МИНИСТЕРТСВО ОБРАЗОВАНИЯ И НАУКИ КЫРГЫЗСКОЙ РЕСПУБЛИКИ
ОШСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
Институт математики, физики, техники и информационных технологий
Кафедра прикладной математики, информатики и графического дизайна

АЪЗИМОВА АКЫЛАЙ ОДИЛБЕКОВНА

КВАЛИФИКАЦИОННАЯ РАБОТА

Направление 570400, Дизайн: “Графический дизайн”
тема: "Разработка мобильной аркадной игры”

Студентка группы ГД(Б)-1-20	А.О. Аъзимова
Научный руководитель	Рецензент
Ф.-к.т.н., доцент	КМИГД каф. преподаватель
_____ Шестаков Е.И.	_____ Бекмурза уулу Ы.

Защита предоставлена
«___» _____ 2024 г.
Заведующий каф. КМИГД, доцент Т.М. Жолдошов

Ош 2024

Оглавление

Введение

С развитием компьютерных технологий игры становятся неотъемлемой частью современной культуры, включая образовательный, развивающий, развлекательный опыт. Однако при этом, разработка игр является сложным и многогранным процессом, требующим интеграции знаний из различных областей, таких как программирование, дизайн и даже психология.

Целью данной дипломной работы является разработка трехмерной игры, в которой игрок управляет машиной, собирая звездочки в ограниченное время. Основными задачами исследования являются:

1. Изучение процесса разработки игрового приложения с использованием Unity и языка программирования C#.
2. Анализ эффективности игровой механики, включая систему времени и сбора звездочек.
3. Тестирование разработанного приложения.

Методы исследования включают в себя анализ существующих игровых механик, разработку прототипа игры, а также проведение тестирования среди целевой аудитории с последующим анализом результатов.

Данная работа актуальна в контексте постоянного развития игровой индустрии и повышенного интереса к созданию качественных игровых продуктов, способных увлечь и заинтересовать широкую аудиторию.

Глава 1. Теоретические аспекты разработки игр

1.1 Актуальность разработки компьютерных и мобильных игр

Разработка компьютерных и мобильных игр является актуальной задачей по целому ряду причин:

1. **Растущий рынок:** Индустрия видеоигр постоянно растет, привлекая все большее количество игроков и инвестиций. Это создает возможности для новых и существующих разработчиков игр.
2. **Технологический прогресс:** Развитие аппаратных и программных технологий, таких как улучшенные графические движки, искусственный интеллект, виртуальная и дополненная реальность, позволяет создавать более качественные и увлекательные игровые продукты.
3. **Разнообразие платформ:** Игры доступны на различных платформах, таких как ПК, мобильные устройства, даже виртуальная реальность. Это открывает широкие возможности для целевой аудитории.
4. **Мобильные игры:** Рынок мобильных игр продолжает расти, привлекая не только казуальных игроков, но и серьезных геймеров. Мобильные устройства становится мощнее, что позволяет создавать более сложные игры.
5. **Игровые движки и инструменты:** Существует множество мощных игровых движков и инструментов разработки, таких как Unity, Unreal Engine, Godot, Construct, которые делают процесс создания игр более доступным и эффективным.
6. **Модели монетизации:** С появлением различных моделей монетизации, таких как покупка приложений, внутриигровые покупки, реклама, подписки, игровые разработчики имеют возможность зарабатывать на своих продуктах.

7. **Игровое сообщество:** Стремительное развитие игровой культуры и сообщества позволяет играм стать не только развлечением, но и платформой для общения, соревнований и творчества.

Таким образом, разработка компьютерных и мобильных игр остается актуальной, привлекающей как крупные компании, так и индивидуальных разработчиков.

1.1.1 Понятие о компьютерных и мобильных игр

Компьютерные игры - это программы, созданные для запуска и игры на персональных компьютерах (ПК) или ноутбуках. Компьютерные игры имеют огромное влияние на культуру, образование и развлечения, становясь неотъемлемой частью современного информационного общества. Они объединяют миллионы игроков по всему миру и продолжают развиваться с развитием технологий и потребностей аудитории.

Одна из главных характеристик компьютерных игр — это возможность взаимодействия игрока с виртуальным миром, персонажами, объектами и событиями в игре. А также игроки могут принимать решения, влияющие на развитие сюжета или на исход игровых ситуаций. Развитие компьютерных технологий, графики, звука, искусственного интеллекта и физики позволяет создавать более реалистичные и захватывающие игровые миры. Мощные игровые движки (например, Unity, Unreal Engine) облегчают процесс создания игр.

Многие компьютерные игры выпускаются для различных платформ, таких как ПК (Windows, macOS, Linux), игровые консоли (PlayStation, Xbox, Nintendo) и мобильные устройства (iOS, Android). Это позволяет игрокам выбирать удобную платформу для игры. Многие игры поддерживают режимы онлайн-игры и мультиплеера, что позволяет игрокам играть с друзьями или другими игроками по всему миру, обмениваться опытом и соревноваться.

1.1.2 Виды компьютерных и мобильных игр

Компьютерные и мобильные игры охватывают широкий спектр жанров и типов игр, каждый из которых имеет свои особенности и привлекает определенную аудиторию. Рассмотрим основные виды компьютерных и мобильных игр:

Компьютерные игры

Экшен (Action): Игры этого жанра обычно ориентированы на быструю и динамичную игру с акцентом на боевые действия. Примеры: "Call of Duty", "Devil May Cry" (Рис.1)

Рис. 1.1 - Игра "Call of Duty"

Приключение (Adventure): Эти игры часто содержат интересный сюжет, головоломки, исследование мира и взаимодействие с персонажами. Примеры: "The Legend of Zelda", "Life is Strange". (Рис.2.)

Рис. 1.2 - Игра "Life is Strange"

Шутеры (Shooter): Основной акцент делается на стрельбе и боях с противниками. Могут быть как от первого лица (FPS), так и от третьего лица (TPS). Примеры: "Counter-Strike", "Destiny".

Рис. 1.3 – Игра "Counter-Strike"

Стратегии (Strategy): Игры, где игроку необходимо разрабатывать стратегию, управлять ресурсами и войсками для достижения целей. Примеры: "Civilization", "StarCraft".

Рис. 1.4 – Игра "StarCraft"

Ролевые игры (RPG): Основной упор делается на развитие персонажа, выполнение квестов, исследование мира и взаимодействие с другими персонажами. Примеры: "The Witcher", "Final Fantasy".

Рис. 1.5 – Игра "Final Fantasy"

Симуляторы (Simulator): Игры, имитирующие различные аспекты реальной жизни, такие как авиасимуляторы, гоночные симуляторы, симуляторы жизни и т.д. Примеры: "Flight Simulator", "The Sims".

Рис. 1.6 – Игра "The Sims"

Мобильные игры:

Аркады (Arcade): Простые и быстрые игры, часто с одним основным геймплеем, например, прыгание или стрельба. Примеры: "Flappy Bird", "Angry Birds".

Рис. 1.7 – Игра "Angry Birds"

Головоломки (Puzzle): Игры, требующие логического мышления и решения различных головоломок или задач. Примеры: "Candy Crush", "Monument Valley", "Skillz".

Рис. 1.8 – Игра "Monument Valley"

Казуальные игры (Casual): Легкие и доступные игры, часто без строгого сюжета или требований к навыкам. Примеры: "Temple Run", "Crossy Road".

Рис. 1.9 – Игра "Temple Run"

Спортивные игры (Sports): Игры, имитирующие различные виды спорта или включающие в себя элементы соревнований. Примеры: "FIFA Mobile", "NBA 2K Mobile".

Рис. 1.10 – Игра "NBA 2K Mobile"

Карточные и настольные игры (Card & Board): Игры, основанные на картах или настольных играх, такие как шахматы, покер, дурак и др. Примеры: "Hearthstone", "Chess".

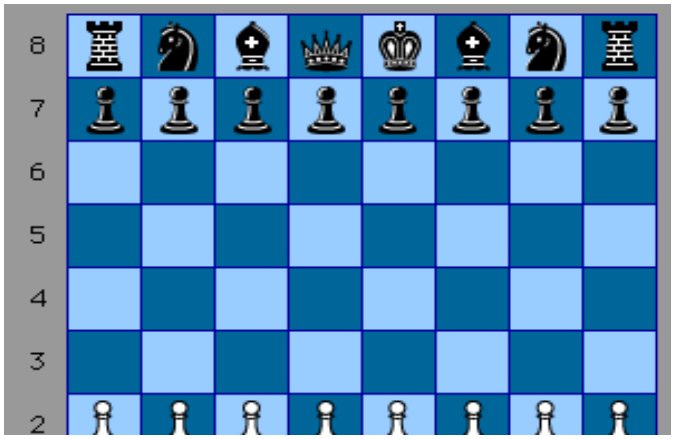


Рис. 1.11 – Игра "Chess"

Игры в жанре (Endless Runner): Игры, в которых персонаж автоматически бежит вперед, а игрок управляет его движением, избегая препятствий. Примеры: "Subway Surfers", "Temple Run".

