Министерство образования Республики Беларусь
Учреждение образования
БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Факультет компьютерного проектирования

Кафедра проектирования информационных компьютерных систем

Дисциплина "Объектно-ориентированное программирование"

К защите допустить:
Руководитель курсовой работы
старший преподаватель
кафедры
_____ А.В.Михалькевич
26.07.2025

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

к курсовой работе

на тему

Развлекательное приложение “Falling Ball”

БГУИР КР 1-40 05 01-10 № 122 ПЗ

Студент (подпись студента)

Курсовая работа
представлена на проверку
26.07.2025
_____
(подпись студента)

2025

Реферат

БГУИР КР 1-40 05 01-10 № 122 ПЗ, гр. 814303

, Развлекательное приложение “Falling Ball”, Минск: БГУИР - 2025.

Пояснительная записка 60266 с., 10 рис., 1 табл.

Ключевые слова: Unity, гипер-казуальная, игровая индустрия

Предмет Объектно-ориентированное программирование, А.В.Михалькевич

В данной курсовой работе была разработана гипер-казуальная игра “Falling Ball”. Ссылка на github: https://github.com/kostyaon/Falling-ball

-

Содержание

Введение

Видеоигры – один из самых быстрорастущих рынков в мире. Доходы компаний, производящих и реализующих видеоигры, составляют миллиарды долларов. На сегодняшний день по доходам игровая индустрия опережает киноиндустрию, а также догоняет спортивный рынок. Видеоигры окружают нас везде. Мы можем играть в них в любой точке мире, причем необязательно иметь с собой компьютер. Достаточно мобильного телефона, который всегда находится рядом с нами. Каждый год мы совершенствуем наши устройства. Сегодня производительность мобильного телефона позволяет запускать игры, в которые совсем недавно мы играли на своих компьютерах. Мобильный рынок превосходит компьютерный и консольный. Сегодня 57% от всего рынка занимают мобильные игры. Популярность мобильных игр заключается в том, что вы можете играть в игры в любом месте, при этом вам не нужно громоздкое устройство. Почему же люди играют в видеоигры? Ведь, по сути, сама игра ничего не стоит, если в нее не играют. Играя в игры люди получают опыт, при этом сама игра не является опытом. Игра – инструмент создания опыта. Если ваша игра не создает никакого опыта, то ее ценность можно приравнять к нулю. На сегодняшний день люди предпочитают играть в гипер-казуальные игры. Если классифицировать игры по жанрам, беря за основу их игровую механику, то гипер-казуальные игры – это не новый жанр, а категория, которая имеет похожую модель монетизации, схожую аудиторию и длину игровой сессии, как казуальные игры. Основные черты гипер-казуальных игр, следующие: мгновенный доступ к контенту, минимализм в геймплее и дизайне, простая модель монетизации. В последнее время граница между казуальными и гипер-казуальными играми расплывается, так как постепенно гипер-казуальные игры меняются, добавляя все более привычные элементы казуальных игр, такие как сохранение прогресса игрока или смена уровней. Курсовой проект позволил изучить возможности игрового движка Unity3D, приобрести навыки работы с языком

программирования C#, а также изучить основные аспекты работы геймдизайнера.

1 ОПИСАНИЕ ПРОЕКТА

1.1Концепция игры

Жанр игры – казуальные игры. Категория – гипер-казуальные игры. Почему именно гипер-казуальная игра? Гипер-казуальные игры появились совсем недавно в 2017 году. Но уже сегодня по многим параметрам они превосходят казуальные игры. Проведем сравнение по следующим параметрам:

Игровые сессии (их количество и длительность)
Удержание
Количество ежедневных активных пользователей (DAU)

Рисунок 1.1 показывает количество игровых сессий и их длительность (Arcade – гипер-казуальные игры, Casual - казуальные).

