**КАФЕДРА ВЕБ-ТЕХНОЛОГИЙ И КОМПЬЮТЕРНОГО МОДЕЛИРОВАНИЯ**

**МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ**

**БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ**

**МЕХАНИКО-МАТЕМАТИЧЕСКИЙ ФАКУЛЬТЕТ**

**Создание игры на С#/Unity**

Дипломная работа

Суражева Кирилла Леонидовича,

студента 3 курса, специальность 1-31 81 06 веб-программирование и интернет-технологии

Научный руководитель: доцент, кандидат физ.-мат. наук Кравчук Анжелика Ивановна

Минск,2021

Содержание

[ВВЕДЕНИЕ 4](#_Toc72483435)

[ГЛАВА 1. ФУНКЦИИ, ЖАНРЫ, СОЗДАНИЕ ИГР 6](#_Toc72483436)

[1.1 Функции игр 6](#_Toc72483437)

[1.2 Жанры игр 6](#_Toc72483438)

[1.3 Процесс создания игр 9](#_Toc72483439)

[1.4 Технологическая часть 11](#_Toc72483440)

[ГЛАВА 2. РАЗРАБОТКА ИГРЫ 14](#_Toc72483441)

[2.1 Базовая настройка 14](#_Toc72483442)

[2.3 Управление птицей, её физика, анимация 15](#_Toc72483443)

[2.4 Генерация уровня, сложность 15](#_Toc72483444)

[2.5 Интерфейс в игре 15](#_Toc72483445)

[2.6 Экран загрузки 17](#_Toc72483446)

[2.7 Главное меню 17](#_Toc72483447)

[2.8 Звук 17](#_Toc72483448)

[2.9 Ожидание игрока 18](#_Toc72483449)

[2.10 Диаграмма классов 18](#_Toc72483450)

[ЗАКЛЮЧЕНИЕ 19](#_Toc72483451)

[СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 20](#_Toc72483452)

[ПРИЛОЖЕНИЕ А 21](#_Toc72483453)

[А1 Скрипт GameHandler.cs 21](#_Toc72483454)

[А2 Скрипт GameAssets.cs 21](#_Toc72483455)

[А3 Скрипт Bird.cs 22](#_Toc72483456)

[А4 Скрипт SpriteAnimator.cs 25](#_Toc72483457)

[А5 Скрипт Level.cs 28](#_Toc72483458)

[А6 Скрипт ScoreWindow.cs 39](#_Toc72483459)

[А7 Скрипт Score.cs 40](#_Toc72483460)

[А8 Скрипт Loader.cs 42](#_Toc72483461)

[А9 Скрипт LoaderUpdate.cs 43](#_Toc72483462)

[А10 Скрипт MainMenuWindow.cs 43](#_Toc72483463)

[А11 Скрипт SoundManager.cs 44](#_Toc72483464)

[А12 Скрипт WaitingToStartWindow.cs 45](#_Toc72483465)

ВВЕДЕНИЕ

Тип работы - опытно-экспериментальный

Целью данной курсовой работы является написание 2D игры в программе Unity.Название игры: Flappy Bird Simulator

Для чего вообще нужны игры? Игра - это совместная деятельность людей и она способствует улучшению коммуникативных качеств участников. Игра способствует улучшению межличностных отношений, развивает мотивационный аспект личности человека, помогает выстраивать систему жизненных ценностей, развивает умение ставить цели, навык планирования, готовность к совершению выборов, навыки саморегуляции, умение соотносить свои действия с действиями других людей.

Описание: Данная игра создавалась с нуля. Flappy Bird Simulator — игра-платформер для мобильных устройств, в которой игрок с помощью касаний экрана должен контролировать полёт птицы между рядами зелёных труб, не задевая их. Была реализована для Android и PC.

Задачи:

* Изучить функции игр
* Изучить классификацию игр по жанрам
* Изучить процесс создания игр
* Реализовать главное меню с названием игры (Flappy Bird Simulator) и с кнопками Play (при её нажатии игра начинается) и Quit (при её нажатии происходит выход из игры)
* Реализовать игровой уровень, а именно:
  + Нижнюю границу уровня, пол (когда птичка касается её, она умирает)
  + Движение птицы (прыжок, гравитация, движение вправо)
  + Столкновение объектов друг с другом (трубы и пол не сталкиваются друг с другом, а птичка с трубами или полом – сталкивается)
  + Генерация новых препятствий и удаления старых, а также увеличение сложности по мере продвижения по уровню
  + Появление новых облаков и удаление старых
  + Смерть птички (когда она сталкивается с трубой или полом, она умирает, и появляется меню с надписью Game Over, рекордом и текущим результатом, а также кнопки Retry (при нажатии на неё уровень перезапускается) и Main Menu (при нажатии на неё происходит возврат на главное меню)
  + Звук (при совершении следующих действий: нажатии на кнопку, прыжке, обновлении текущего счёта, смерти должен появляться определённый звук)
  + Текущий счёт и сохранение рекорда (когда птичка пролетает через отверстие в трубе, счёт увеличивается на 1, и если счёт становится больше, чем текущий рекорд, он записывается в рекорд, причём рекорд хранится в отдельном файле и сохраняется даже после выхода из игры
  + Ожидание игрока (время как бы остановлено, пока игрок не нажмёт на кнопку прыжка)

Актуальность работы заключается в том, что в наши дни огромное количество самых разных по интересам людей частенько поигрывают в компьютерные игры, причем это не только скучающие школьники или прогульщики-студенты. Среди игроков встречаются и бизнесмены, и политики, и домохозяйки, и инженеры, и художники - в целом абсолютно разные люди. Всех их объединяет одно - желание испытать в виртуальных мирах что-то новое, неизведанное, попытать удачу и получить наслаждение как от игрового процесса, так и от достигнутых в игре результатов.

Актуальность создания состоит в том, что создатели игр могут создавать свои миры, людей, флору и фауну, различных существ и связь всего этого друг с другом. И эти творения способны развлекать, развивать и обладают силой затрагивать человеческие сердца

# ГЛАВА 1. ФУНКЦИИ, ЖАНРЫ, СОЗДАНИЕ ИГР

## 1.1 Функции игр

Игра наряду с трудом и ученьем - один из основных видов деятельности человека, удивительный феномен нашего существования.