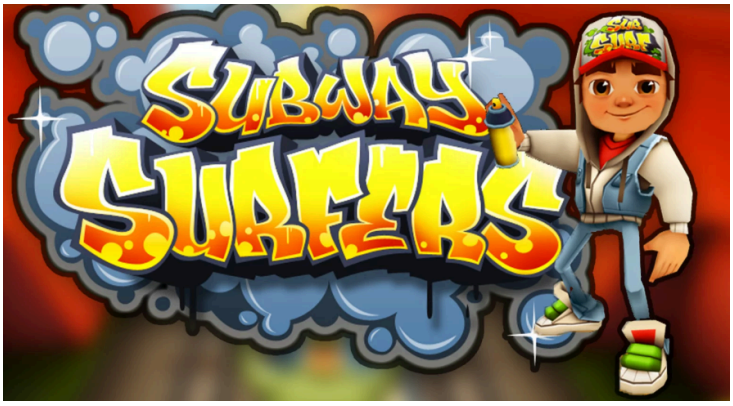


Рис. 1.12 – Игра "Subway Surfers"

1.1.3 Применение компьютерных и мобильных игр

Применение компьютерных и мобильных игр в современном мире очень разнообразно и простирается от развлечений до образования и развития. Вот несколько областей, в которых игры активно используются:

1. **Развлечение:** Это, основное применение игр. Они предоставляют возможность людям отдохнуть, развлечься и погрузиться в виртуальные миры.

2. **Образование:** Многие образовательные игры предназначены для обучения различным навыкам, начиная от математики и грамотности до истории и наук.
3. **Социализация:** Многие игры позволяют игрокам взаимодействовать друг с другом онлайн, что способствует социализации и коммуникации.
4. **Тренировка мозга:** Некоторые игры разработаны для тренировки ума, улучшения памяти, реакции, логического мышления и других когнитивных навыков.
5. **Симуляция:** В различных областях, таких как авиация, медицина и инженерия, игры используется для тренировки и симуляции определенных сценариев.
6. **Реклама и маркетинг:** Многие компании используют игры для продвижения своих товаров и услуг, создавая брендированные игры или размещая рекламу в существующих играх.
7. **Терапия:** В некоторых случаях игры используются игры в качестве терапевтического инструмента, например, для реабилитации после травмы или для лечения психологических расстройств.
8. **Исследование:** Некоторые игры используются для сбора данных и проведения исследований в различных областях, таких как психология и наука о климате.

Это лишь несколько примеров того, как широко могут применяться компьютерные и мобильные игры. Важно понимать, что их влияние и потенциал становятся все более значимыми в современном мире.

1.2 Аналитический обзор программных средств для создания игр

Аналитический обзор программных средств для создания игр позволяет оценить инструменты, которые могут быть полезны при разработке игр. Вот несколько из них:

1. Unity - одно из самых популярных и мощных программных средств для создания игр. Оно поддерживает разработку игр для различных платформ, включая компьютеры, мобильные устройства, консоли и виртуальную реальность. Unity имеет интуитивно понятный интерфейс и обширную документацию, что делает его привлекательным выбором для как начинающих, так и опытных разработчиков
2. Unreal Engine - еще один мощный движок для создания игр, который широко используется в индустрии. Он известен своими высококачественными графическими возможностями и продвинутыми инструментами для разработки. Unreal Engine также поддерживает множество платформ и имеет активное сообщество разработчиков.
3. Godot - это бесплатный и открытый исходный код движок для создания 2D и 3D игр. Он обладает простым в использовании интерфейсом и хорошей поддержкой скриптования на нескольких языках, включая GDScript (язык программирования, разработанный специально для Godot), C#, и другие.
4. GameMaker Studio - это инструмент, который прежде всего ориентирован на создание 2D игр. Он предлагает простой в использовании интерфейс и интуитивно понятные инструменты для создания игр без необходимости в глубоком знании программирования.
5. Construct - еще одно программное средство для создания 2D игр, которое обладает простым в использовании интерфейсом и фокусируется на создании игр без необходимости в программировании. Он предоставляет

множество готовых к использованию элементов и возможностей для быстрой разработки игр.

Это только небольшой обзор различных программных средств для создания игр. Выбор конкретного инструмента зависит от потребностей проекта, уровня опыта разработчика и других факторов.

1.2.1 UnrealEngine

Unreal Engine (UE) - это один из наиболее мощных и широко используемых программных средств для создания игр и визуализации в реальном времени. Вот некоторые ключевые особенности и характеристики Unreal Engine:

1. **Графика высокого качества:** Unreal Engine славится своими впечатляющими графическими возможностями. Он использует передовые технологии рендеринга, такие как реалистичное освещение, теней и отражений, что делает игры и визуализации виртуальной реальности великолепными визуальными произведениями.
2. **Многоплатформенность:** Unreal Engine поддерживает разработку игр для широкого спектра платформ, включая ПК, консоли, мобильные устройства и виртуальную реальность. Это делает его идеальным выбором для создания игр с учетом потенциально различных целевых платформ.
3. **Блюпринты и программирование:** Unreal Engine предлагает два основных способа создания игровой логики: с использованием визуальной системы графов под названием "блюпринты" и с использованием языка программирования C++. Это обеспечивает гибкость и выбор разработчикам в зависимости от их предпочтений и навыков.
4. **Сообщество и ресурсы:** Unreal Engine имеет активное и обширное сообщество разработчиков, которые делятся своими знаниями, опытом и ресурсами. Существует множество онлайн-уроков, документации и форумов,

которые помогут новичкам начать работу с движком и узнать его возможности в подробностях.

5. **Бесплатность и открытый исходный код:** Unreal Engine доступен для использования бесплатно с опцией выплаты роялти при коммерческой выпускаемой игре. Кроме того, доступен полный доступ к исходному коду, что позволяет разработчикам настраивать и расширять функциональность по своему усмотрению.
6. **Виртуальная реальность и дополненная реальность:** Unreal Engine имеет интегрированную поддержку для разработки приложений виртуальной реальности (VR) и дополненной реальности (AR), что делает его популярным выбором для создания иммерсивных визуальных и виртуальных сред.

Unreal Engine остается одним из наиболее влиятельных и инновационных инструментов для создания игр и визуализации в реальном времени. Его мощные возможности и обширный функционал делают его привлекательным выбором для профессиональных разработчиков и студий, а также для начинающих, желающих освоить и войти в индустрию игровой разработки.

1.2.2 Godot

Godot - это бесплатный и открытый исходный код игровой движок, который позволяет создавать 2D и 3D игры. Он предоставляет разработчикам инструменты для проектирования, создания и запуска игр на различных платформах, таких как ПК, мобильные устройства и консоли. Godot обладает простым в использовании интерфейсом и поддерживает скриптование на нескольких языках программирования, что делает его доступным для начинающих разработчиков без глубоких знаний в области программирования.

1.2.3 GameMaker

GameMaker - это интегрированная среда разработки (IDE), которая позволяет создавать 2D игры без глубокого погружения в программирование. Она предлагает простой в использовании интерфейс и набор инструментов для создания игровой логики, графики и звуков. GameMaker обладает графическим редактором для создания спрайтов и объектов, а также системой событий и действий для программирования поведения объектов в игре. Этот инструмент позволяет как начинающим, так и опытным разработчикам создавать качественные игры для различных платформ, включая компьютеры, мобильные устройства и консоли.

1.2.4 Unity 3D

Unity 3D - это мощное программное средство для создания игр и интерактивных приложений. В основном оно используется для разработки трехмерных (3D) проектов, но также поддерживает создание двухмерных (2D) игр. Unity предоставляет широкий спектр инструментов для разработки игровой логики, создания графики, анимаций, звуков и управления физикой. Одной из ключевых особенностей Unity является его многоплатформенность: игры, созданные в Unity, могут быть развернуты на различных платформах, включая ПК, мобильные устройства, консоли и виртуальную реальность. Unity также обладает мощной средой разработки (IDE) и поддерживает различные языки программирования, такие как C#, что делает его доступным для широкого круга разработчиков, как для начинающих, так и для опытных специалистов.

