

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

Кафедра информационных систем и технологий  
(наименование)

ОТЧЁТ ПО ПРАКТИКЕ  
ЗАЩИЩЁН С ОЦЕНКОЙ

РУКОВОДИТЕЛЬ

старший преподаватель

должность, уч. степень, звание

Отлично



подпись, дата

С. Ю. Гуков

инициалы, фамилия

ОТЧЁТ ПО ПРАКТИКЕ

вид практики

учебная

тип практики

на тему индивидуального задания

The Bird

разработка видеоприглашения

выполнен

Томчуком Григорием Сергеевичем

фамилия, имя, отчество обучающегося в творительном падеже

по направлению подготовки

09.03.02  
код

Информационные системы и технологии  
наименование направления

наименование направления

направленности

03  
код

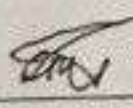
Информационные технологии  
наименование направленности

в сфере

наименование направленности

Обучающийся группы №

4326  
номер

 01.06.24  
подпись, дата

Г. С. Томчук

инициалы, фамилия

Санкт-Петербург 2024



### ИНДИВИДУАЛЬНОЕ ЗАДАНИЕ

на прохождение учебной практики обучающегося направления  
подготовки/ специальности 09.03.02 Информационные системы и технологии

1. Фамилия, имя, отчество обучающегося: Томчук Григорий Сергеевич

2. Група: 4326

3. Тема индивидуального задания:

разработка видеопары - Floppa The Bird

4. Исходные данные:

5. Содержание отчетной документации:

5.1 индивидуальное задание:

5.2 отчёт, включающий в себя:

- титульный лист;
- материалы о выполнении индивидуального задания (содержание определяется группой);
- выводы по результатам практики;
- список использованных источников.

6. Срок представления отчета на кафедру: «    » 20   

Руководитель практики

старший преподаватель  
должность, уч. степень, звание

ПОДПИСЬ ДИТА

С. Ю. ГУКОВ

инициалы, фамилия

Задание принял к исполнению обучающийся

BLIND

## ПОЛИКС

Г. С. Томчук

ИНИЦИАЛЫ, ФАМИЛИЯ

## СОДЕРЖАНИЕ

ВВЕДЕНИЕ.....	4
1 Тема и описание игры.....	7
2 Пользовательская документация.....	8
2.1 Установка и запуск.....	8
2.2 Управление.....	8
2.3 Выход из игры.....	9
3 Техническая документация.....	10
4 Тестирование программы.....	16
5 Описание назначения и процесса загрузки проекта на GitHub.....	17
ЗАКЛЮЧЕНИЕ.....	19
СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ.....	21
ПРИЛОЖЕНИЕ.....	22

## 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 мне предстоит улучшить свои профессиональные навыки и освоить новые технологии, выполняя следующие задачи:

- Изучение документаций, специальной литературы и другой научно-технической информации.
- Сбор, обработка и систематизация 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.