По определению, игра - это вид деятельности в условиях ситуаций, направленных на воссоздание и усвоение общественного опыта, в котором складывается и совершенствуется самоуправление поведением.

В человеческой практике игровая деятельность выполняет такие функции:

 развлекательную;

 коммуникативную: освоение диалектики общения;

 самореализации в игре как полигоне человеческой практики;

 игротерапевтическую: преодоление различных трудностей, возникающих в других видах жизнедеятельности;

 диагностическую: выявление отклонений от нормативного поведения, самопознание в процессе игры;

 функцию коррекции: внесение позитивных изменений в структуру личностных показателей;

 межнациональной коммуникации: усвоение единых для всех людей социально-культурных ценностей;

 социализации: включение в систему общественных отношений, усвоение норм человеческого общежития [1].

## 1.2 Жанры игр

1.Action

Этот жанр означает использование виртуального пространства, в котором есть мишень и стрелок. Может быть человек, например, и какое—либо транспортное средство (танк, космический корабль, самолет). Пространство игры или двухмерное, или трехмерное. Основная цель «экшена» – поразить как можно большее количество мишеней. В качестве примера: Left 4 Dead, третий Doom, четвертый Devil May Cry.

2.MMORPG

Речь идет о многопользовательском игровом жанре, в котором предусмотрен отдельный виртуальный мир. В этом пространстве приходится уживаться большому количеству различных персонажей. Чаще всего, речь идет о мире фэнтези. Здесь, как известно, можно встретить кого угодно – фантастические персонажи. У каждого из них есть определенные способности, собственная цель, а также роль в игре. Игрок имеет возможность стать тем, кто ему больше по душе.

Сравнивая со стандартной RPG, геймплей MMORPG не исчезает никуда (не имеет значения, находится пользователь в игре или нет). В качестве примера: World Of Tanks, World of Warcraft, второй LineAge.

3.RPG

Категория ролевых игр может иметь как развлекательные, так и учебные разработки. Игра начинается в придуманном мире. В основе может использоваться сюжет художественного вида или от начала до конца придуманный разработчиками игры. В данной вымышленной реальности геймеру, как правило (но не всегда), приходится решать какую—нибудь определенную задачу. Во время игры необходимо достичь определенных навыков. В виртуальной реальности можно увидеть адекватную реакцию на различные действия, а также поступки. Для примера: второй Mass Effect, King’s Bounty, четвертый Fallout.

4.Аркады

Достаточно примитивный жанр игры, несмотря на это, очень популярной в наши дни. «Перекочевал» на компьютеры с различных игровых автоматов, а также консолей. Аркадными бывают гонки, файтинги и еще большое количество разновидностей игр. Суть обычной аркады основывается на том, что геймеру необходимо пройти определенный уровень за минимальный период времени. Во время игры приходится преодолевать различные препятствия, а также собирать бонусы. В качестве примера: Super Mario, Super Meat Boy, Pac—Man.

5.Головоломки

О смысле этой игры можно легко догадаться из названия. Геймеру нужно разобраться с какой—нибудь непростой задачей. К примеру, речь может идти о составлении предмета или выбора из большого количества вариантов одного правильного решения. Также возможны и другие вариации.

Известно, головоломки являются самым старым видом игр на Земле. В древнюю китайскую игру «Танграм» коротали время свыше 3000 лет назад. В качестве примера: Crayon Physics Deluxe, World of Goo.

6.Гонки

Смысл этого жанра понятен каждому геймеру. Игрок должен участвовать в различных гонках. Из возможных вариантов могут быть соревнования «Формулы 1», турниры стритрейсеров, а также демонстрация мастер—класса на более сложных транспортных средствах – космические корабли или самолеты. Гонки могут быть как трехмерными, так и двухмерными.

7.Приключения

Данный вид игр отличается наличием специального литературного сюжета. Именно он используется для создания геймплея. Герой, которым придется стать игроку, будет жить в данном мире. Во время игры он будет решать большое количество различных интеллектуальных задач, в том числе и головоломок.

Сравнивая с RPG, в данном случае геймеру не придется случайно взаимодействовать с другими персонажами игры – есть четкий сюжет, за рамки которого нельзя выходить. В качестве примера: Syberia, Fahrenheit, The Longest Journey.

8.Симуляторы

Речь идет о полноценной и очень точной имитации управления определенным объектом (может быть машина, танк или самолет). Сначала симуляторы создавались для военных солдат, чтобы обучать их управлять техническими устройствами. Однако прогресс стал причиной того, что обучающее программное обеспечение заняло свое отдельное место в индустрии игровых разработок.

Сравнивая с гонками, в симуляторах реализовано более точное управление и нет необходимости соревноваться. В качестве примера: Trainz, пятый Silent Hunter.

9.Спортивные

Данная категория игр представляет собой соревнования различных команд. В основном, речь идет о футбольном, баскетбольном, хоккейном или теннисном матчах. Геймерам придется взять шефство над командой: формировать ее состав и привести к победе. В качества примера: серия PES и серия Fifa.

10.Стратегии

Наверное, мы сейчас говорим о наиболее интеллектуальном виде игровых разработок. С самого начала геймер получает возможность создать собственный мир или, в крайнем случае, государство, где придется не только жить, а и управлять. Это не так легко и приятно, как можно подумать сначала. Необходимо детально продумать инфраструктуру, тонко организовать все планы, создать для жителей подходящие условия проживания, дать требуемое количество рабочих мест. В этом деле будут мешать жадные правители других государств – им так и хочется заполучить вашу территорию, поэтому ее нужно будет защищать.

В качестве примера: третий Warcraft, Age of Empires III.

11.Файтинги

Данный жанр игр подходит тем, кто желает ощутить себя в роли «настоящего мужика». Геймера ждет нескончаемое число рукопашных боев с самыми различными врагами. Каждый бой проходит на арене. Покинуть ее удастся, если закончить игру. То есть, игровой мир в этом случае сильно ограничен. В качестве примера: игры серии Mortal Kombat или Tekken [2].