1.3 Теоретические аспекты программирования игр

Теоретические аспекты программирования игр включают в себя ряд основных концепций и принципов, которые являются основой для создания игровых приложений. Вот некоторые из них:

1. **Игровой цикл (Game Loop):** Это основной цикл выполнения программы игры, который обрабатывает ввод от пользователя, обновляет состояние игры и рендерит графику. Правильное управление игровым циклом обеспечивает плавное и реактивное поведение игры.
2. **Управление ресурсами:** Эффективное управление ресурсами, такими как текстуры, звуки и модели, является важным аспектом программирования игр. Это включает в себя загрузку, выгрузку и кэширование ресурсов для оптимизации производительности и памяти.
3. **Обработка ввода (Input Handling):** Игровое приложение должно эффективно обрабатывать ввод от пользователя, такой как нажатия клавиш, движения мыши или касания на сенсорном устройстве, чтобы реагировать на действия игрока.
4. **Управление состоянием (State Managemenet):** Игровое приложение часто имеет различные состояния, такие как главное меню, игровой уровень, пауза и т. д. Эффективное управление этими состояниями позволяет плавно переходить между ними и поддерживать последовательность игрового процесса.
5. **Физика:** Многие игры имеют элементы физики, такие как движение объектов, коллизии и силы. Эффективная реализация физических законов и эффектов может значительно улучшить реалистичность игрового мира.
6. **Искусственный интеллект (AI):** В некоторых играх требуется программирование искусственного интеллекта для управления поведением непротивников, союзников или других объектов в игровом мире.
7. **Алгоритмы рендеринга (Rendering Algoritmhs):** Рендеринг игровых сцен требует использования различных алгоритмов, таких как растеризация, освещение и текстурирование, для создания реалистичных графических изображений.

8. **Оптимизация производительности:** Эффективное использование ресурсов компьютера, таких как процессор, графический процессор и память, является важным аспектом программирования игр. Это включает в себя оптимизацию алгоритмов, управление памятью и параллельную обработку.

Этапы создания игры

Этапы создания игры обычно включают в себя следующие шаги:

1. **Планирование и концепция:**
    - Определение жанра игры и целевой аудитории.
    - Создание концепции игры, включая сюжет, механики игры и основные функции.
    - Составление дизайна игры, включая характеристики персонажей, уровни и миры.
2. **Проектирование:**
    - Создание детального проекта игры, включая дизайн уровней, интерфейса пользователя и игровой логики.
    - Разработка документации, включающей в себя графические концепции, сценарии и спецификации функций.
3. **Разработка:**
    - Создание игровых артов, включая спрайты, модели, текстуры и анимации.
    - Программирование игровой логики, включая обработку ввода, управление ресурсами, физику и искусственный интеллект.
    - Интеграция аудио, включая звуковые эффекты и музыку.
    - Тестирование и отладка игры для выявления и исправления ошибок.
4. **Тестирование и отладка:**

- Проведение различных тестов, включая функциональное тестирование, тестирование производительности и тестирование совместимости.
- Исправление ошибок и улучшение игрового процесса на основе об5ратной связи тестировщиков.

5. **Выпуск и публикация:**
   - Подготовка игры к выпуску, включая упаковку, установщики и документацию.
   - Развертывание игры на выбранных платформах, таких как ПК, мобильные устройства, консоли и онлайн-платформы.
   - Маркетинг и продвижение игры, включая рекламу, публикации в социальных сетях и контент-маркетинг.
6. **Поддержка и обновления:**
   - Поддержка игры после выпуска, включая решение проблем, обновления и добавление нового контента,
   - Взаимодейтсвие с сообществом игроков, прослушивание обратной связи и внесение изменений в игру на основе этой обратной связи.

Каждый из этих этапов играет важную роль в создании игры и требует внимания к деталям и координации усилий между различными членами команды разработки игр.

1.3.1 Математические и программные концепции используемые при разработке игр в Unity3D (вектора, кватернионы, физика твердого тела, GameObject, Transform)

Концепция векторов играет ключевую роль в различных областях математики, физики и программирования. Вот основные концепции векторов:

- **Направление (Direction):**

Вектор представляет собой величину, которая имеет как величину (длину), так и направление. Направление вектора определяется его ориентацией в пространстве. Например, вектор может указывать на север, юг, восток или запад в двумерном пространстве, или же он может указывать на конкретное направление в трехмерном пространстве, такое как вперед, вниз или влево.

- **Длина (Magnitude):**

Длина вектора, также известная как модуль или величина, определяет его размер или протяженность в пространстве. Длина вектора вычисляется как расстояние от его начальной точки (обычно начала координат) до его конечной точки. Длина вектора всегда является неотрицательным числом.

Векторы широко используются в различных аспектах программирования и гейм дизайна. Например, векторы используются для представления позиций объектов в игровом мире, указания направления движения, моделирования сил и скоростей, а также для обнаружения и обработки коллизий между объектами.

В игровой разработке в Unity3D, векторы часто используются для управления перемещением объектов, вычисления силы гравитации, определения направления света и многих других задач.

Примеров того, как векторы используются для представления различных физических величин в игровом мире:

Позиции объектов: Векторы используются для представления позиций объектов в игровом мире. Каждый объект может быть размещен в трехмерном пространстве с помощью трех координат (x, y, z). Например, позиция игрового персонажа может быть представлена вектором (x, y, z), где x, y и z - это координаты в трехмерном пространстве.

Направления движения: Векторы также используются для определения направления движения объектов. Например, если игровой персонаж движется

вперед, его направление движения может быть представлено вектором, который указывает вперед от текущей позиции персонажа.

Скорости: Скорость объекта также может быть представлена вектором. Вектор скорости показывает не только величину скорости (модуль вектора), но и ее направление. Например, если скорость объекта равна 5 м/с в направлении вправо, это может быть представлено вектором (5, 0, 0).

Силы и ускорения: Векторы также используются для представления сил и ускорений, воздействующих на объекты. Например, гравитационная сила, действующая на объект, может быть представлена вектором, указывающим вниз от объекта с определенной силой.

Нормали и ориентации: Векторы могут использоваться для определения ориентации объектов или поверхностей в игровом мире. Например, вектор нормали к поверхности используется для определения угла падения света на объект и, следовательно, его отображения.

Это лишь несколько примеров того, как векторы широко применяются в игровой разработке для моделирования различных физических величин и поведения объектов в игровом мире.

Примеры использования векторов для различных аспектов игровой разработки:

Перемещение объектов: При перемещении объектов, таких как игровые персонажи или машины, векторы используются для задания направления и расстояния перемещения. Например, если игровой персонаж должен двигаться вперед на определенное расстояние, вы можете использовать вектор (0, 0, 1), чтобы указать направление движения вперед, а затем умножить этот вектор на скорость и время движения, чтобы получить точку, в которой объект будет после перемещения.

Расчет силы и направления столконовений: При моделировании физики в игровом мире векторы используются для расчета сил, действующих на объекты, и

для определения направления столкновений. Например, если два объекта сталкиваются друг с другом, их скорости и массы могут быть использованы для расчета силы столкновения с помощью законов сохранения импульса. Эта сила может быть представлена вектором, указывающим направление и интенсивность столкновения.

Управление движением камеры: Векторы также могут использоваться для управления движением камеры в игровом мире. Например, для создания эффекта следования за игровым персонажем вы можете использовать вектор, указывающий на позицию персонажа, и плавно изменять позицию камеры в направлении этого вектора.

Определение направления света: Векторы также могут использоваться для определения направления света в игровом мире. Например, вектор нормали к поверхности объекта может использоваться для определения угла падения света на эту поверхность, что позволяет создавать реалистичные эффекты освещения и теней.

Кватернионы

Кватернионы - это математический объект, используемый для представления и описания ориентации и вращений в трехмерном пространстве. Они представляют собой более эффективный способ описания поворотов, чем обычные углы Эйлера. Вот как они отличаются:

Углы Эйлера: Углы Эйлера представляют собой три угла поворота вокруг осей x, y и z. Обычно это углы крена (roll), тангажа (pitch) и рысканья (yaw). Такое представление может вызывать проблемы из-за явления гимбальной блокировки, когда одна из осей выравнивается с другой, что приводит к потере одной степени свободы.