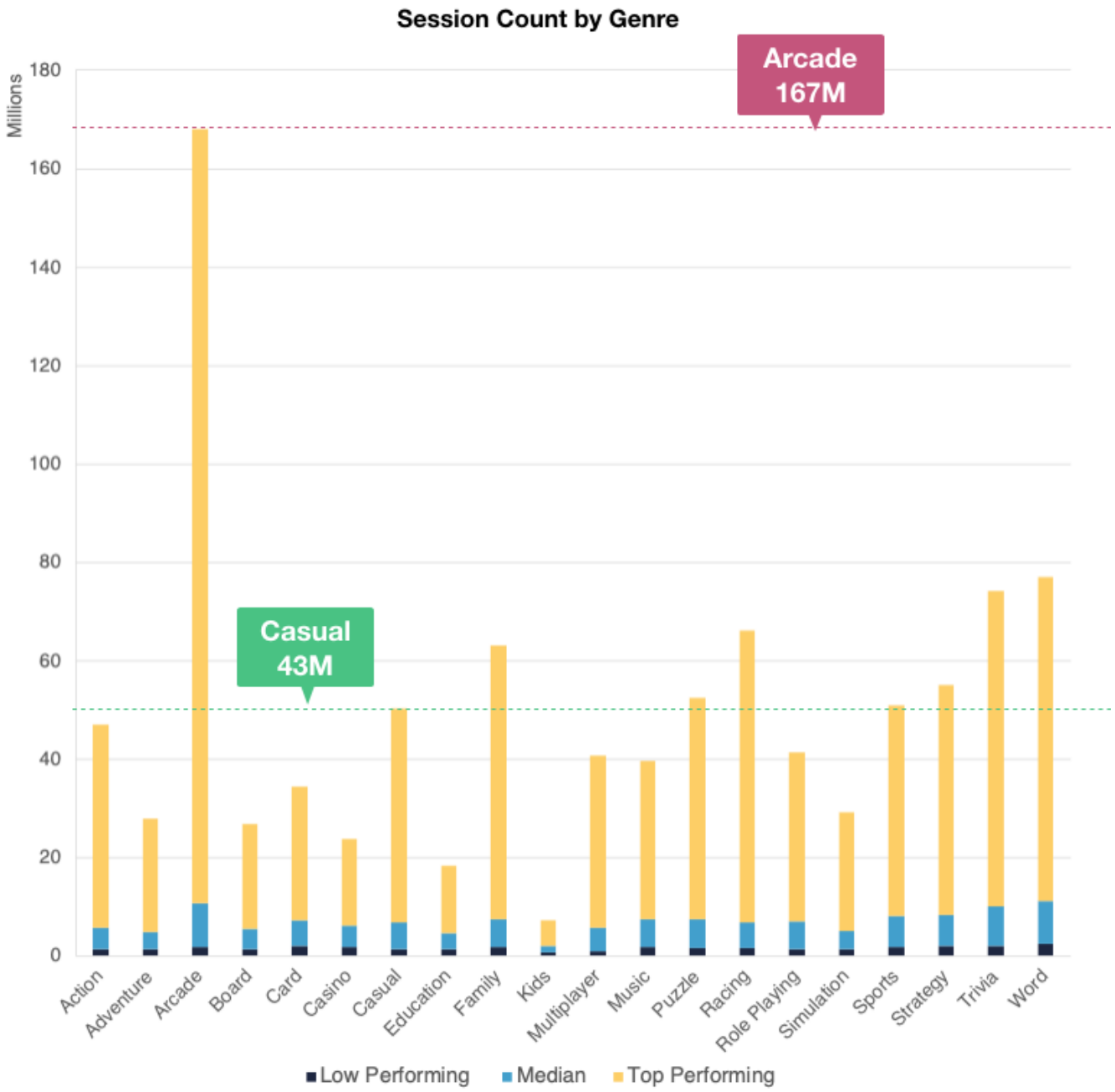

Рисунок 1.1 – Количество и длительность игровых сессий

По количеству игровых сессий гипер-казуальные игры в 4 раза превосходят казуальные, однако длительность игровой сессии в 2 раза меньше. Если сравнивать медианную игровую сессию индустрии в целом, то она составляет 14 минут 30 секунд. В гипер-казуальных играх данный показатель находится на уровне 6 минут 42 секунд.

Однако для гипер-казуальных игр это отличный показатель, так как суть таких игр и есть большое количество игровых сессий, при их небольшой длительности.

Если просуммировать длительность игровых сессий за целый год, то получится, что в гипер-казуальных играх люди проводят в 5 раз больше времени, по сравнению со всеми остальными категориями. Что говорит о популярности данной категории.

Рисунок 1.2 показывает количество ежедневных активных пользователей (DAU) (гипер-казуальные игры – розовый цвет, казуальные - зеленый).

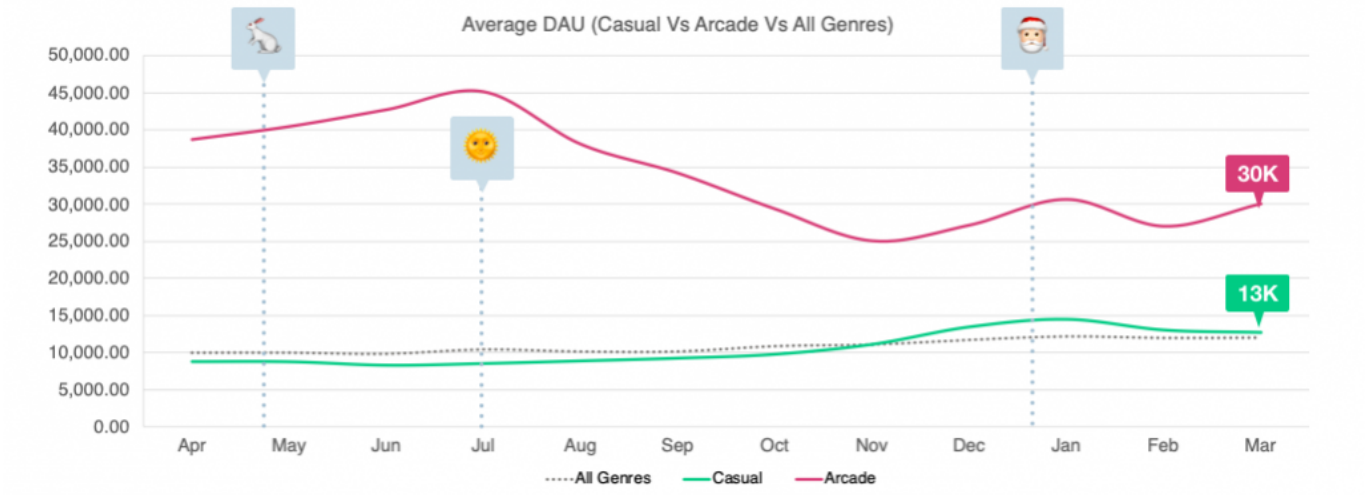

Рисунок 1.2 – Количество ежедневных активных пользователей

Показатель DAU гипер-казуальных игр является наивысшим в мобильном рынке. Однако исходя из рисунка 1.2, мы можем заметить отрицательную динамику DAU гипер-казуальных игр, а у казуальных наоборот.

Однако среднее DAU у популярных гипер-казуальных игр – 94 тысячи человек, что в 2,5 раза больше, чем у казуальных игр.

Показатель D7 – показатель удержания на 7 дней после установки приложения. Данный показатель оказывает непосредственное влияние на доход. Хорошими значениями для D7 считаются 20%.

Данный показатель у казуальных и гипер-казуальных игр равен 15%. Его можно улучшить путем изменения основного цикла игры и рекламной монетизацией.

Исходя из этих данных была выбрана категория гипер-казуальных игр, так как она позволит охватить большое количество пользователей.

1.2 Описание основных элементов игры

Гипер-казуальная игра – игра с затягивающим геймплеем и минималистичным артом. Правила должны быть простыми, а цели понятные. Доступ к контенту должен быть максимально быстрым.

Определим основные составляющие любой игры. Их 4, и называют их совокупность - элементной тетрадой. В элементную тетраду входят: механика, история, эстетика и технология. Все эти элементы связаны между собой. Разработчики считают, что самым важным является технология, геймдизайнеры – механика, художники – эстетика, писатели – история. В игре все элементы одинаково важны и для нее нужны все четыре элемента. Однако не все элементы являются заметными. Самым заметным элементом является эстетика, за ней посередине располагаются история и механика, а в конце идет технология.

Механика игры. Механика – это правила и процессы игры, она описывает цель и методы, которые помогут достичь данную цель. Механика делают игру – игрой.