## 1.3 Процесс создания игр

1. Концептирование (Concept)

На этом первом шаге команда придумывает концепцию игры, и проводит начальную проработку игрового дизайна. Главная цель данного этапа – это геймдизайнерская документация, включающая в себя Vision (развернутый документ, описывающий игру, как конечный бизнес-продукт) и Concept Document (начальную проработку всех аспектов игры).

В продуктовой документации геймдизайнер формулирует и сохраняет свои идеи. Исполнителю документация позволяет правильно понимать свои задачи по реализации продукта. Тестировщик четко видит, что и как тестировать. Для Продюсера/ПМа эта документация предоставляет материал для формирования планов и контроля выполнения задач. Инвестор же (особенно на ранних этапах) получает понимание, на что именно он выделяет средства.

Принципиально важно, чтобы вся проектная и продуктовая документация поддерживалась в актуальном состоянии на всех этапах развития проекта. Для её эффективного использования и обновления правильно использовать специальные инструменты. Например, использование Confluence для ведения геймдизайнерской документации сильно упрощает процесс параллельного внесения изменений несколькими участниками разработки, а также позволяет всем членам команды оперативно получать любую актуальную информацию, касающуюся продукта и всех его изменений.

Среди ключевых принципов формирования продуктовой документации стоит отметить: структурированность, защищенность от разночтений, полное описание продукта, регулярную актуализацию.

2. Прототипирование (Prototyping)

Важный этап проектирования любой игры – это создание прототипа. То, что хорошо выглядит «на бумаге», совершенно не обязательно будет интересно в реальности. Прототип реализуется для оценки основного игрового процесса, проверки различных гипотез, проведения тестов игровых механик, для проверки ключевых технических моментов.

Очень важно на этапе создания прототипа реализовывать только то, что нужно проверить и в сжатые сроки. Прототип должен быть простым в реализации, т.к. после достижения поставленных перед ним целей, он должен быть «выкинут». Серьёзная ошибка начинающих разработчиков – нести временную инфраструктуру и «костыли» реализации кода в основной проект.

3. Вертикальный срез (Vertical Slice)

Цель Вертикального среза – получить минимально возможную полноценную версию игры, включающую в себя полностью реализованный основной игровой процесс. При этом высокое качество проработки обязательно нужно воплотить только для тех игровых элементов, которые существенно влияют на восприятие продукта. При этом все базовые фичи игры присутствуют как минимум в черновом качестве. Реализован минимальный, но достаточный для воплощения полноценного игрового процесса набор контента (один уровень или одна локация).

4. Производство контента (Content production)

На этом этапе производится достаточное количество контента для первого запуска на внешнюю аудиторию. Реализуются все фичи, запланированные к закрытому бета-тестированию. Это наиболее продолжительный этап, который может занимать, для крупных клиентских проектов год и более.

На этом этапе задействуется наибольшее количество специалистов, которые занимаются производством всего основного наполнения игры. Художники создают все графические ресурсы, геймдизайнеры настраивают баланс и заполняют конфиги, программисты реализуют и полируют все фичи.

5. Friends & Family / CBT (закрытое бета-тестирование)

На этапе CBT продукт впервые демонстрируется достаточно широкой публике, хотя и лояльной продукту или компании. Среди наиболее важных задач на этом этапе выступают: поиск и исправление гейм-дизайнерских ошибок, проблем игровой логики и устранение критических багов. На этом этапе в игре присутствуют уже все ключевые фичи, создано достаточно контента для полноценной игры продолжительное время, настроены сбор и анализ статистики. Тестирование идет по тест-плану, проводятся стресс-тесты уже с привлечением реальных игроков.

6. Soft Launch / OBT (открытый бета-тест)

На этом этапе продолжается тестирование игры, но уже на широкой аудитории. Идет оптимизация под большие нагрузки. Игра должна быть готова для приема большого трафика. В игре реализован биллинг и принимаются платежи.

На этом этапе полностью завершается разработка новых фичей. Происходит feature freeze, программисты перестают реализовывать что-то новое, а полностью переключаются на отладку и тюнинг имеющихся фичей. Геймдизайнеры, продюсер и аналитики делают выводы из собранной на CBT статистики и проверяют эффективность монетизации.

При этом, к началу этапа должна полностью функционировать инфраструктура проекта: сайт, группы соц. сетях, каналы привлечения (User Acquisition), поддержка пользователей.

7. Release

Ключевая цель – это получение прибыли. Базовый применяемый для оценки прибыльности критерий: количество денег, принесенных в среднем одним игроком за все время (LTV aka lifetime value), должно превосходить расходы на привлечение этого игрока (CPI aka cost per install).

На этом этапе должно быть полностью отлажено оперирование продукта (техническая поддержка, работа с комьюнити), соблюдаются маркетинговые и финансовые планы, ведутся работы по улучшению финансовых показателей, активно отрабатываются каналы по привлечению трафика.

Команда разработки на этом этапе занимается исправлением технических багов, выявляемых в процессе эксплуатации и оптимизацией продукта. Геймдизайнеры занимаются тонкой настройкой геймплея под реальную ситуацию в игровом мире (особенно актуально для ММО проектов). Также реализует различные внутриигровые фичи, поддерживающие новые монетизационные схемы. И конечно идет разработка и интеграция в продукт нового контента, поддерживающего интерес игроков [3].

## Технологическая часть

1.Ресурсы разработки

Игра была разработана с помощью туториала CodeMonkey [4].

2.Описание системных требований

Игра запустится на любом современном компьютере или Android-е

3. Описание пользовательского интерфейса

В главном меню игры есть два пункта: Play и Quit. Play позволяет начать игру, Quit – выйти из неё. После нажатия на кнопку Start игра сразу же начинается. В игре вы можете нажать Space, чтобы прыгнуть, и Esc, чтобы выйти в главное меню. Также в игре есть интерфейс, показывающий результаты и текущий рекорд (он сохраняется при выходе из игры) (см. рисунок 1.1)



Рисунок 1.1 – Результат и рекорд

Также в игре есть интерфейсное меню, появляющееся после смерти персонажа (см рисунок 1.2), с вариантами:

1)Retry (перезапуск уровня)

2)Main menu (вернуться на главное меню)



Рисунок 1.2 – Меню, появляющееся после смерти персонажа

4. Руководство к использованию и эксплуатации

После того как вы установите на компьютер игру, на рабочем столе запустите ярлык с игрой (см рисунок 1.3)



Рисунок 1.3 - Ярлык игры

После запуска файла вы увидите окно (см рисунок 1.4)



Рисунок 1.4 - Меню игры

5. Результаты экспериментальной проверки

При проверке проекта никаких отклонений от поставленной задачи не наблюдалось. В целом игра готова к эксплуатации. В результате экспериментальной проверки никаких ошибок выявлено не было. Игра работает без сбоев и нарушений.