Кватернионы: Кватернионы представляют собой четырехмерные числа, состоящие из скалярной части и трех компонент вектора. Они используются для

представления вращений и ориентации в пространстве. Одна из главных преимуществ кватернионов - это отсутствие гимбальной блокировки и более удобное представление поворотов.

Главное отличие между кватернионами и углами Эйлера заключается в их математической природе и способе представления вращений. Кватернионы являются более компактным и эффективным способом представления ориентации, особенно в компьютерной графике и программировании игр. Они также обеспечивают плавные и безопасные операции вращения, что делает их предпочтительным выбором при разработке игр и анимаций.

Кватернионы используются для представления поворотов и ориентации объектов в трехмерном пространстве.

Кватернионы используются для представления поворотов и ориентации объектов в трехмерном пространстве, обеспечивая эффективный способ хранения и применения вращений.

Кватернионы представляют ориентацию объекта в трехмерном пространстве. Каждый кватернион состоит из скалярной (w) и векторной (x, y, z) компонент. Эта компонентная структура позволяет кватерниону представлять ось вращения и угол поворота вокруг этой оси.

Для применения поворота к объекту с использованием кватернионов, кватернион, представляющий поворот, умножается на кватернион, представляющий текущую ориентацию объекта. Результат этой операции будет новым кватернионом, представляющим ориентацию объекта после применения поворота. Этот процесс называется умножением кватернионов.

Кватернионы также используются для плавного перехода между различными ориентациями объектов. Этот процесс называется интерполяцией кватернионов. Он позволяет создавать анимации вращения и перемещения объектов с плавными и реалистичными переходами между различными состояниями.

Кватернионы являются более компактным и эффективным способом хранения ориентации объектов по сравнению с матрицами поворота. Они также обеспечивают более быстрые и стабильные вычисления вращений.

Кватернионы широко используются для анимации вращения объектов в игровых движках. Они позволяют плавно изменять ориентацию объекта с течением времени, что создает реалистичные и плавные анимации.

Использование кватернионов для представления поворотов и ориентации объектов в трехмерном пространстве обеспечивает эффективный и удобный способ управления вращениями и анимациями в игровых сценах.

1.3.2 Примеры использования кватернионов для плавных и корректных вращений объектов в игре.

Кватернионы широко используются в игровой разработке для плавных и корректных вращений объектов.

Вращение камеры. Кватернионы позволяют плавно и естественно вращать камеру вокруг объекта или в пространстве. Это позволяет создавать более реалистичные и плавные эффекты при управлении камерой в игре.

Анимация объектов.Кватернионы могут использоваться для анимации объектов в игре. Они позволяют плавно интерполировать между различными ориентациями объекта, что делает движение более естественным и реалистичным.

Управление игровыми персонажами. В играх, где игрок управляет персонажем, кватернионы могут использоваться для вращения и анимации персонажа. Это помогает сделать движение более плавным и реалистичным, особенно при выполнении сложных анимаций, таких как боевые приемы или акробатические трюки.

Физика. Кватернионы могут применяться в физическом моделировании для моделирования вращательных движений объектов. Они позволяют точно описать ориентацию объекта в пространстве и корректно применять физические законы к его вращательным движениям.

Визуальные эффекты. Кватернионы могут использоваться для создания различных визуальных эффектов, таких как вращение объектов вокруг своей оси, эффекты размытия при движении камеры и многие другие.

В целом, кватернионы предоставляют эффективный и удобный способ управления вращениями объектов в игре, обеспечивая плавность и корректность анимации.

1.3.3 Основные принципы физики твердого тела, такие как коллизии, динамика и гравитация.

Физика твердого тела изучает движение и взаимодействие твердых объектов под воздействием различных сил и условий. Основные принципы физики твердого тела включают в себя коллизии, динамику и гравитацию.

Коллизии происходят, когда два твердых объекта вступают в контакт друг с другом и взаимодействуют. В физике игр, системы коллизий используются для определения, когда и как объекты сталкиваются друг с другом, и расчета реакции на столкновение, такой как отскок, деформация или разрушение. Различные методы, такие как дискретные и непрерывные системы коллизий, используются для обнаружения и обработки столкновений в играх.

Динамика твердого тела описывает движение объектов под воздействием сил. Это включает в себя законы Ньютона, которые описывают отношения между силой, массой и ускорением объекта. В играх динамика твердого тела используется для моделирования движения объектов в пространстве, включая их вращение, сдвиг и взаимодействие с другими объектами.

Гравитация является силой притяжения между объектами с массой. В играх гравитация используется для моделирования воздействия земного притяжения на объекты, что создает реалистичные условия движения и поведения объектов в виртуальном мире.

Гравитация может также регулироваться или изменяться для создания уникальных игровых сценариев или эффектов.

В целом, понимание и учет этих основных принципов физики твердого тела позволяет создавать более реалистичные и интерактивные игровые миры, где объекты ведут себя естественным образом под воздействием различных сил и условий.

Рассмотрим, как Unity3D использует физический движок для симуляции поведения объектов в игровом мире. Unity3D использует встроенный физический движок под названием "PhysX" для симуляции поведения объектов в игровом мире. Вот как Unity3D использует физический движок для симуляции:

Компоненты коллайдеров. В Unity3D каждый объект, который должен участвовать в физическом взаимодействии, должен иметь компонент коллайдера. Коллайдеры определяют форму и размер объекта для определения столкновений с другими объектами.

Компоненты твердости. Unity3D позволяет определить, как объект должен реагировать на физические силы, с помощью компонента Rigidbody. Этот компонент определяет массу объекта, его ускорение, угловую скорость и другие параметры, связанные с физикой.

Материалы и настройки физических свойств. Unity3D позволяет настраивать различные свойства физических материалов, такие как трение, упругость и масса. Это позволяет создавать разнообразные эффекты в игровом мире, от скольжения

объектов по поверхности до реалистичного отскока и деформации при столкновениях.

Обработка столкновений и сил. PhysX в Unity3D обеспечивает обнаружение столкновений между объектами и расчет реакции на эти столкновения в соответствии с их физическими свойствами. Это включает в себя расчет силы и направления отталкивания, деформации объектов при столкновениях, а также возможность применения других физических эффектов, таких как гравитация и аэродинамические силы.

Интеграция с анимацией. Unity3D позволяет интегрировать физическую симуляцию с анимацией, что позволяет создавать более реалистичные и живые персонажи и объекты в игровом мире. Например, персонаж может реагировать на физические силы, такие как удары или гравитация, с помощью анимированных движений и реакций.

В целом, Unity3D использует физический движок для создания более реалистичных и интерактивных игровых миров, где объекты ведут себя в соответствии с принципами физики, что делает игровой опыт более увлекательным и убедительным.

Примеры настройки параметров физического движка Unity3D и использования коллайдеров для обнаружения столкновений между объектами.

```
using UnityEngine;

public class ExampleScript : MonoBehaviour
{
    Rigidbody rb;

    void Start()
    {
        rb = GetComponent<Rigidbody>();
        // Настройка массы объекта
        rb.mass = 2f;
        // Настройка коэффициента трения
        rb.drag = 0.5f;
        // Настройка коэффициента восстановления скорости после столкновения
        rb.angularDrag = 0.5f;
        // Настройка, является ли объект кинематическим
        rb.isKinematic = false;
        // И другие параметры...
    }
}
```

Рис. 2.1 – Настройки параметров Rigibody

```
using UnityEngine;

public class ExampleScript : MonoBehaviour
{
    Collider collider;

    void Start()
    {
        collider = GetComponent<Collider>();
        // Установка коллайдера в качестве триггера (только обнаружение столкновений)
        collider.isTrigger = false;
        // Включение/выключение коллайдера
        collider.enabled = true;
        // Изменение материала коллайдера для определения свойств столкновений
        collider.material = new PhysicMaterial();
        // И другие параметры...
    }
}
```

Рис. 2.2 – Настройки параметров Collider

```
using UnityEngine;

public class CollisionHandler : MonoBehaviour
{
```

Рис. 2.3 – Обработка столкновений с помощью OnCollisionEnter

```
using UnityEngine;

public class TriggerHandler : MonoBehaviour
{
    void OnTriggerEnter(Collider other)
    {
        // Обработка столкновений с триггером
        Debug.Log("Trigger with " + other.gameObject.name);
    }
}
```

Рис. 2.4 – Обработка столкновений с помощью OnTriggerEnter

Это простые примеры использования параметров физического движка Unity3D и коллайдеров для настройки столкновений между объектами и обработки этих столкновений в скриптах Unity.

В Unity3D GameObject и Transform являются основными строительными блоками для создания и управления объектами в игровом мире.

