public class **ButtonWrapper** : MonoBehaviour  
{  
 private AudioManager audioManager;  
 private Button btn;  
  
 void **Start**()  
 {  
 audioManager = GameObject.FindGameObjectWithTag("Audio").GetComponent<AudioManager>();  
 btn = GetComponent<Button>();  
 btn.onClick.AddListener(OnButtonClick);  
 }  
  
 void OnButtonClick()  
 {  
 audioManager.PlaySound("select");  
 }  
}