ГЛАВА 2. РАЗРАБОТКА ИГРЫ

## 2.1 Базовая настройка

Для того, чтобы создать игровой проект, необходимо: находясь в главном окне Unity, создать новый проект, назвать его Flappy Bird Simulator (название игры), из Templates выбрать 2D, выбрать местоположение и нажать Create (см. Рисунок 2.1)

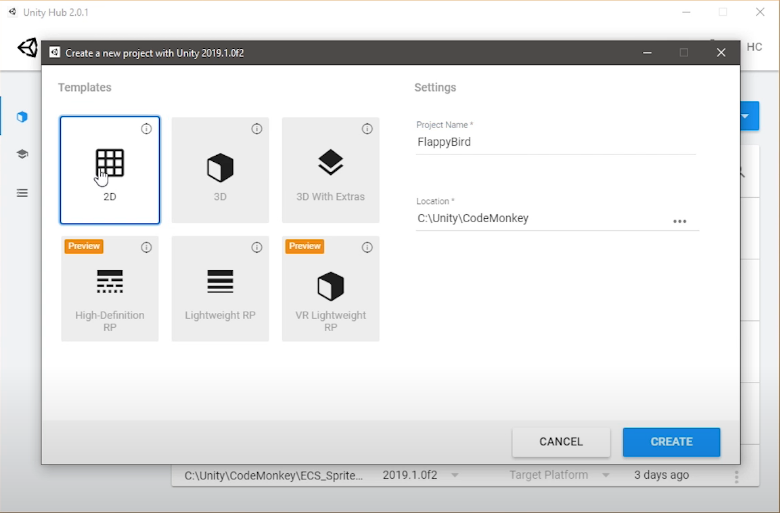


Рисунок 2.1 – Создание нового проекта в Unity

Далее был создан объект GameHandler и к нему был привязан скрипт GameHandler.cs (cм. Приложение А1), отвечающий за отладочную загрузку в начале

Далее была выполнены импорт и настройка текстур объектов, использующихся в игре (см. Рисунок 2.2)

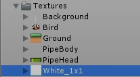


Рисунок 2.2 – Текстуры объектов, использующихся в игре

Также на GameScene был добавлен спрайт птицы (см. Рисунок 2.3)



Рисунок 2.3 – Спрайт птицы

Далее был создан скрипт GameAssets.cs (см. Приложение А2) для хранения всех объектов для лёгкого доступа, привязанный к новому пустому объекту GameAssets

## 2.3 Управление птицей, её физика, анимация

Был создан и привязан к объекту Bird скрипт управления и физики птицы Bird.cs (см. Приложение А3), отвечающий за перемещение птицы, физику и логику столкновений (если птица сталкивается с полом или препятствием, то появляется интерфейс Game Over, с результатом и меню с вариантами: 1) Restart, 2) Return to the main menu:

Также был разработан скрипт SpriteAnimator.cs (см. Приложение А4), отвечающий за анимацию птицы

## 2.4 Генерация уровня, сложность

Был создан объект Level в GameScene и к нему был привязан скрипт Level.cs (см. Приложение А5), отвечающий за генерацию новых препятствий, облаков в поле зрения камеры, а также удаление тех, что находятся вне поля зрения камеры

## 2.5 Интерфейс в игре

В редакторе Unity в GameScene были созданы следующие объекты (см. рисунки 2.4, 2.5, 2.6)

На объект ScoreWindow был наложен скрипт ScoreWindow.cs (см. Приложение А6), который отвечает за изменение числа очков вверху экрана, а если оно становится выше, чем HIGHSCORE, то HIGHSCORE меняется после смерти персонажа на это число.

Также был создан файл Score.cs (см. Приложение А7), который содержит основную логику ScoreWindow.cs

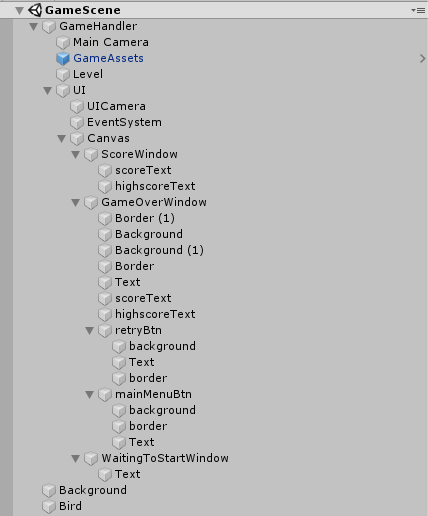


Рисунок 2.4 – Начальные игровые объекты

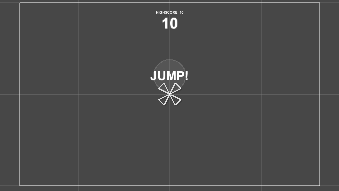


Рисунок 2.5 – Интерфейс игрового уровня

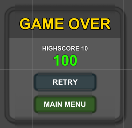


Рисунок 2.6 – Экран смерти

## 2.6 Экран загрузки

Была создана сцена Loading (для того, чтобы было видно, когда игра грузится) (см. Рисунок 2.7)

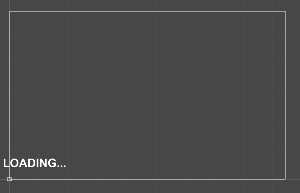


Рисунок 2.7 – Экран загрузки

Далее были разработаны скрипты Loader.cs (см. Приложение А8) и LoaderUpdate.cs (см. Приложение А9), отвечающие за логику экрана загрузки

Скрипт LoaderUpdate.cs был привязан к объекту LoaderUpdate.cs

## 2.7 Главное меню

Была создана сцена MainMenu (см. Рисунок 2.8)