GameObject представляет собой базовый элемент в Unity3D, который может содержать компоненты и определять положение объекта в иерархии сцены. GameObject может быть чем угодно - персонажем, камерой, звуковым источником, фоновым объектом и т. д. Каждый GameObject может содержать один или несколько компонентов, определяющих его поведение и внешний вид. Transform определяет положение, поворот и масштаб GameObject в трехмерном пространстве. Каждый GameObject обязательно имеет компонент Transform. Transform состоит из трансформаций по оси X, Y и Z. Перемещение, вращение и масштабирование объекта происходит путем изменения значений его Transform.

```
void Start()
{
    // Создание нового экземпляра GameObject из префаба
    GameObject newObj = Instantiate(myObjectPrefab, Vector3.zero, Quaternion.identity);

    // Получение компонента Transform нового объекта
    Transform objTransform = newObj.transform;

    // Изменение позиции, поворота и масштаба объекта
    objTransform.position = new Vector3(0, 1, 0);
    objTransform.rotation = Quaternion.Euler(0, 45, 0);
    objTransform.localScale = new Vector3(2, 2, 2);
}
```

Рис. 2.5 – Пример создания и управления объектом в игровом мире

В этом примере:

- Создается новый экземпляр GameObject из префаба **myObjectPrefab**
- Получается компонент Transform нового объекта.
- Изменяются значения позиции, поворота и масштаба объекта с помощью Transform.

Это простой пример того, как GameObject используется для создания и управления объектами в игровом мире. В реальных проектах вы также можете добавлять и удалять компоненты, присваивать значения переменным и многое другое для создания интерактивных и сложных игровых сцен.

Компонент Transform играет ключевую роль в Unity, определяя положение, масштаб и ориентацию объекта в трехмерном пространстве. Давайте рассмотрим каждый аспект, как компонент Transform определяет их:

Положение объекта определяется трехмерным вектором (x, y, z), который указывает, где объект находится в трехмерном пространстве относительно начала координат.

Значение этого вектора задает, насколько далеко объект смещен относительно начальной точки координат.

Масштаб определяет размер объекта в пространстве.

Компонент Transform хранит трехмерный вектор (x, y, z), который определяет масштаб объекта по каждой оси.

Значения вектора масштаба указывают, во сколько раз размеры объекта должны быть увеличены или уменьшены по каждой оси.

Ориентация определяет направление, в котором объект смотрит, или его угловое положение в трехмерном пространстве.

Ориентация обычно выражается в виде кватерниона (Quaternion) или углов Эйлера (Euler angles).

В случае углов Эйлера, компонент Transform хранит углы поворота объекта вокруг каждой из трех осей (x, y, z).

Кватернион представляет собой комплексное число с векторной и скалярной частью, которое используется для более точного представления ориентации объекта в пространстве.

Таким образом, компонент Transform в Unity определяет положение объекта в трехмерном пространстве по его координатам, задает его размер с помощью масштаба и определяет его ориентацию с помощью поворота вокруг осей. Эти три аспекта совместно формируют пространственное положение и внешний вид объекта в игровом мире.

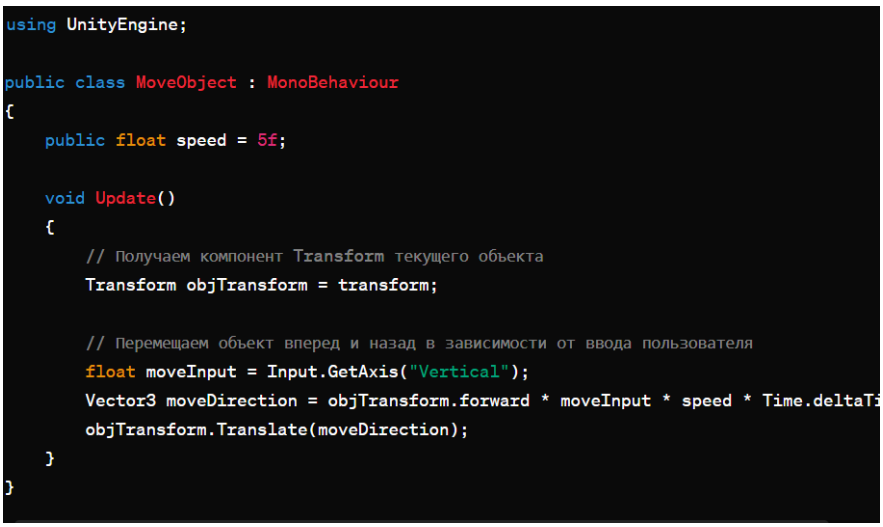
Ниже приведены примеры создания и манипулирования GameObject и Transform компонентами с помощью кода на языке C#.

```
using UnityEngine;

public class CreateObject : MonoBehaviour
{
    void Start()
    {
        // Создаем новый GameObject
        GameObject newObj = new GameObject("MyObject");

        // Получаем компонент Transform нового объекта
        Transform objTransform = newObj.transform;

        // Устанавливаем позицию, поворот и масштаб объекта
        objTransform.position = new Vector3(0, 0, 0);
        objTransform.rotation = Quaternion.Euler(0, 45, 0);
        objTransform.localScale = new Vector3(1, 1, 1);
    }
```

Рис. 2.6 – Пример создания GameObject и управления его Transform

```
using UnityEngine;

public class ManipulateObject : MonoBehaviour
{
    // Ссылка на существующий GameObject
    public GameObject existingObject;

    void Start()
    {
        // Получаем компонент Transform существующего объекта
        Transform objTransform = existingObject.transform;

        // Изменяем позицию, поворот и масштаб объекта
        objTransform.position = new Vector3(1, 2, 3);
        objTransform.rotation = Quaternion.Euler(0, 90, 0);
        objTransform.localScale = new Vector3(2, 2, 2);
    }
}
```

Рис. 2.7 – Пример получения существующего GameObject и манипулирования его Transform



```
using UnityEngine;

public class MoveObject : MonoBehaviour
{
    public float speed = 5f;

    void Update()
    {
        // Получаем компонент Transform текущего объекта
        Transform objTransform = transform;

        // Перемещаем объект вперед и назад в зависимости от ввода пользователя
        float moveInput = Input.GetAxis("Vertical");
        Vector3 moveDirection = objTransform.forward * moveInput * speed * Time.deltaTi
        objTransform.Translate(moveDirection);
    }
}
```

Рис. 2.8 – Пример перемещения GameObject вместе с клавишами управления

В этих примерах показано создание, получение и манипулирование GameObject и их Transform компонентами с помощью скриптования на языке C# в Unity.

1.4 Постановка задачи

Целью данного дипломного проекта является разработка и реализация трехмерной компьютерной игры с использованием современных технологий, в которой игрок управляет машиной, собирая звездочки на трассе в течение ограниченного времени.

Основные задачи:

Изучение литературы и анализ существующих игр. Провести обзор литературы и изучить существующие трехмерные игры, особенно в жанре гоночных аркад.

Проектирование игры. Разработать концепцию игры, включая геймплей, визуальный стиль, трассы и механику сбора звездочек.

Выбор технологий и инструментов разработки. Проанализировать различные движки и инструменты для создания трехмерных игр и выбрать наиболее подходящие с учетом задач проекта и имеющихся навыков.

Разработка игровых механик. Создать игровые механики с возможностью управления машиной, реализовать механику сбора звездочек и управление временем.

Создание трехмерных моделей и анимаций. Разработать трехмерные модели машин, звездочек и окружающей среды, а также создать анимации для персонажей и объектов.

Программирование и тестирование. Написать программный код игры с использованием выбранных технологий, а также провести тестирование для выявления и устранения ошибок и недочетов.

Оформление и документирование. Создать интерфейс пользователя, включая меню, настройки и экраны победы/поражения, а также оформить документацию к игре.

Ожидаемые результаты:

В результате выполнения дипломного проекта ожидается создание полноценной трехмерной компьютерной игры, способной обеспечить увлекательное игровое испытание для пользователей и демонстрирующей уровень профессионализма и навыков разработчика.

Глава 2. Практическая часть

2.1 Структура разработанного программного обеспечения

В соответствии с поставленной задачей разработана следующая структура программы.