В “Falling ball” игрок перемещает игровой объект (мяч) свайпом по экрану. Есть 4 способа передвижения: вверх, вниз, вправо, влево. В начале игры игровой объект начинает свое

падение с высоты.

Игрок зарабатывает игровые очки с помощью попадания мяча в корзины, которые меняют свое положение. Чем больше очков зарабатывает игрок, тем больше становится скорость падения мяча. Если игрок попал в корзину, то ему дается 2 очка, при попадании с более низкой высоты – 3.

После каждой игры игрок получает игровые монеты. В течении игры, игрок может 2 раза тапнуть на экран, что приведет к увеличению очков на 10, при каждом попадании в корзину, в течении 20 секунд.

Если игрок прошел мимо корзины 2 раза, то игра заканчивается и считается общее количество очков. Таким образом главная цель игрока – заработать как можно больше игровых очков, что становится понятно при старте игрового процесса.

```
private IEnumerator SmoothMove(string direction)
{
    Vector3 startBallPosition, endBallPosition;
    float ballTime = 0f;
    float interpolate = 0.1f;
    startBallPosition = endBallPosition = transform.position;
    if (direction == "left")
    {
        endBallPosition = new Vector3(startBallPosition.x - 1.55f, startBallPosition.y, startBallPosition.z);
    }
    if (direction == "right")
    {
        endBallPosition = new Vector3(startBallPosition.x + 1.55f, startBallPosition.y, startBallPosition.z);
    }
    if (direction == "up")
    {
        endBallPosition = new Vector3(startBallPosition.x, startBallPosition.y + 1.75f, startBallPosition.z);
    }
    if (direction == "down")
    {
        endBallPosition = new Vector3(startBallPosition.x, startBallPosition.y - 1.75f, startBallPosition.z);
    }
    while (ballTime < interpolate)
    {
        ballTime += Time.deltaTime;
        FindObjectOfType<AudioManager>().Play("Swipe");
        transform.position = Vector3.Lerp(startBallPosition, endBallPosition, ballTime / interpolate);
        yield return null;
    }
}
```

Рисунок 1.3 – Скрипт движения игрока

История игры. История – последовательность событий, происходящих в игре. В “Falling ball” простая история. Игрок подкидывает мяч высоко вверх, затем мяч летит в низ, и чем больше очков набрал игрок, тем выше он подкинул мяч.

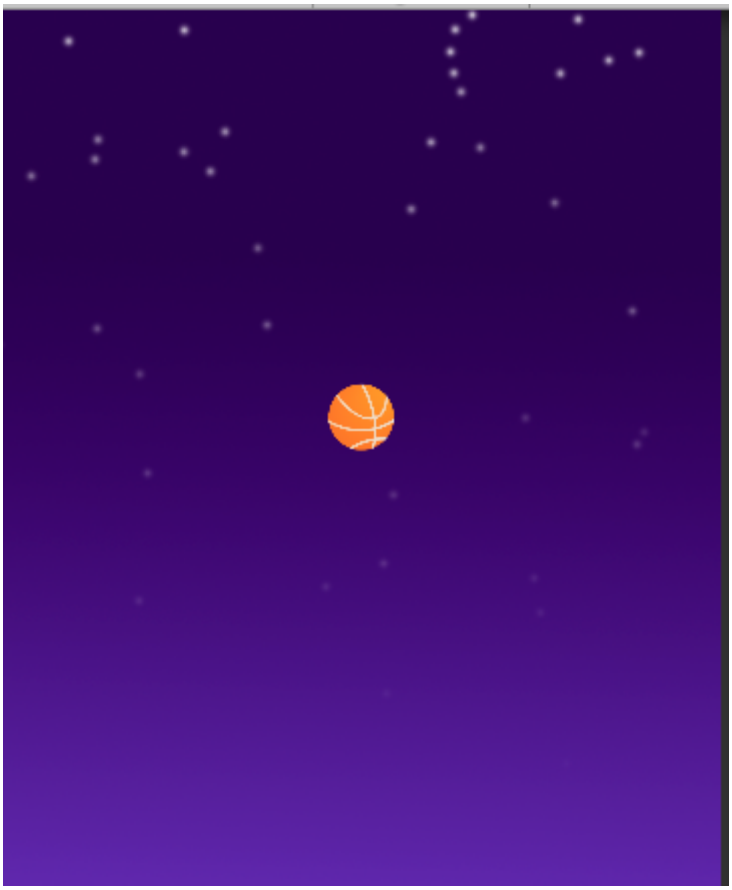

Рисунок 1.4 – Мяч в космосе

Эстетика игры. Эстетика игры отвечает за 2 важные детали: визуальные эффекты и звук.

В “Falling ball”, чтобы подчеркнуть факт падения и взлета мяча, используются звуки космического ветра, ветра, поющих птиц, летающих самолетов, а для увеличения эффекта падения были добавлены звезды, и эффект скорости.

Для того, чтобы понять, что игрок попал по корзине, используется определенный звук, который мотивирует игрока двигаться все дальше. А если промахнулся, то звук ошибки.

Для окончания игры используется звук лопнувшего мяча, который дает понять, что игра была окончена. Также в игре постоянно меняется раскраска фона, что делает игру более красочной и динамичной, создавая при этом продолжение падения.

Когда игрок ставит новый рекорд, то воспроизводятся фанфары. В главном меню располагается облако с количеством очков, которое заработал игрок, и его рекордное количество, что мотивирует игрока сыграть еще раз, чтобы поставить новый рекорд.