Рисунок 2.8 – Главное меню

К объекту MainMenuWindow был привязан скрипт MainMenuWindow.cs (см. Приложение А10), отвечающий за логику главного меню (если нажимаешь Play, игра начинается, если Quit, происходит выход из игры)

## 2.8 Звук

Также был создан скрипт SoundManager.cs (см. Приложение А11), который отвечает за весь звук в игре

## 2.9 Ожидание игрока

Для большего удобства был добавлен экран ожидания игрока и был разработан скрипт WaitingToStartWindow.cs (см. Приложение А12) (время в игре как бы остановлено, пока игрок не нажмёт кнопку прыжка

## 2.10 Диаграмма классов

С помощью средств Visual Studio была получена диаграмма классов приложения (см. Рисунок 2.9)

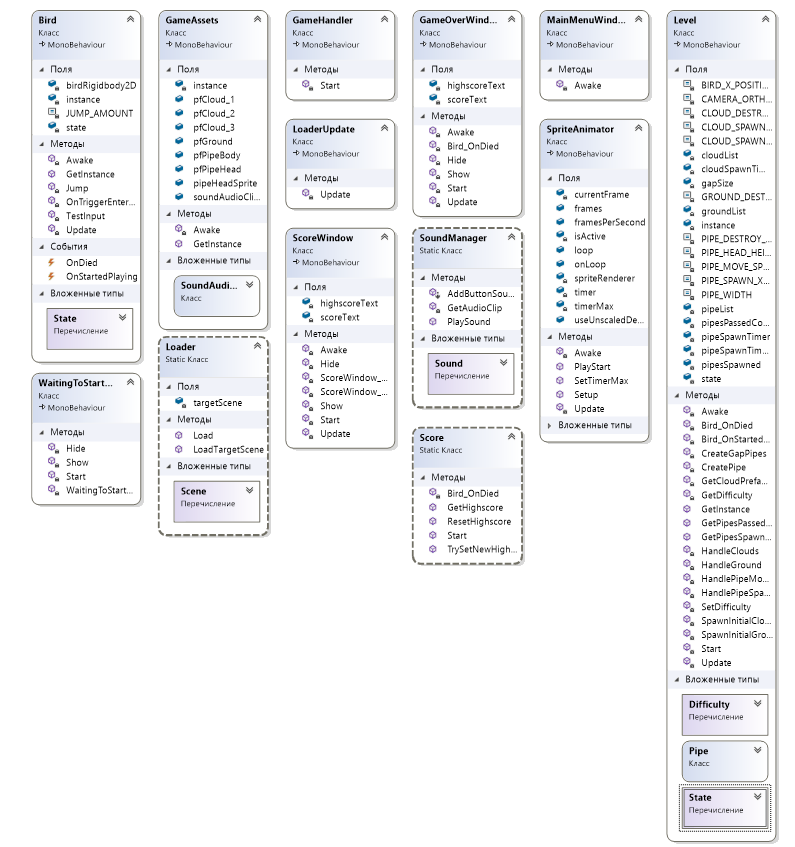


Рис. 2.9 – Диаграмма классов

ЗАКЛЮЧЕНИЕ

В настоящей дипломной работе были:

* Изучены функции игр
* Изучена классификация игр по жанрам
* Изучен процесс создания игр
* Реализовано главное меню с названием игры (Flappy Bird Simulator) и с кнопками Play (при её нажатии игра начинается) и Quit (при её нажатии происходит выход из игры)
* Реализован игровой уровень, а именно:
  + Нижняя граница уровня, пол
  + Движение птицы
  + Столкновение объектов друг с другом
  + Генерация новых препятствий и удаления старых, а также увеличение сложности по мере продвижения по уровню
  + Появление новых облаков и удаление старых
  + Смерть птички
  + Звук
  + Текущий счёт и сохранение рекорда
  + Ожидание игрока

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Игровая деятельность. Игра наряду с трудом и ученьем - один из основных видов деятельности человека, удивительный феномен нашего существования [Электронный ресурс]. – 2014. – Режим доступа: <http://www.un.org/russian/documen/basicdoc/statut.htm>. –Дата доступа: 16.05.2021.
2. Жанры игр — какие они бывают [Электронный ресурс]. – 2015. – Режим доступа:<https://dmagame.ru/zhanry-igr-kakie-oni-byvajut/#:~:text=Жанры%20игр%20—%20какие%20они%20бывают> –Дата доступа: 16.05.2021.
3. Семь этапов создания игры: от концепта до релиза [Электронный ресурс]. – 2017. – Режим доступа: <https://habr.com/ru/company/miip/blog/308286/> –Дата доступа: 16.05.2021.
4. Making Flappy Bird in Unity 2019 [Электронный ресурс]. – 2019. – Режим доступа: <https://unitycodemonkey.com/video.php?v=b5Wpni9KPik> –Дата доступа: 16.05.2021.

# ПРИЛОЖЕНИЕ А

**Код программы**

## А1 Скрипт GameHandler.cs

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

using CodeMonkey;

using CodeMonkey.Utils;

public class GameHandler : MonoBehaviour {

private void Start() {

Debug.Log("GameHandler.Start");

Score.Start();

}

}

## А2 Скрипт GameAssets.cs

using System;

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

public class GameAssets : MonoBehaviour {

private static GameAssets instance;

public static GameAssets GetInstance() {

return instance;

}

private void Awake() {

instance = this;

}

public Sprite pipeHeadSprite;

public Transform pfPipeHead;

public Transform pfPipeBody;

public Transform pfGround;

public Transform pfCloud\_1;

public Transform pfCloud\_2;

public Transform pfCloud\_3;

public SoundAudioClip[] soundAudioClipArray;

[Serializable]

public class SoundAudioClip {

public SoundManager.Sound sound;

public AudioClip audioClip;

}

}

## А3 Скрипт Bird.cs

using System;

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

using CodeMonkey;

public class Bird : MonoBehaviour {

private const float JUMP\_AMOUNT = 90f;//константа для контроля высоты прыжка

private static Bird instance;

public static Bird GetInstance() {

return instance;