## 2 Пользовательская документация

### 2.1 Установка и запуск

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

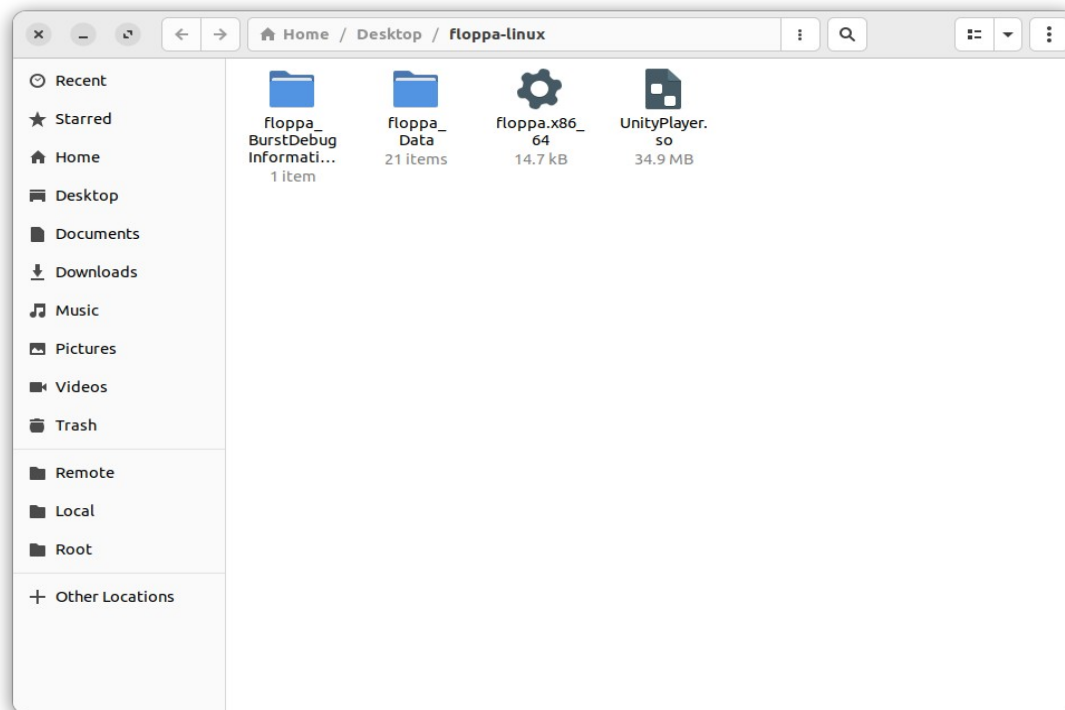


Рисунок 1 – Директория игры на Linux

### 2.2 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

### 2.3 Выход из игры

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



Рисунок 3 – Главное меню игры

### 3 Техническая документация

Для работы с проектом необходим Unity Editor версии 2021.3.17f1. При использовании редактора другой версии могут возникнуть проблемы (например несоответствие версий используемых пакетов и вытекающие из этого ошибки).

Описание основных функций, методов, классов и полей приведено в README-файле репозитория проекта.

#### **LogicManager.cs**

Логический менеджер. Работает со счетом и рекордами, переключает сцены.

Тип возвращаемого значения	Имя и назначение
void	Start() Инициализирует начальные параметры и состояния игры при запуске сцены. Устанавливает текущую сцену, воспроизводит соответствующую музыку и обновляет текстовые элементы для отображения счетов.
void	AddScore(int scoreToAdd) Увеличивает текущий счет игрока на заданное значение, обновляет отображение счета и при необходимости обновляет рекордный счет.
void	ResetHighScore() Сбрасывает рекордный счет игрока до нуля и обновляет отображение рекорда.
void	SwitchToTitleScreen() Переключает текущую сцену на экран заглавного меню.
void	SwitchToMainScene() Переключает текущую сцену на основную игровую сцену.
void	RestartScene() Перезапускает текущую сцену. Доступна как команда контекстного меню в Unity Editor.
void	SetGameOver() Устанавливает состояние "игра окончена", отключает HUD и активирует экран окончания игры.
void	ExitGame() Закрывает приложение. Полезно для завершения игры на устройствах или в тестовой среде.
void	OnInspectorGUI() Переопределяет интерфейс инспектора для объекта LogicManager в Unity Editor, позволяя настроить отображение полей в зависимости от выбранного пресета.
void	SerializePropertyField(string propertyPath) Упрощает отображение полей свойства в инспекторе Unity Editor. Эта функция используется для отрисовки полей свойств в зависимости от выбранного пресета.

Рисунок 4 – Описание методов LogicManager

Класс	Назначение
LogicManagerEditor	Вложенный класс, используемый в редакторе Unity для настройки пользовательского интерфейса инспектора. Этот класс позволяет редактировать сериализованные поля LogicManager в зависимости от выбранного значения preset.

Тип поля	Название и назначение
Preset	preset поле типа Preset, указывающее режим текущей сцены (TitleScreen или MainScene).
Text	scoreText текстовое поле для отображения текущего счета игрока.
Text	highScoreText текстовое поле для отображения рекорда игрока.
GameObject	hud объект интерфейса для отображения HUD.
GameObject	gameOverScreen объект интерфейса для отображения экрана "Game Over".
AudioManager	audioManager ссылка на компонент AudioManager для управления звуком в игре.
int	playerScore текущий счет игрока.
int	playerHighScore рекорд игрока.
static string	currentSceneName имя текущей сцены, сохраненное в статическом поле для доступа из других экземпляров менеджера.
string	previousSceneName имя предыдущей сцены.
enum	Preset перечисление, определяющее режим текущей сцены.