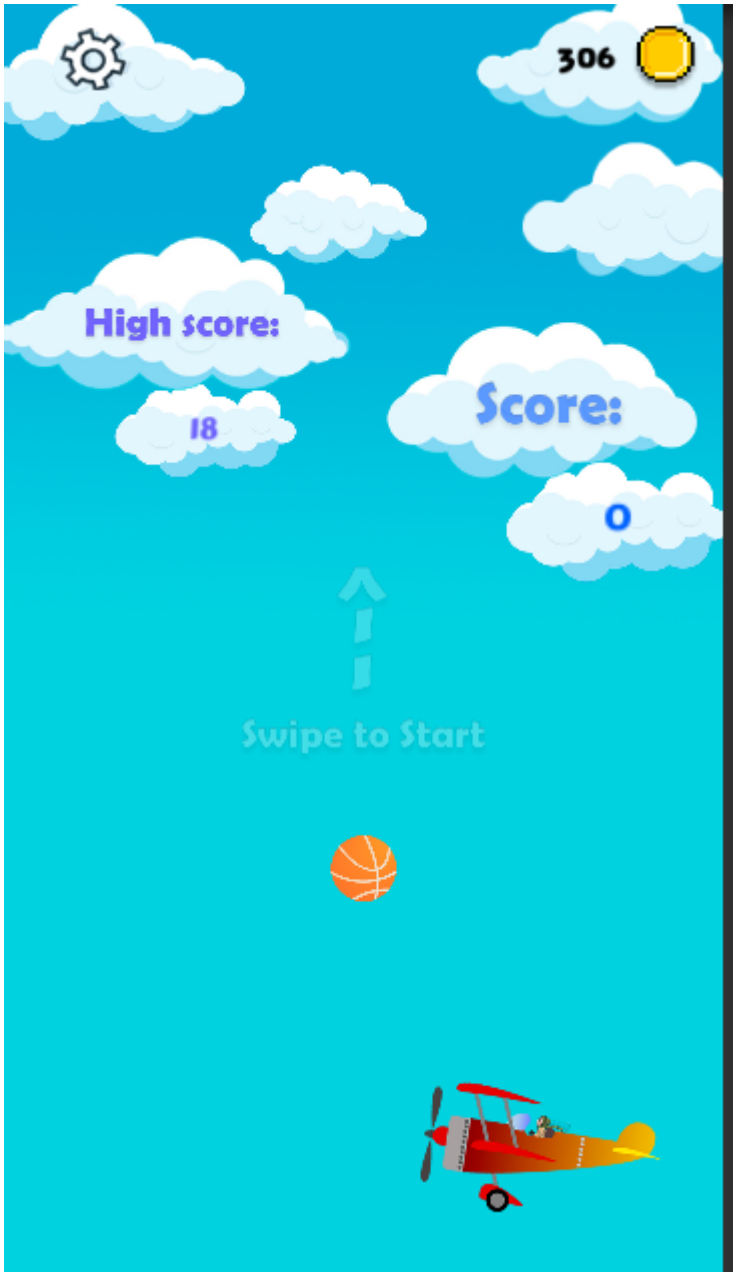

Рисунок 1.5 – Главное меню

Игра сделаны для мобильных устройств, так как данный сегмент рынка имеет большее количество пользователей, а также помогает удерживать игрока, за счет мобильности телефонных аппаратов.

В данную игру можно играть в любом месте, в любое время, в любом возрасте.

2 ОБОСНОВАНИЕ ВЫБОРА ТЕХНОЛОГИЙ

2.1 Технологии программирования, используемые для решения поставленных задач

Для выполнения поставленных задач был выбран язык программирования C#.

C# - язык программирования от Microsoft. Изначально компания планировала выпустить свою версию Java, однако, в итоге, дело дошло до суда. После этого было принято решение о

создании собственного языка, который отвечал их требованиям и развитие которого возможно было контролировать.

Данный язык программирования применяется в различным продуктах:

DirectX – API для программирования под Windows, основное использование – разработка видеоигр.
Mono – проект, посвящённый свободной реализации C# и .NET
XamarinStudio – позволяет создавать мобильные приложения на C#, был создан на основе Mono
Unity – кроссплатформенный игровой движок.

Сегодня C# - современный объектно-ориентированный, типобезопасный язык программирования. Несмотря на то, что C# объектно-ориентированный язык программирования, он также поддерживает компонентно-ориентированное программирование.

Благодаря своим функциям язык программирования C# обеспечивает надежность и устойчивость приложений:

Сборка мусора – автоматически освобождает память, занятую недостижимыми неиспользуемыми объектами
Обработка исключений – предоставляет структурированный и расширяемый подход к обнаружению ошибок и их восстановлению
Типобезопасная структура – делает невозможным чтение из неинициализированных переменных, индексацию массивов за пределами их границ или выполнение непроверенных приведений типов

В C# существует единая система типов, то есть все типы в C# наследуются от одного корневого типа object. Это позволяет всем типам использовать общий набор операций, а также значение любого типа можно передавать и обрабатывать схожим образом.

Одно из главных преимуществ данного языка – управление версиями. Разработчики уделили ей особое внимание для того, чтобы обеспечить совместимость программ и библиотек C# при дальнейшем развитии языка. Это повлияло на такие аспекты разработки языка, как модификаторы virtual и override, правила разрешения перегрузки методов и поддержка явного объявления членов интерфейса.

Каждый язык программирования имеет свои достоинства и недостатки. И C# не исключение. Данный язык программирования имеет следующие достоинства:

Поддержка от Microsoft. Благодаря этому язык развивается быстрыми темпами
Красивый синтаксис. Большое количество синтаксического сахара – конструкции, созданные для облегчения написания и понимания кода, которые не играют никакой роли при компиляции
Добавлено функциональное программирование (F#)
Огромное сообщество программистов по всему миру

К недостаткам можно отнести:

Преимущественно ориентирован на .NET (на Windows платформу)
Бесплатная версия распространяется только для небольших компаний и программистов-одиночек
Сохранился оператор go to

2.2 Реализация объектно-ориентированных технологий программирования в современных программно-математических средах

Объектно-ориентированное программирование. SIMULA I (1962-65) и Simula 67 (1967) — два первых объектно-ориентированных языка программирования. Simula 67 включала в себя большую часть концепций объектно-ориентированного программирования: классы и объекты, подклассы (наследование), виртуальные функции, безопасные ссылки и механизмы, позволяющие внести в программу коллекцию программных структур, описанных общим заголовком класса (префиксные блоки).

Алан Кэй из Xerox PARC использовал Simula как платформу для его разработки Smalltalk (первых версий языка в 1970-х), расширяя объектно-ориентированное программирование с помощью интеграции пользовательского интерфейса и интерактивного исполнения. Бьерн Страуструпп начал разработку C++ (в 1980-х) привнеся основные концепции Simula в C.