}

public event EventHandler OnDied;

public event EventHandler OnStartedPlaying;

private Rigidbody2D birdRigidbody2D;

private State state;

private enum State {

WaitingToStart,

Playing,

Dead

}

private void Awake() {

instance = this;

birdRigidbody2D = GetComponent<Rigidbody2D>();

birdRigidbody2D.bodyType = RigidbodyType2D.Static;

state = State.WaitingToStart;

}

private void Update() {

switch (state) {

default:

case State.WaitingToStart:

if (TestInput()) {

// Start playing

state = State.Playing;

birdRigidbody2D.bodyType = RigidbodyType2D.Dynamic;

Jump();

if (OnStartedPlaying != null) OnStartedPlaying(this, EventArgs.Empty);

}

break;

case State.Playing:

if (TestInput()) {

Jump();

}

// Rotate bird as it jumps and falls

transform.eulerAngles = new Vector3(0, 0, birdRigidbody2D.velocity.y \* .15f);

break;

case State.Dead:

break;

}

}

private bool TestInput() {

return

Input.GetKeyDown(KeyCode.Space) ||

Input.GetMouseButtonDown(0) ||

Input.touchCount > 0;

}

private void Jump() {

birdRigidbody2D.velocity = Vector2.up \* JUMP\_AMOUNT;

SoundManager.PlaySound(SoundManager.Sound.BirdJump);

}

private void OnTriggerEnter2D(Collider2D collider) {

birdRigidbody2D.bodyType = RigidbodyType2D.Static;

SoundManager.PlaySound(SoundManager.Sound.Lose);

if (OnDied != null) OnDied(this, EventArgs.Empty);

}

}

## А4 Скрипт SpriteAnimator.cs

using UnityEngine;

using UnityEngine.UI;

using System.Collections;

public class SpriteAnimator : MonoBehaviour {

public Sprite[] frames;

public int framesPerSecond = 30;

public bool loop = true;

public delegate void OnLoopDel();

public OnLoopDel onLoop;

public bool useUnscaledDeltaTime;

private bool isActive = true;

private float timer;

private float timerMax;

private int currentFrame;

private SpriteRenderer spriteRenderer;

private void Awake() {

timerMax = 1f/framesPerSecond;

spriteRenderer = transform.GetComponent<SpriteRenderer>();

if (frames != null) {

spriteRenderer.sprite = frames[0];

} else {

isActive = false;

}

}

private void Update() {

if (!isActive) return;

timer += useUnscaledDeltaTime ? Time.unscaledDeltaTime : Time.deltaTime;

bool newFrame = false;

while (timer >= timerMax) {

timer -= timerMax;

//Next frame

currentFrame = (currentFrame+1) % frames.Length;

newFrame = true;

if (currentFrame == 0) {

//Looped

if (!loop) {

isActive = false;

newFrame = false;

}

if (onLoop != null) {

onLoop();

}

}

}

if (newFrame) {

spriteRenderer.sprite = frames[currentFrame];

}

}

public void Setup(Sprite[] frames, int framesPerSecond) {

this.frames = frames;

this.framesPerSecond = framesPerSecond;

timerMax = 1f/framesPerSecond;

spriteRenderer.sprite = frames[0];

timer = 0f;

PlayStart();

}

public void SetTimerMax(float timerMax) {

this.timerMax = timerMax;

}

public void PlayStart() {

timer = 0;

currentFrame = 0;

spriteRenderer.sprite = frames[currentFrame];

isActive = true;

}

}

## А5 Скрипт Level.cs

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

using CodeMonkey;

using CodeMonkey.Utils;

public class Level : MonoBehaviour {

private const float CAMERA\_ORTHO\_SIZE = 50f;

private const float PIPE\_WIDTH = 7.8f;

private const float PIPE\_HEAD\_HEIGHT = 3.75f;

private const float PIPE\_MOVE\_SPEED = 30f;

private const float PIPE\_DESTROY\_X\_POSITION = -100f;

private const float PIPE\_SPAWN\_X\_POSITION = +100f;

private const float GROUND\_DESTROY\_X\_POSITION = -200f;

private const float CLOUD\_DESTROY\_X\_POSITION = -160f;

private const float CLOUD\_SPAWN\_X\_POSITION = +160f;

private const float CLOUD\_SPAWN\_Y\_POSITION = +30f;

private const float BIRD\_X\_POSITION = 0f;

private static Level instance;

public static Level GetInstance() {

return instance;

}

private List<Transform> groundList;

private List<Transform> cloudList;

private float cloudSpawnTimer;

private List<Pipe> pipeList;

private int pipesPassedCount;

private int pipesSpawned;

private float pipeSpawnTimer;

private float pipeSpawnTimerMax;

private float gapSize;

private State state;

public enum Difficulty {

Easy,

Medium,

Hard,

Impossible,

}

private enum State {

WaitingToStart,

Playing,

BirdDead,

}

private void Awake() {

instance = this;

SpawnInitialGround();

SpawnInitialClouds();

pipeList = new List<Pipe>();

pipeSpawnTimerMax = 1f;

SetDifficulty(Difficulty.Easy);

state = State.WaitingToStart;

}

private void Start() {

Bird.GetInstance().OnDied += Bird\_OnDied;

Bird.GetInstance().OnStartedPlaying += Bird\_OnStartedPlaying;

}

private void Bird\_OnStartedPlaying(object sender, System.EventArgs e) {

state = State.Playing;

}

private void Bird\_OnDied(object sender, System.EventArgs e) {

//CMDebug.TextPopupMouse("Dead!");

state = State.BirdDead;

}

private void Update() {

if (state == State.Playing) {

HandlePipeMovement();

HandlePipeSpawning();

HandleGround();

HandleClouds();

}

}

private void SpawnInitialClouds() {

cloudList = new List<Transform>();

Transform cloudTransform;

cloudTransform = Instantiate(GetCloudPrefabTransform(), new Vector3(0, CLOUD\_SPAWN\_Y\_POSITION, 0), Quaternion.identity);

cloudList.Add(cloudTransform);

}

private Transform GetCloudPrefabTransform() {

switch (Random.Range(0, 3)) {

default:

case 0: return GameAssets.GetInstance().pfCloud\_1;

case 1: return GameAssets.GetInstance().pfCloud\_2;

case 2: return GameAssets.GetInstance().pfCloud\_3;