Рисунок 5 – Описание классов и полей LogicManager

## AudioManager.cs

Аудио менеджер. Отвечает за проигрывание звуков и музыки.

Тип возвращаемого значения	Имя и назначение
void	Awake() Метод, вызываемый при инициализации объекта. Проверяет, существует ли уже экземпляр AudioManager. Если нет, устанавливает текущий экземпляр как одиночный (singleton) и предотвращает его уничтожение при смене сцен. Если экземпляр уже существует, уничтожает текущий объект.
void	PlaySound(string name) Воспроизводит звуковой эффект, заданный по имени. Ищет звук в списке sounds. Если звук не найден, выводит сообщение об ошибке в консоль. Для звука "flap" изменяет высоту тона в диапазоне 0.9-1.1 и воспроизводит его через soundSourceB. Для остальных звуков изменяет высоту тона в том же диапазоне и воспроизводит их через soundSourceA.
void	PlayMusic(string name) Воспроизводит музыкальную дорожку, заданную по имени. Ищет трек в списке tracks. Если трек не найден, выводит сообщение об ошибке в консоль. Если трек найден, устанавливает его в качестве текущего клипа musicSource и воспроизводит.

Класс	Назначение
Sound	Представляет звуковой объект с именем и аудиоклипом. Используется для хранения звуковых эффектов и музыкальных дорожек в списках sounds и tracks.

Тип поля	Название и назначение
AudioSource	soundSourceA Аудиоисточник для воспроизведения звуковых эффектов с изменяемой высотой тона.
AudioSource	soundSourceB Дополнительный аудиоисточник для воспроизведения звуковых эффектов с изменяемой высотой тона. Используется для звука "flap", чтобы избежать наложения звуков.
AudioSource	musicSource Аудиоисточник для воспроизведения музыкальных треков.
List<Sound>	sounds Список звуковых эффектов, доступных для воспроизведения.
List<Sound>	tracks Список музыкальных треков, доступных для воспроизведения.
static AudioManager	objectInstance Статическое поле для хранения экземпляра AudioManager. Обеспечивает реализацию паттерна одиночка (singleton).

Рисунок 6 – Описание AudioManager



## PipeGenerator.cs

Генератор объектов препятствий со случайной позицией по Y.

Тип возвращаемого значения	Имя и назначение
void	Start() Вызывается один раз при запуске сцены. Выполняет первоначальное создание труб с помощью метода SpawnPipes().
void	Update() Вызывается каждый кадр. Увеличивает таймер на значение Time.deltaTime. Если таймер превышает значение spawnRate, создает новые трубы, сбрасывая таймер.
void	SpawnPipes() Создает трубы в случайной позиции по оси Y в диапазоне, заданном с использованием heightOffset. Вычисляет случайную высоту в диапазоне от transform.position.y - heightOffset до transform.position.y + heightOffset и создает экземпляр объекта pipes в этой позиции.

Тип поля	Название и назначение
GameObject	pipes Префаб труб, который будет создаваться и размещаться в сцене.
float	spawnRate Частота появления труб в секундах. Определяет, через какой промежуток времени будут создаваться новые трубы.
float	heightOffset Смещение по высоте, которое используется для определения случайной позиции по оси Y при создании труб.
float	timer Таймер, отслеживающий время, прошедшее с момента последнего появления труб. Сбрасывается при достижении значения spawnRate.

Рисунок 7 – Описание PipeGenerator

## **BirdController.cs**

Контроллер игрока.

Тип возвращаемого значения	Имя и назначение
void	Start() Инициализирует компоненты и переменные при запуске сцены. Находит и сохраняет ссылки на LogicManager, AudioManager, а также компоненты Rigidbody2D, SpriteRenderer и Animator.
void	Update() Вызывается каждый кадр. Проверяет ввод пользователя (нажатие клавиш Space или Mouse0) для выполнения прыжка, если птица жива. Также проверяет позицию птицы по оси Y и вызывает метод Die(), если птица выходит за пределы допустимой зоны (deadZoneY).
void	Flap() Воспроизводит звук "flap" и анимацию "BirdFlap", устанавливает вертикальную скорость птицы, чтобы она "взлетела" вверх.
void	Die() Устанавливает флаг isAlive в false, переворачивает спрайт птицы по оси Y и вызывает метод SetGameOver() из LogicManager для отображения экрана окончания игры.
void	OnCollisionEnter2D(Collision2D collision) Вызывается при столкновении с другим 2D-коллайдером. Воспроизводит звук "hit" и вызывает метод Die().