«Чистым» называют объектно-ориентированный язык, все типы которого являются или могут быть прозрачно представленными классами. Первым чистым. Первым «чистым» объектно-ориентированным языком был именно Smalltalk. Поэтому Java стала «чистым» объектно-ориентированным языком только с 5 версии, когда появилась возможность преобразования класса-обёртки в соответствующий ему примитивный тип.

Существует 4 принципа, которые делают язык объектно-ориентированным:

Инкапсуляция – это механизм сокрытия реализации данных путем ограничения доступа к публичным методам. Это механизм, который связывает воедино код и данные, которыми он манипулирует.
Полиморфизм – способность объектно-ориентированных языков эффективно различать сущности с одинаковыми именами.
Наследование – механизм, с помощью которого один класс может наследовать свойства (поля и методы) другого класса. Существует суперкласс и подкласс. Суперкласс – класс, свойства которого наследуются, также называют базовым классом или родительским классом. Подкласс – класс, который наследует другой класс, его также называют дочерний класс или производный класс. Подкласс может добавлять свои собственные поля и методы, в дополнение к полям и методом суперкласса.
Абстракция – упрощенная модель реальной жизни. То есть выделение значимой информации и исключение незначимой. Абстракция лишь частично описывает реальную сущность объекта.

2.3 Основные понятия ООП

Основные понятия ООП:

Класс — это способ описания сущности, определяющий состояние и поведение, зависящее от этого состояния, а также правила для взаимодействия с данной сущностью.
Объект (экземпляр) – это отдельный представитель класса, имеющий конкретное состояние и поведение, полностью определяемое классом.
Интерфейс – это набор методов класса, доступных для использования другими классами.
Метод – функция, работающая с объектом. свойство - переменная, определяемые в

классе.
Спецификаторы доступа: public, private, protected - определение доступа к методам или свойствам класса.
extends - ключевое слово, определяющее класс родителя.
Абстрактный класс - класс, от которого нельзя создать объект, предназначен только для дальнейшего наследования.
Финальный класс - класс, от которого нельзя наследоваться, предназначен только для создания объектов, но не для дальнейшего наследования.
Конструктор - метод, вызываемый в момент создания объекта;
Деструктор – метод, вызываемый в момент уничтожения объекта;
Имплемент — это класс реализующий методы интерфейса.

Принципы объектно-ориентированного программирования. Для того, чтобы помочь всем программистам разрабатывать качественные объектно-ориентированные приложения были разработаны принципы объектно-ориентированного программирования и проектирования.

Существует пять основных принципов объектно-ориентированного программирования и проектирования:

Принцип единственной ответственности (The Single Responsibility Principle). Каждый класс выполняет лишь одну задачу. Нарушение этого принципа приводит к тому, что класс отвечает за решение нескольких задач, его подсистемы, реализующие решение этих задач, оказываются связанными друг с другом. Изменения в одной такой подсистеме ведут к изменениям в другой.
Принцип открытости/закрытости (The Open Closed Principle). Программные сущности должны быть открыты для расширения, но закрыты для модификации.
Принцип подстановки Барбары Лисков (The Liskov Substitution Principle). Объекты в программе должны быть заменяемыми на экземпляры их подтипов без изменения правильности выполнения программы. Смысл этого принципа заключаются в том, чтобы классы-наследники могли бы использоваться вместо родительских классов, от которых они образованы, не нарушая работу программы. Если оказывается, что в коде проверяется тип класса, значит принцип подстановки нарушается. Наследующий класс должен дополнять, а не изменять базовый.
Принцип разделения интерфейса (The Interface Segregation Principle). Много интерфейсов, специально предназначенных для клиентов, лучше, чем один интерфейс общего назначения. Нужно создавать узкоспециализированные интерфейсы, предназначенные для конкретного клиента. Клиенты не должны зависеть от интерфейсов, которые они не используют.
Принцип инверсии зависимостей (The Dependency Inversion Principle). Объектом зависимости должна быть абстракция, а не что-то конкретное. Абстракции не должны зависеть от деталей. Наоборот, детали должны зависеть от абстракций. Модули верхних уровней не должны зависеть от модулей нижних уровней. Соответственно, модули и верхних и нижних уровней должны зависеть от абстракций.

Сокращенно эти принципы называют SOLID. Придерживание данным принципам позволяет создавать такую систему, которую будет легко поддерживать и расширять в течение долгого времени.

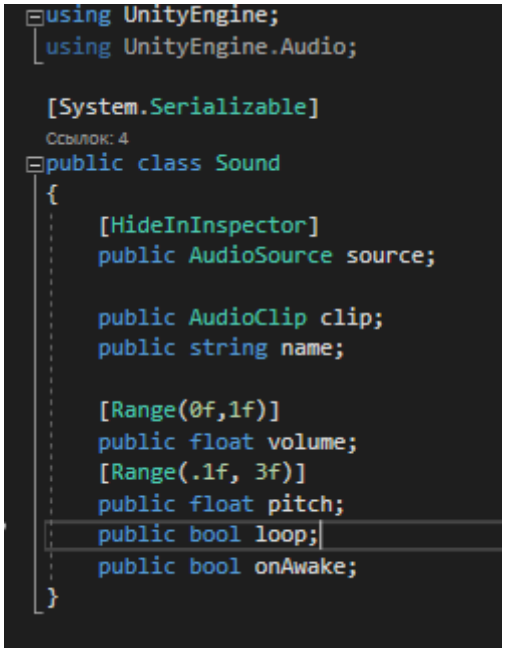

Рисунок 2.1 – Пример класса Sound из игры

3 ИНСТРУМЕНТАРИЙ

1.1Обоснование используемых инструментов

Для решения поставленных задач были использованы следующие инструменты:

IDE Microsoft Visual Studio 2019
Unity
GitHub Desktop

Примерно 80% от разработки программист тратит на понимание кода, который он написал, а также на перемещение по нему. Для того, чтобы облегчить данный процесс нужна IDE (интегрированная среда разработки). Любая IDE имеет в наличии синтаксический анализатор языка программирования, а это автодополнение, навигация, подсветка синтаксических, в некоторых случаях, семантических ошибок.

Главное преимущество IDE – позволить программисту сосредоточиться на решении алгоритмических задач, что значительно влияет на производительность труда разработчика.