1. **Игровой мир (Game World):**
- Создание 3D-модели мира, включая террейн, объекты окружения и звездочки, которые машина должна собирать.
2. **Персонаж (Character):**
- Модель машины, которую игрок будет управлять.
- Управление движением машины с помощью клавиатуры или сенсорных устройств.
3. **Звездочки (Stars):**
- Создание и размещение объектов "звездочка" в игровом мире.
- Логика их поведения (например, исчезновение после сбора).
4. **Таймер (Timer):**
- Реализация таймера, отсчитывающего 30 секунд.
- Отображение оставшегося времени на экране.
5. **Счетчик очков (Score Counter):**
- Отслеживание количества собранных звездочек.
- Отображение текущего счета на экране.
6. **Логика игры (Game Logic):**
- Определение условий победы (например, сбор определенного количества звездочек за отведенное время).
- Обработка событий завершения игры (победа или поражение).
7. **Управление сценами (Scene Management):**
- Создание необходимых сцен (главное меню, игровая сцена, экран завершения игры и т. д.).

- Управление переходами между сценами.

8. **Интерфейс пользователя (User Interface):**

- Создание элементов интерфейса, таких как кнопки, текстовые поля и т. д.
- Отображение информации о счете, времени и т. д.

9. **Звуковые эффекты (Sound Effects):**

- Добавление звуковых эффектов для действий в игре (сбор звездочек, таймер, победа/поражение и т. д.).

Это базовая структура программы. Каждый из этих компонентов будет иметь свои собственные скрипты (или классы, в случае Unity) для реализации необходимой функциональности.

2.2 Разработка основных программных элементов

В соответствии с разработанной структурой рассмотрим реализацию основных программных компонентов игры.

1. **Игровой мир (Game World):**

   Создайте 3D-модели мира, объектов окружения и звездочек в Unity.

   Разместите эти объекты на сцене, чтобы создать игровой мир.

2. **Персонаж (Character):**

   Создайте модель машины и добавьте ее на сцену.

Напишите скрипт управления движением машины. Этот скрипт должен реагировать на ввод с клавиатуры или сенсорных устройств и перемещать машину соответственно.

3. **Звездочки (Stars):**

Создайте префаб (prefab) звездочки в Unity.

Напишите скрипт, который будет управлять поведением звездочки. Например, звездочка может исчезать при контакте с машиной.

Разместите звездочки на сцене в начале игры.

4. **Таймер (Timer):**

Напишите скрипт таймера, который будет отсчитывать время и обновлять отображение оставшегося времени на экране.

Начните отсчет времени при старте игры и завершите игру после прохождения 30 секунд.

5. **Счетчик очков (Score Counter):**

Создайте переменную для хранения текущего счета.

Напишите скрипт, который будет увеличивать счет при сборе каждой звездочки.

Обновляйте отображение счета на экране.

6. **Логика игры (Game Logic):**

Напишите скрипт, который будет проверять условия победы и поражения.

Если игрок собрал достаточное количество звездочек за отведенное время, то это победа. Если время истекло до того, как игрок собрал достаточно звездочек, это поражение.

После завершения игры отобразите экран с результатами и возможностью начать игру заново.

7. **Управление сценами (Scene Management):**

Настройте сцены в Unity для главного меню, игровой сцены и экрана завершения игры.

Напишите скрипт, который будет управлять переходами между этими сценами.

8. **Интерфейс пользователя (User Interface):**

Создайте интерфейс с отображением счета и оставшегося времени.

Напишите скрипт, который будет обновлять эти элементы интерфейса в соответствии с изменениями счета и времени.

9. **Звуковые эффекты (Sound Effects):**

Добавьте звуковые эффекты для событий в игре, такие как сбор звездочки или завершение игры.

Движение машины вперед/назад.

В новом проекте добавлена плоскость для расположения модели машины, определены ее размеры и центральное. Поскольку Unity поддерживает формат FBX для импорта трехмерных моделей, используемая модель сконвертирована в этот формат (рис. 3.1). Затем с помощью встроенных в Unity средств импортирована в проект (пункт *Assets* > *Import New Assets*).

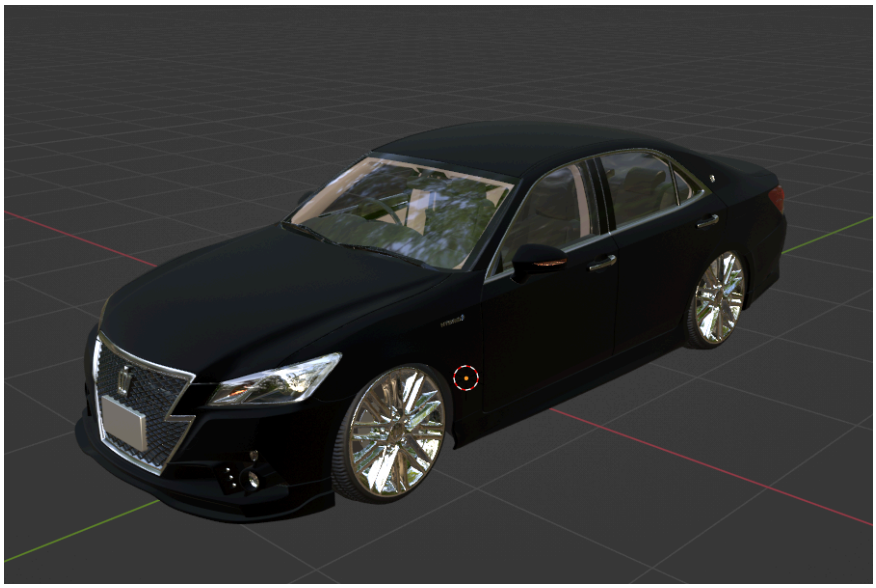


Рисунок 3.1 – Внешний вид модели машины

После импорта модели в Unity, создан игровой объект машины (GameObject) с именем «Player». К нему также добавлены компоненты RigidBody и BoxCollider, для обеспечения симуляции физических эффектов, таких как столкновения и действия силы тяжести. Параметры компонентов приведены на рис.3.2.