}

}

private void HandleClouds() {

// Handle Cloud Spawning

cloudSpawnTimer -= Time.deltaTime;

if (cloudSpawnTimer < 0) {

// Time to spawn another cloud

float cloudSpawnTimerMax = 6f;

cloudSpawnTimer = cloudSpawnTimerMax;

Transform cloudTransform = Instantiate(GetCloudPrefabTransform(), new Vector3(CLOUD\_SPAWN\_X\_POSITION, CLOUD\_SPAWN\_Y\_POSITION, 0), Quaternion.identity);

cloudList.Add(cloudTransform);

}

// Handle Cloud Moving

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

Transform cloudTransform = cloudList[i];

// Move cloud by less speed than pipes for Parallax

cloudTransform.position += new Vector3(-1, 0, 0) \* PIPE\_MOVE\_SPEED \* Time.deltaTime \* .7f;

if (cloudTransform.position.x < CLOUD\_DESTROY\_X\_POSITION) {

// Cloud past destroy point, destroy self

Destroy(cloudTransform.gameObject);

cloudList.RemoveAt(i);

i--;

}

}

}

private void SpawnInitialGround() {

groundList = new List<Transform>();

Transform groundTransform;

float groundY = -47.5f;

float groundWidth = 192f;

groundTransform = Instantiate(GameAssets.GetInstance().pfGround, new Vector3(0, groundY, 0), Quaternion.identity);

groundList.Add(groundTransform);

groundTransform = Instantiate(GameAssets.GetInstance().pfGround, new Vector3(groundWidth, groundY, 0), Quaternion.identity);

groundList.Add(groundTransform);

groundTransform = Instantiate(GameAssets.GetInstance().pfGround, new Vector3(groundWidth \* 2f, groundY, 0), Quaternion.identity);

groundList.Add(groundTransform);

}

private void HandleGround() {

foreach (Transform groundTransform in groundList) {

groundTransform.position += new Vector3(-1, 0, 0) \* PIPE\_MOVE\_SPEED \* Time.deltaTime;

if (groundTransform.position.x < GROUND\_DESTROY\_X\_POSITION) {

// Ground passed the left side, relocate on right side

// Find right most X position

float rightMostXPosition = -100f;

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

if (groundList[i].position.x > rightMostXPosition) {

rightMostXPosition = groundList[i].position.x;

}

}

// Place Ground on the right most position

float groundWidth = 192f;

groundTransform.position = new Vector3(rightMostXPosition + groundWidth, groundTransform.position.y, groundTransform.position.z);

}

}

}

private void HandlePipeSpawning() {

pipeSpawnTimer -= Time.deltaTime;

if (pipeSpawnTimer < 0) {

// Time to spawn another Pipe

pipeSpawnTimer += pipeSpawnTimerMax;

float heightEdgeLimit = 10f;

float minHeight = gapSize \* .5f + heightEdgeLimit;

float totalHeight = CAMERA\_ORTHO\_SIZE \* 2f;

float maxHeight = totalHeight - gapSize \* .5f - heightEdgeLimit;

float height = Random.Range(minHeight, maxHeight);

CreateGapPipes(height, gapSize, PIPE\_SPAWN\_X\_POSITION);

}

}

private void HandlePipeMovement() {

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

Pipe pipe = pipeList[i];

bool isToTheRightOfBird = pipe.GetXPosition() > BIRD\_X\_POSITION;

pipe.Move();

if (isToTheRightOfBird && pipe.GetXPosition() <= BIRD\_X\_POSITION && pipe.IsBottom()) {

// Pipe passed Bird

pipesPassedCount++;

SoundManager.PlaySound(SoundManager.Sound.Score);

}

if (pipe.GetXPosition() < PIPE\_DESTROY\_X\_POSITION) {

// Destroy Pipe

pipe.DestroySelf();

pipeList.Remove(pipe);

i--;

}

}

}

private void SetDifficulty(Difficulty difficulty) {

switch (difficulty) {

case Difficulty.Easy:

gapSize = 50f;

pipeSpawnTimerMax = 1.4f;

break;

case Difficulty.Medium:

gapSize = 40f;

pipeSpawnTimerMax = 1.3f;

break;

case Difficulty.Hard:

gapSize = 33f;

pipeSpawnTimerMax = 1.1f;

break;

case Difficulty.Impossible:

gapSize = 24f;

pipeSpawnTimerMax = 1.0f;

break;

}

}

private Difficulty GetDifficulty() {

if (pipesSpawned >= 24) return Difficulty.Impossible;

if (pipesSpawned >= 12) return Difficulty.Hard;

if (pipesSpawned >= 5) return Difficulty.Medium;

return Difficulty.Easy;

}

private void CreateGapPipes(float gapY, float gapSize, float xPosition) {

CreatePipe(gapY - gapSize \* .5f, xPosition, true);

CreatePipe(CAMERA\_ORTHO\_SIZE \* 2f - gapY - gapSize \* .5f, xPosition, false);

pipesSpawned++;

SetDifficulty(GetDifficulty());

}

private void CreatePipe(float height, float xPosition, bool createBottom) {

// Set up Pipe Head

Transform pipeHead = Instantiate(GameAssets.GetInstance().pfPipeHead);

float pipeHeadYPosition;

if (createBottom) {

pipeHeadYPosition = -CAMERA\_ORTHO\_SIZE + height - PIPE\_HEAD\_HEIGHT \* .5f;

} else {

pipeHeadYPosition = +CAMERA\_ORTHO\_SIZE - height + PIPE\_HEAD\_HEIGHT \* .5f;

}

pipeHead.position = new Vector3(xPosition, pipeHeadYPosition);

// Set up Pipe Body

Transform pipeBody = Instantiate(GameAssets.GetInstance().pfPipeBody);

float pipeBodyYPosition;

if (createBottom) {

pipeBodyYPosition = -CAMERA\_ORTHO\_SIZE;

} else {

pipeBodyYPosition = +CAMERA\_ORTHO\_SIZE;

pipeBody.localScale = new Vector3(1, -1, 1);

}

pipeBody.position = new Vector3(xPosition, pipeBodyYPosition);

SpriteRenderer pipeBodySpriteRenderer = pipeBody.GetComponent<SpriteRenderer>();

pipeBodySpriteRenderer.size = new Vector2(PIPE\_WIDTH, height);