IDE Microsoft Visual Studio 2019 – самая «правильная» среда разработки, так как и язык, и среда разработки были созданы в Microsoft. Среда разработки предоставляет так много инструментов работы с кодом, что в них легко потеряться. К плюсам этой среды можно отнести:

Встроенный Web-сервис, для обслуживания ASP.NET
Поддержка множества языков при разработке
Интуитивный стиль кодирования. Visual Studio форматирует код по мере его ввода, автоматически вставляя необходимые отступы
Возможность отладки
Интеграция с системой контроля версий Git

Unity — межплатформенная среда разработки компьютерных игр. Unity позволяет создавать приложения, работающие под более чем 20 различными операционными системами, включающими персональные компьютеры, игровые консоли, мобильные устройства, интернет-приложения и другие.

Основными преимуществами Unity являются наличие визуальной среды разработки, межплатформенной поддержки и модульной системы компонентов. К недостаткам относят появление сложностей при работе с многокомпонентными схемами и затруднения при подключении внешних библиотек.

GitHub Desktop – это приложение, которое реализует взаимодействие с интернет сервисом GitHub и системой контроля версий Git. GitHub позволяет сохранять и делится своим кодом с другими пользователями. GitHub Desktop это удобный способ использования Git поскольку он показывает разветвления вашего проекта, показывает репозитории и даёт возможность загружать всё это на GitHub.

1.2Использование системы контроля версий GIT

Системы контроля версий (СКВ, VCS, Version Control Systems) позволяют разработчикам сохранять все изменения, внесённые в код, в также дают возможность нескольким разработчиком работать над одним проектом и сохранять все свои изменения.

Git — распределённая система контроля версий, которая даёт возможность разработчикам отслеживать изменения в файлах и работать совместно с другими разработчиками. Git стоит отдельно от других СКВ из-за подхода к работе с данными. Большинство других систем хранят информацию в виде списка изменений в файлах. Вместо этого, подход Git к хранению данных больше похож на набор снимков миниатюрной файловой системы. Каждый раз, когда вы сохраняете состояние своего проекта в Git, система запоминает, как выглядит каждый файл в этот момент, и сохраняет ссылку на этот снимок

Рисунок 3.1 – Распределенная система контроля версий

Когда вы производите какие-либо действия в Git, практически все из них только добавляют новые данные в базу Git. Очень сложно заставить систему удалить данные либо сделать что-то, что нельзя впоследствии отменить. Как и в любой другой СКВ, вы можете потерять или испортить свои изменения, пока они не зафиксированы, но после того, как вы зафиксируете снимок в Git, будет очень сложно что-либо потерять, особенно, если вы регулярно синхронизируете свою базу с другим репозиторием.

Для того, чтобы работать с Git необходимо изучить базовые команды:

Git add – добавляет содержимое рабочей директории в индекс для последующего коммита
Git commit – берет все данные, добавленные в индекс, и сохраняет слепок во внутренней базе данных, далее сдвигает указатель текущей ветки на этот слепок
Git status – показывает состояние файлов в рабочей директории и индексе
Git rm – удаляет файлы из текущей директории или индекса
Git mv – перемещение файлов между директориями

4 ИГРОВОЙ ДВИЖОК UNITY

Интерфейс Unity каждый разработчик может подстроить под себя. Но существуют главные окна, благодаря которым работать над созданием игры намного удобнее.

На рисунке 4.1 представлен внешний вид Unity. Справа находится панель Inspector, которая позволяет посмотреть свойства выбранного объекта, изменить данные свойства, а также добавить другие компоненты.

Окно, в которой отображается наше игра называется Scene View, и она имеет 2 вкладки: scene и game. Scene показывает как выглядит наша сцена, на ней можно трансформировать и

перемещать различные объекты. Game показывает как в конечном итоге будет выглядеть наша игра.

Снизу находится Project Window, в которой расположены все наши объекты (спрайты, звуки, ассеты). Рядом с Inspector располагается Hierarchy Window, где указана иерархия всех объектов, расположенных на сцене. Любой объект в Unity имеют тип GameObject.

Сверху находится Toolbar, который позволяет нам запускать игру, ставить на паузу, и переключиться на следующий кадр.

Для того, чтобы протестировать мобильную игру, без сборки проекта, Unity разработала Unity Remote. Установив на мобильное устройство Unity Remote, и нажав кнопку play, вы можете поиграть в свою игру.