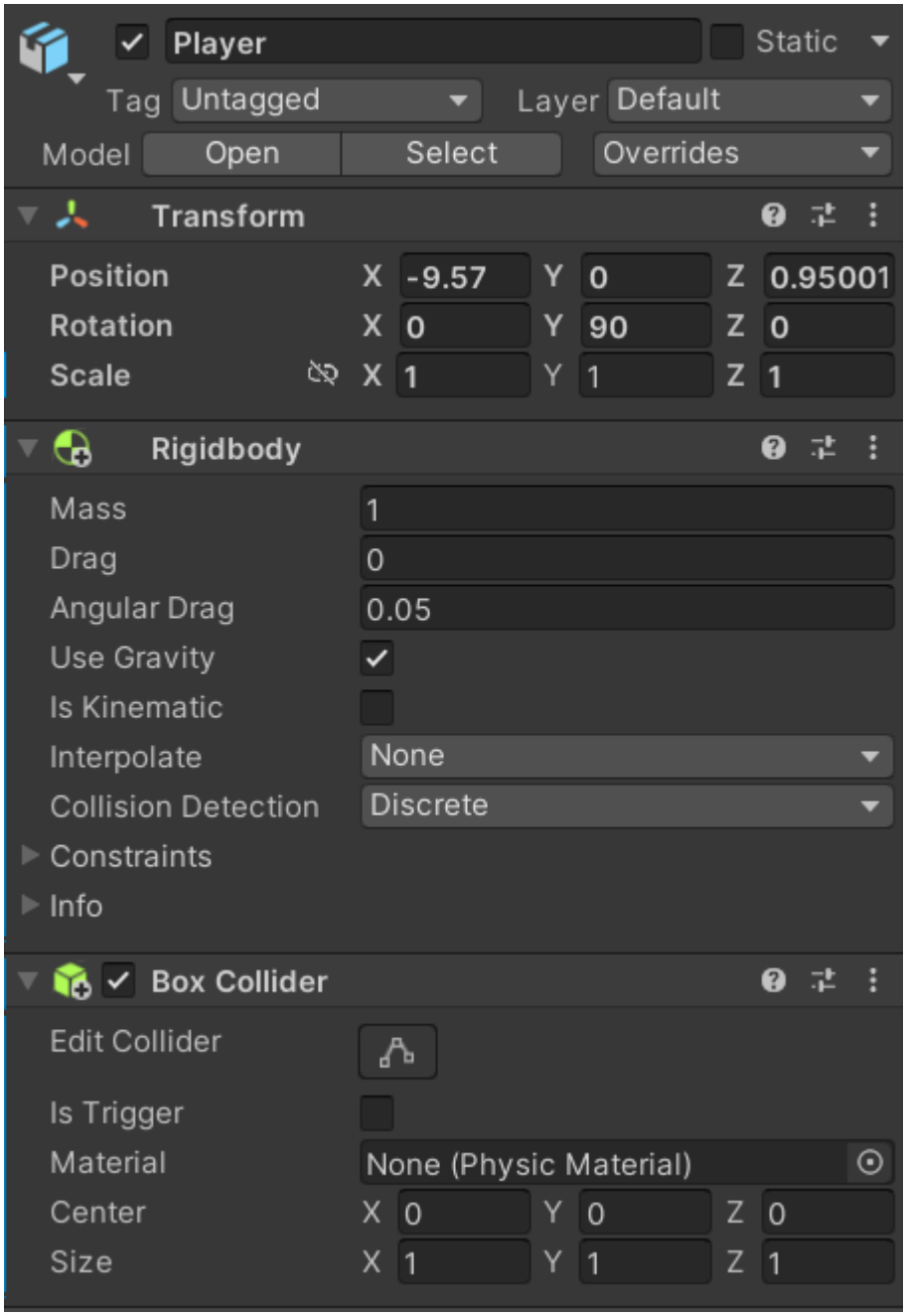


Рисунок 3.2 – Параметры компонентов RigidBody и BoxCollider машины

В качестве метода управления машиной выбран UI интерфейс, в котором расположены две кнопки, отвечающие за две педали. При нажатии на первую педаль машина двигается вперед, при нажатии на вторую – назад. Каждая кнопка создана в виде элемента Button (UI > Button), к которым добавлены изображения педалей в формате PNG (рис. 3.3).

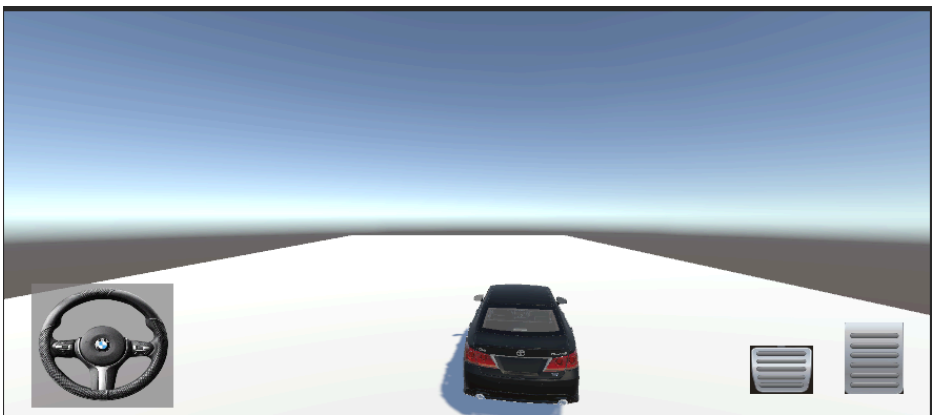
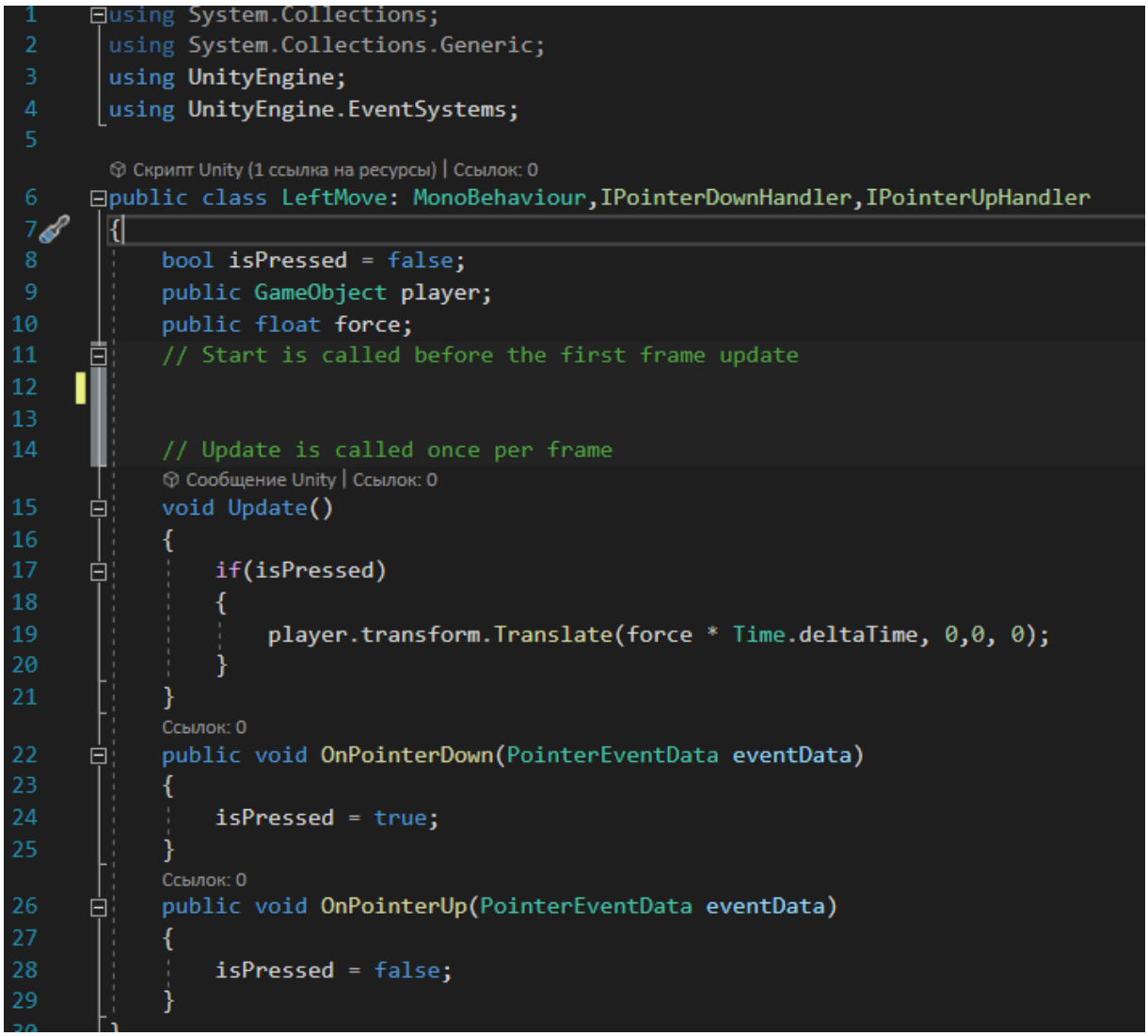


Рисунок 3.3 – Внешний вид интерфейса управления машиной

Для обеспечения работы кнопок и движения машины написан скрипт *LeftMove.cs* (рис. 3.4).