BoxCollider2D pipeBodyBoxCollider = pipeBody.GetComponent<BoxCollider2D>();

pipeBodyBoxCollider.size = new Vector2(PIPE\_WIDTH, height);

pipeBodyBoxCollider.offset = new Vector2(0f, height \* .5f);

Pipe pipe = new Pipe(pipeHead, pipeBody, createBottom);

pipeList.Add(pipe);

}

public int GetPipesSpawned() {

return pipesSpawned;

}

public int GetPipesPassedCount() {

return pipesPassedCount;

}

/\*

\* Represents a single Pipe

\* \*/

private class Pipe {

private Transform pipeHeadTransform;

private Transform pipeBodyTransform;

private bool isBottom;

public Pipe(Transform pipeHeadTransform, Transform pipeBodyTransform, bool isBottom) {

this.pipeHeadTransform = pipeHeadTransform;

this.pipeBodyTransform = pipeBodyTransform;

this.isBottom = isBottom;

}

public void Move() {

pipeHeadTransform.position += new Vector3(-1, 0, 0) \* PIPE\_MOVE\_SPEED \* Time.deltaTime;

pipeBodyTransform.position += new Vector3(-1, 0, 0) \* PIPE\_MOVE\_SPEED \* Time.deltaTime;

}

public float GetXPosition() {

return pipeHeadTransform.position.x;

}

public bool IsBottom() {

return isBottom;

}

public void DestroySelf() {

Destroy(pipeHeadTransform.gameObject);

Destroy(pipeBodyTransform.gameObject);

}

}

}

## А6 Скрипт ScoreWindow.cs

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

using UnityEngine.UI;

public class ScoreWindow : MonoBehaviour {

private Text highscoreText;

private Text scoreText;

private void Awake() {

scoreText = transform.Find("scoreText").GetComponent<Text>();

highscoreText = transform.Find("highscoreText").GetComponent<Text>();

}

private void Start() {

highscoreText.text = "HIGHSCORE: " + Score.GetHighscore().ToString();

Bird.GetInstance().OnDied += ScoreWindow\_OnDied;

Bird.GetInstance().OnStartedPlaying += ScoreWindow\_OnStartedPlaying;

Hide();

}

private void ScoreWindow\_OnStartedPlaying(object sender, System.EventArgs e) {

Show();

}

private void ScoreWindow\_OnDied(object sender, System.EventArgs e) {

Hide();

}

private void Update() {

scoreText.text = Level.GetInstance().GetPipesPassedCount().ToString();

}

private void Hide() {

gameObject.SetActive(false);

}

private void Show() {

gameObject.SetActive(true);

}

}

## А7 Скрипт Score.cs

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

public static class Score {

public static void Start() {

//ResetHighscore();

Bird.GetInstance().OnDied += Bird\_OnDied;

}

private static void Bird\_OnDied(object sender, System.EventArgs e) {

TrySetNewHighscore(Level.GetInstance().GetPipesPassedCount());

}

public static int GetHighscore() {

return PlayerPrefs.GetInt("highscore");

}

public static bool TrySetNewHighscore(int score) {

int currentHighscore = GetHighscore();

if (score > currentHighscore) {

// New Highscore

PlayerPrefs.SetInt("highscore", score);

PlayerPrefs.Save();

return true;

} else {

return false;

}

}

public static void ResetHighscore() {

PlayerPrefs.SetInt("highscore", 0);

PlayerPrefs.Save();

}

}

## А8 Скрипт Loader.cs

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

using UnityEngine.SceneManagement;

public static class Loader {

public enum Scene {

GameScene,

Loading,

MainMenu,

}

private static Scene targetScene;

public static void Load(Scene scene) {

SceneManager.LoadScene(Scene.Loading.ToString());

targetScene = scene;

}

public static void LoadTargetScene() {

SceneManager.LoadScene(targetScene.ToString());

}

}

## А9 Скрипт LoaderUpdate.cs

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

public class LoaderUpdate : MonoBehaviour {

private void Update() {

Loader.LoadTargetScene();

}

}

## А10 Скрипт MainMenuWindow.cs

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

using CodeMonkey.Utils;

public class MainMenuWindow : MonoBehaviour {

private void Awake() {

transform.Find("playBtn").GetComponent<Button\_UI>().ClickFunc = () => { Loader.Load(Loader.Scene.GameScene); };

transform.Find("playBtn").GetComponent<Button\_UI>().AddButtonSounds();

transform.Find("quitBtn").GetComponent<Button\_UI>().ClickFunc = () => { Application.Quit(); };

transform.Find("quitBtn").GetComponent<Button\_UI>().AddButtonSounds();

}

}

## А11 Скрипт SoundManager.cs

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

using CodeMonkey.Utils;

public static class SoundManager {

public enum Sound {

BirdJump,

Score,

Lose,

ButtonOver,

ButtonClick,

}

public static void PlaySound(Sound sound) {

GameObject gameObject = new GameObject("Sound", typeof(AudioSource));

AudioSource audioSource = gameObject.GetComponent<AudioSource>();

audioSource.PlayOneShot(GetAudioClip(sound));

}

private static AudioClip GetAudioClip(Sound sound) {

foreach (GameAssets.SoundAudioClip soundAudioClip in GameAssets.GetInstance().soundAudioClipArray) {

if (soundAudioClip.sound == sound) {

return soundAudioClip.audioClip;

}

}

Debug.LogError("Sound " + sound + " not found!");

return null;

}

public static void AddButtonSounds(this Button\_UI buttonUI) {

buttonUI.MouseOverOnceFunc += () => PlaySound(Sound.ButtonOver);

buttonUI.ClickFunc += () => PlaySound(Sound.ButtonClick);

}

}

## А12 Скрипт WaitingToStartWindow.cs

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

public class WaitingToStartWindow : MonoBehaviour {

private void Start() {

Bird.GetInstance().OnStartedPlaying += WaitingToStartWindow\_OnStartedPlaying;

}

private void WaitingToStartWindow\_OnStartedPlaying(object sender, System.EventArgs e) {

Hide();

}

private void Hide() {

gameObject.SetActive(false);

}

private void Show() {

gameObject.SetActive(true);

}

}