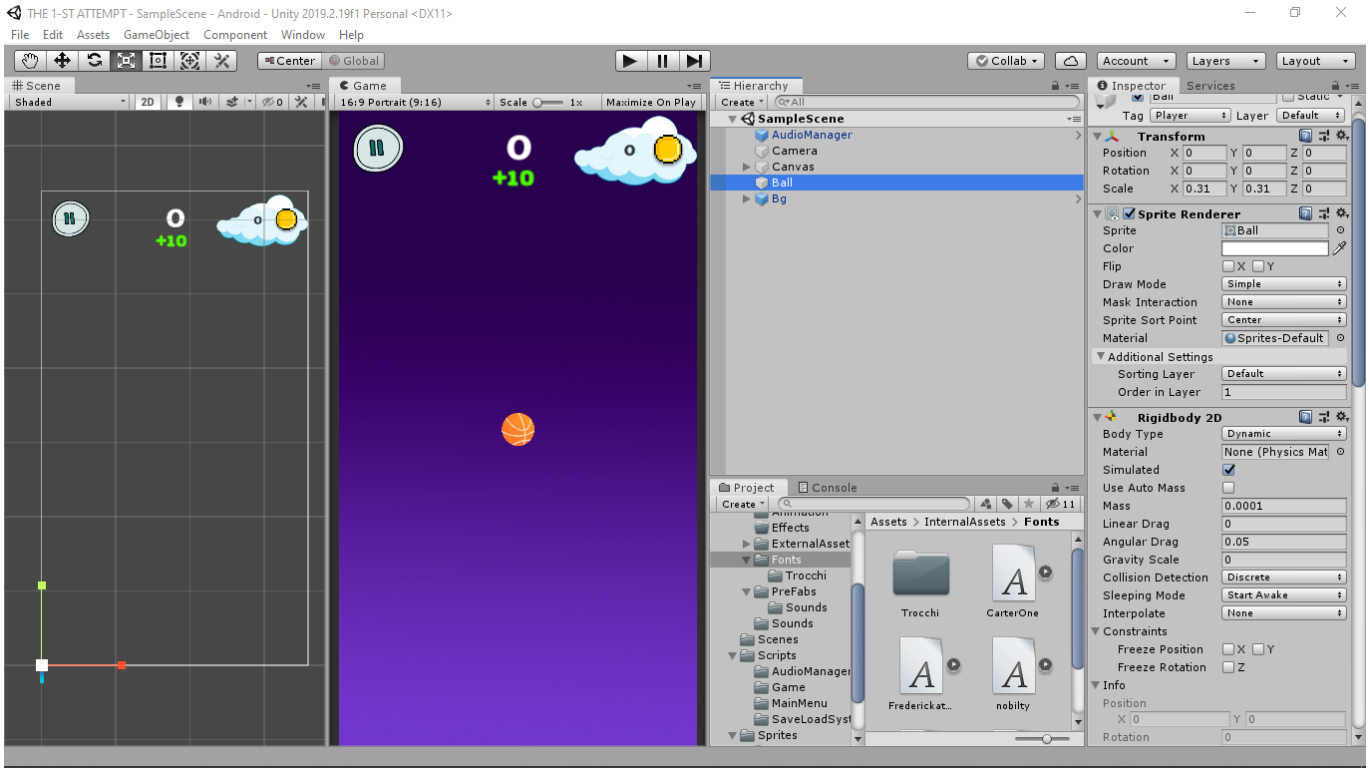

Рисунок 4.1 – Пользовательский интерфейс Unity

Unity позволяет работать с огромным количеством компонентов внутри своей среды. Например, имеется возможность редактирования изображений, создание собственных материалов, свою систему частиц, UI, звуков, шрифтов и много всего другого. В данной среде вы можете создавать анимации. Если вам чего-то не хватает, то можно зайти во вкладку Assets Store.

Asset Store -магазин Unity, где вы можете найти различные плагины или ассеты, в которых уже реализованы некоторые возможности.

Для создания поведения игрового объекта в Unity необходимо создать C# скрипт, который, по умолчанию, наследуется от базового класса MonoBehaviour. Данный класс содержит большое количество методов, помогающим вам работать с Unity. По умолчанию у вас создается 2 метода: Start() и Update().

Метод Start() запускается при появлении объекта на сцене. А метод Update() обрабатывается каждый кадр. В Unity существует такое понятие как Coroutine – корутины.

Корутины – сопрограммы в Unity, простой способ запускать функции, которые должны работать в том же потоке.

С точки зрения языка программирования C#, корутины – это простые итераторы, которые используют ключевое слово yield и возвращают IEnumerator.

Метод итератора использует оператор yield return для поочередного возврата каждого элемента.

В Unity корутины регистрируются и выполняются до первого yield. Для того, чтобы запустить корутин используют метод StartCoroutine(). После вызова данного метода Unity опрашивает зарегистрированные корутины после каждого вызова Update().

Существует несколько вариантов для возвращаемых в yield значений:

Продолжить после следующего FixedUpdate:

yield return new WaitForFixedUpdate()

Продолжить после следующего LateUpdate и рендеринга сцены:

yield return new WaitForEndOfFrame();

Продолжить через некоторое время (100ms):

yield return new WaitForSeconds(0.1f);

Продолжить по завершению другого короутина:

yield return StartCoroutine(AnotherCoroutine());

Продолжить после загрузки удаленного ресурса:

yield return new WWW(SomeLink);

Продолжить после прохода итерации цикла Update():

yield return null;

Это удобно использовать для того, чтобы вызвать какой-то объект через определенное время, после некоторого события.

```
private IEnumerator Wait()
{
    while (true)
    {
        yield return new WaitForSeconds(2.7f);
    }
}
```

Рисунок 4.2 – Реализация корутина Wait

Благодаря встроенному окну Inspector в Unity очень удобно создавать различные системы. Например, система управления аудио. Для этого создадим класс Sound, который будет описывать звуковой файл. Затем создадим AudioManager, в котором можно будет настраивать каждый звуковой файл. Затем созданный скрипт AudioManager добавляем к любому пустому

объекту на сцене. В окне Insptector появляется окошко с нашими звуками. В противном случае нам пришлось бы настраивать каждый звук отдельно.

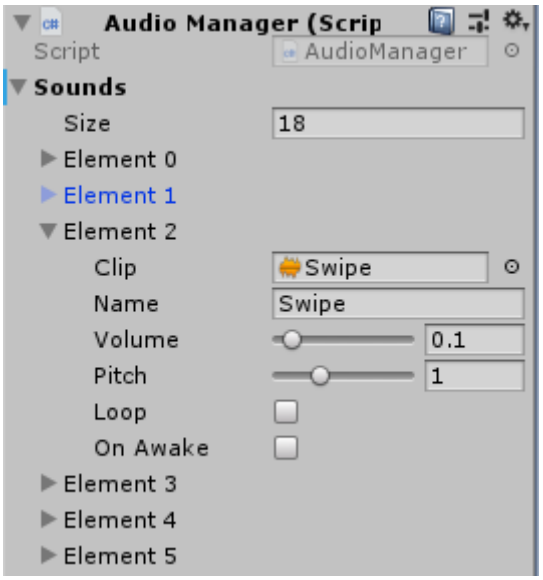

Рисунок 4.3 – AudioManager – система управлением звука

5 Разработка системы управления звукового сопровождения

Шаблон проектирования или паттерн в разработке программного обеспечения — повторяемая архитектурная конструкция, представляющая собой решение проблемы проектирования в рамках некоторого часто возникающего контекста. Шаблоны нельзя преобразовать в код, это лишь пример решения поставленной задачи, который можно использовать в различных ситуациях.