Рисунок 8 – Описание методов BirdController

Тип поля	Название и назначение
float	flapForce Сила, с которой птица взлетает вверх при нажатии клавиши.
LogicManager	logicManager Ссылка на объект LogicManager, управляющий логикой игры.
AudioManager	audioManager Ссылка на объект AudioManager, отвечающий за воспроизведение звуков.
Rigidbody2D	rb Ссылка на компонент Rigidbody2D для управления физикой птицы.
SpriteRenderer	sprite Ссылка на компонент SpriteRenderer для управления визуальным отображением птицы.
Animator	anim Ссылка на компонент Animator для управления анимациями птицы.
bool	isAlive Флаг, указывающий, жива ли птица.
float	deadZoneY Пределы по оси Y, за которые птица не должна выходить. Если птица выходит за эти пределы, вызывается метод Die().

Рисунок 9 – Описание полей BirdController

## ParticleAttachment.cs

Компонент, который привязывает экземпляры префабов к частицам системы ParticleSystem. Он следит за состоянием частиц и управляет позициями экземпляров префабов, чтобы они совпадали с позициями соответствующих частиц. Используется в объекте частиц-звезд, так как частицы в форме игровых объектов позволяют работать, например, с уровнем излучаемого света этих частиц.

Тип возвращаемого значения	Имя и назначение
void	Start() Инициализирует компонент ParticleSystem и массив particles, настраивая его размер в соответствии с максимальным количеством частиц системы.
void	LateUpdate() Обновляет позиции и активность экземпляров префабов в соответствии с текущими позициями частиц. Получает количество активных частиц. Добавляет недостающие экземпляры префабов в список instances, если их меньше, чем активных частиц. Устанавливает позиции экземпляров префабов в зависимости от системы координат (мировые или локальные). Деактивирует лишние экземпляры префабов, если их больше, чем активных частиц.

Тип поля	Название и назначение
GameObject	prefab Префаб объекта, который будет создаваться и привязываться к частицам.
ParticleSystem	_particleSystem Ссылка на компонент ParticleSystem, с которым работает скрипт.
List<GameObject>	instances Список экземпляров префабов, привязанных к частицам.
ParticleSystem.Particle[]	particles Массив частиц, используемый для хранения текущего состояния частиц системы.

Рисунок 10 – Описание ParticleAttachment

## 4 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, генерация препятствий, подсчет очков и система рекордов.
2. Тестирование реакции игры на нажатия экрана или кнопки, проверка плавности полета игрока.
3. Проверка появления препятствий 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 количество ошибок в программном коде Floppa The Bird.

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



## 5 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 следующем:

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`.

**floppa-the-bird**
Public

Unpin
Unwatch 1
Fork 0
Star 1

main
1 Branch
0 Tags

Add file
Code

**grigorijtomczuk**
Create a logo, update README.md
9dcf4ad · last year
11 Commits

.vscode	Update settings.json	last year
Assets	Create a logo, update README.md	last year
Images	Create a logo, update README.md	last year
Packages	Import Unity project, update .gitignore	last year
ProjectSettings	Import Unity project, update .gitignore	last year
.gitignore	Import Unity project, update .gitignore	last year
LICENSE	Initial commit	last year
README.md	Create a logo, update README.md	last year

README
MIT license

A primitive 2D Unity game about a bird named Floppa.

game
csharp
unity
flappy-bird
unity2d
flappy-bird-clone
unity2d-game

Readme
MIT license
Activity

1 star
1 watching
0 forks

Languages

C# 90.4%
ShaderLab 9.6%

Рисунок 11 – 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();
}
}

```

```

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 abrupt 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();
    }
}
}

```



```

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();
    }
}

```

```
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);
        }
    }
}
```

```

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);
    }
}

```

```
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);
        }
    }
}
```

```

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);
        }
    }
}

```

*// 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);
                }
            }
        }
    }
}
```



```
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");
    }
}
```