Существует три вида шаблонов:

Поведенческие
Порождающие
Структурные

Поведенческие шаблоны – это паттерны, которые решают задачи эффективного и безопасного взаимодействия между объектами программы. Поведенческие паттерны:

цепочка обязанностей – который позволяет передавать запросы последовательно по цепочке обработчиков. Каждый последующий обработчик решает, может ли он обработать запрос сам и стоит ли передавать запрос дальше по цепи
команда – превращает запросы в объекты, позволяя передавать их как аргументы при вызове методов, ставить запросы в очередь, логировать их, а также поддерживать отмену операций
итератор – даёт возможность последовательно обходить элементы составных объектов, не раскрывая их внутреннего представления
посредник – позволяет уменьшить связанность множества классов между собой, благодаря перемещению этих связей в один класс–посредник
снимок – позволяет сохранять и восстанавливать прошлые состояния объектов, не раскрывая подробностей их реализации

наблюдатель – создаёт механизм подписки, позволяющий одним объектам следить и реагировать на события, происходящие в других объектах
состояние – позволяет объектам менять поведение в зависимости от своего состояния
стратегия – определяет семейство схожих алгоритмов и помещает каждый из них в собственный класс, после чего алгоритмы можно взаимозаменять прямо во время исполнения программы
шаблонный метод – определяет скелет алгоритма, перекладывая ответственность за некоторые его шаги на подклассы
посетитель – позволяет добавлять в программу новые операции, не изменяя классы объектов, над которыми эти операции могут выполняться

Порождающие шаблоны – шаблоны проектирования, которые отвечают за удобное и безопасное создание новых объектов или даже целых семейств объектов. Существуют следующие порождающие шаблоны:

фабричный метод – определяет общий интерфейс для создания объектов в суперклассе, позволяя подклассам изменять тип создаваемых объектов
абстрактная фабрика – позволяет создавать семейства связанных объектов, не привязываясь к конкретным классам создаваемых объектов
строитель – позволяет создавать сложные объекты пошагово
прототип – позволяет копировать объекты, не вдаваясь в подробности их реализации
одиночка – гарантирует, что у класса есть только один экземпляр, и предоставляет к нему глобальную точку доступа

Структурные шаблоны – это паттерны, которые отвечают за построение удобных в поддержке иерархий классов. Структурные шаблоны:

адаптер – позволяет объектам с несовместимыми интерфейсами работать вместе
мост – разделяет один или несколько классов на две отдельные иерархии – абстракцию и реализацию, позволяет изменять их независимо друг от друга
компоновщик – позволяет сгруппировать множество объектов в древовидную структуру, а затем работать с ней так, будто это единичный объект
декоратор – позволяет динамически добавлять объектам новую функциональность, оборачивая их в полезные ”обёртки”
фасад – предоставляет простой интерфейс к сложной системе классов, библиотеке или фреймворку
легковес – позволяет вместить большее количество объектов в отведённую оперативную память
заместитель – позволяет подставлять вместо реальных объектов специальные объекты–заменители

Использование шаблонов помогает избежать многих ошибок и позволяет сократить время, используя проверенные решения. Также при работе в команде с другими разработчиками не обязательно объяснять, что вы сделали, а можно сказать название шаблона.

Каждый из шаблонов решает конкретную проблему, поэтому, прежде чем использовать какой-либо шаблон, нужно убедиться, что он подходит под решение вашей задачи.

Практической задачей является разработка системы управления звуковым

сопровождением. В начале мы создаем класс Sound, который описывает наш звуковой файл. Каждый звуковой файл в Unity, который должен быть представлен в игровой среде, описывается с помощью класса AudioSource. Класс AudioClip является контейнером, который хранит в себе информацию об аудиофайле.

Наш класс Sound содержит поле name, которое позволяет присвоить имя нашему аудиофайлу, чтобы в дальнейшем искать наш звуковой файл по имени.

Любой звуковой файл имеет несколько характеристик. В нашем случае мы будем использовать следующие характеристики: volume – громкость (значения варьируются от 0 до 1), pitch – питч, loop – зацикливание.

Перейдем к созданию класса AudioManager. Данный класс – система управления звуковыми файлами в Unity. В нем мы создаем массив, который состоит из наших звуковых файлов. Также в начале, а значит прописываем в методе Awake(), делаем проверку на содержание одного AudioManager на сцене, также вызываем метод DontDestroyOnLoad(), который не удаляет наш объект, при загрузке новой сцены.

В методе Awake() заполняем наш массив, а также звуковые характеристики. Класс AudioManager содержит 2 метода: Play() и Stop(). Каждый из этих методов в качестве входных параметров принимает строку, а именно название звукового файла. Затем мы смотрим существует ли звуковой файл с таким названием. После этого метод Play() воспроизводит аудиофайл, а метод Stop() – останавливает. Таким образом в окне Inspector мы видим полноценную систему управления звуковыми файлами. Каждому звуковому файл мы присваиваем свое имя и необходимые нам характеристики, которые, в последствии, можем изменять.

Заключение

В процессе выполнения курсового проекта были получены навыки работы с игровым движком Unity. Познакомился и изучил язык программирования C#. В ходе проектирования курсового проекта были получены знания из области игровой индустрии, в частности удалось примерить на себя роль геймдизайнера. В результате была разработана мобильная игра категории гипер-казуальные игры – “Falling ball”. В ходе разработки мобильной игры были получены практические знания об использовании методик объектно-ориентированного программирования. Убедившись в корректности работы мобильной игры, можно сказать, что поставленная цель была достигнута.

Список использованных источников

1. [url] **Краткий обзор языка C#** https://docs.microsoft.com/ru-ru/dotnet/csharp/tour-of-csharp/
2. [url] **Кэжуал и гипер-кэжуал: сравнение показателей двух мобильных жанров от GameAnalytics**
https://app2top.ru/analytics/ke-zhual-i-giper-ke-zhual-sravnenie-pokazatelej-dvuh-mobil-ny-h-zhanrov-ot-gameanalytics-144130.html
3. [url] **Scripting** https://docs.unity3d.com/Manual/ScriptingSection.html
4. [url] **Паттерны проектирования** https://refactoring.guru/ru
5. [url] **Редактор или IDE? Очередная попытка анализа** https://habr.com/ru/post/265197/
6. [url] **Серьезные забавы: почему видеоигры становятся популярнее кино**

https://www.forbes.ru/tehnologii/357631-sereznye-zabavy-pochemu-videoigry-stanovyatsya-populyarnee-kino

Приложения

1. [электронный документ] 5ec7dd7a1e83d_Записка.docx
2. [электронный документ] 5ec7de1cc2701_Бланк задания_Петрикевич.docx