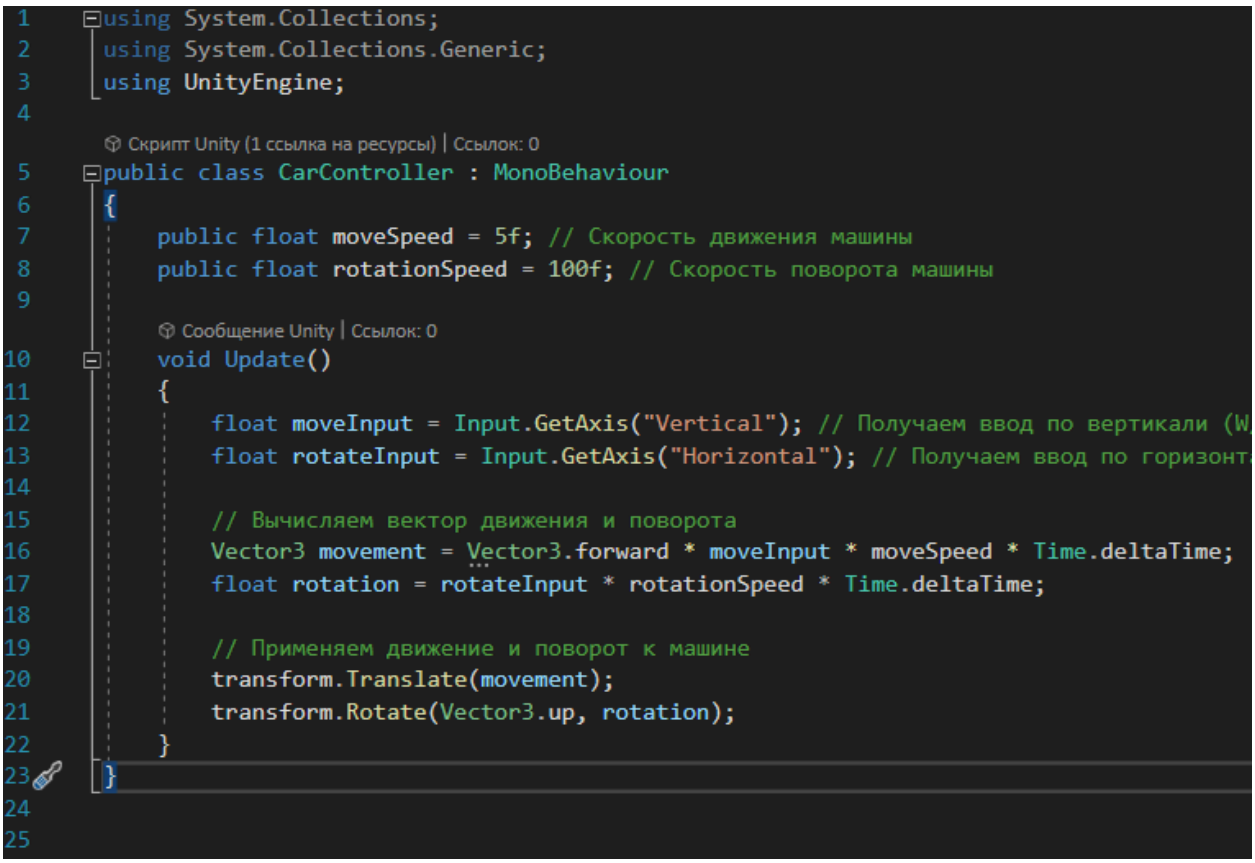
```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.EventSystems;

Скрипт Unity (1 ссылка на ресурсы) | Ссылок: 0
public class LeftMove: MonoBehaviour,IPointerDownHandler,IPointerUpHandler
{
    bool isPressed = false;
    public GameObject player;
    public float force;
    // Start is called before the first frame update


    // Update is called once per frame
    Сообщение Unity | Ссылок: 0
    void Update()
    {
        if(isPressed)
        {
            player.transform.Translate(force * Time.deltaTime, 0,0, 0);
        }
    }
    Ссылок: 0
    public void OnPointerDown(PointerEventData eventData)
    {
        isPressed = true;
    }
    Ссылок: 0
    public void OnPointerUp(PointerEventData eventData)
    {
        isPressed = false;
    }
```

Рисунок 3.4 – Скрипт обеспечивающий движение машины при нажатии кнопки

В качестве параметров для управления в скрипте указывается скорость и направление движения.

Движение машины вперед/назад направо/налево с помощью стрелок клавиатуры

Для управления машиной вперед и назад, направо и налево с помощью стрелок клавиатуры , создан скрипт *CarController* (рис.3.5).



```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

Скрипт Unity (1 ссылка на ресурсы) | Ссылок: 0
public class CarController : MonoBehaviour
{
    public float moveSpeed = 5f; // Скорость движения машины
    public float rotationSpeed = 100f; // Скорость поворота машины

    Сообщение Unity | Ссылок: 0
    void Update()
    {
        float moveInput = Input.GetAxis("Vertical"); // Получаем ввод по вертикали (W
        float rotateInput = Input.GetAxis("Horizontal"); // Получаем ввод по горизонт

        // Вычисляем вектор движения и поворота
        Vector3 movement = Vector3.forward * moveInput * moveSpeed * Time.deltaTime;
        float rotation = rotateInput * rotationSpeed * Time.deltaTime;

        // Применяем движение и поворот к машине
        transform.Translate(movement);
        transform.Rotate(Vector3.up, rotation);
    }
}
```

Рисунок 3.5 – Скрипт управление машиной с помощью стрелок клавиатуры

Присоединена этот скрипт к объекту машины.

Этот скрипт отслеживает нажатия клавиш W/S или стрелок вверх/вниз (вертикальное движение) A/D или стрелок вправо/влево (горизонтальное движение) и изменяет скорость машины соответственно. Он использует компонент Transform для изменения скорости машины.

Для того чтобы реализовать вращение руля при нажатии и перемещении мыши, создан скрипт *SteeringWheelController* (рис.3.6), который будет отслеживать движения мыши и преобразовывать в углы поворота руля.

```
    public Transform steeringWheel; // Ссылка на объект руля
    public float rotationSpeed = 10f; // Скорость вращения руля

    private bool isDragging = false; // Флаг для отслеживания нажатия мыши
    private Vector3 lastMousePosition; // Последняя позиция мыши
    private Quaternion _initPosition;
    Сообщение Unity | Ссылок: 0
    private void Start()
    {
        _initPosition = steeringWheel.rotation;
    }

    Сообщение Unity | Ссылок: 0
    void Update()
    {
        if (Input.GetMouseButtonDown(0))
        {
            isDragging = true;
            lastMousePosition = Input.mousePosition;
        }

        if (Input.GetMouseButtonUp(0))
        {
            isDragging = false;
            steeringWheel.rotation = _initPosition;
        }

        if (isDragging)
        {
            Vector3 mouseDelta = Input.mousePosition - lastMousePosition;
            float rotationAmount = mouseDelta.x * rotationSpeed * Time.deltaTime;

            // Применяем вращение к рулю
            steeringWheel.Rotate(Vector3.back, rotationAmount);

            lastMousePosition = Input.mousePosition;
        }
    }
```

Рисунок 3.6 – Скрипт вращение руля при нажатии и перемещении мыши

Этот скрипт отслеживает нажатие и перемещение мыши при зажатой левой кнопке. Он вычисляет разницу позиций мыши между кадрами и преобразует ее в угол поворота руля, который затем применяется к объекту руля:

```
                Vector3 mouseDelta = Input.mousePosition - lastMousePosition;
float rotationAmount = mouseDelta.x * rotationSpeed * Time.deltaTime;

// Применяем вращение к рулю
steeringWheel.Rotate(Vector3.back, rotationAmount
lastMousePosition = Input.mousePosition;
```

Кнопка создана в виде элемента Button (UI > Button), к которой добавлено изображение руля в формате PNG (рис. 3.7). и присоединен в виде элемента

Transform к переменной SteeringWheel скрипта *SteeringWheelController* в скрипте через инспектор.

Рисунок 3.7 – Внешний вид интерфейса управления машиной

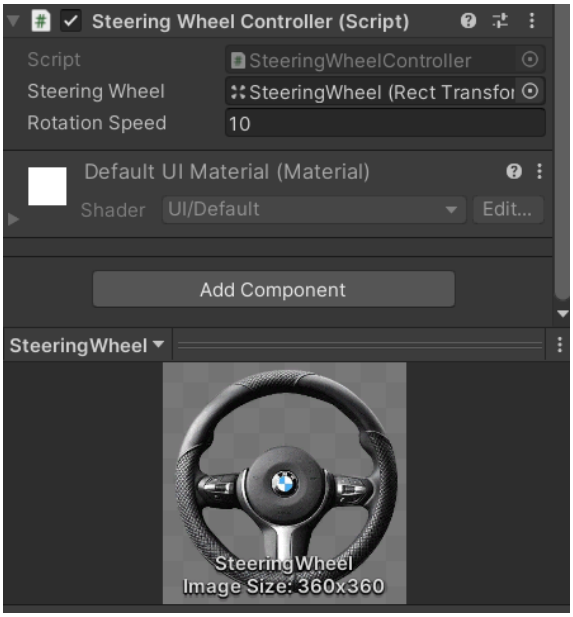


Рисунок 3.8 – Присоединение объекта к переменной *SteeringWheelController*

Появление звездочек на дороге

С помощью встроенных в Unity средств импортирована в проект (пункт *Assets > Import New Assets*).модель звездочки (Рис. 3.9)

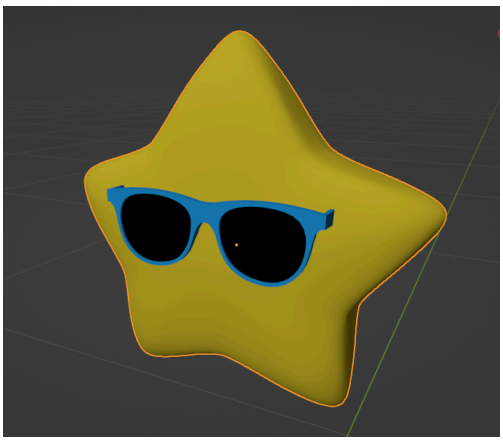


Рисунок 3.9 –Звездочка

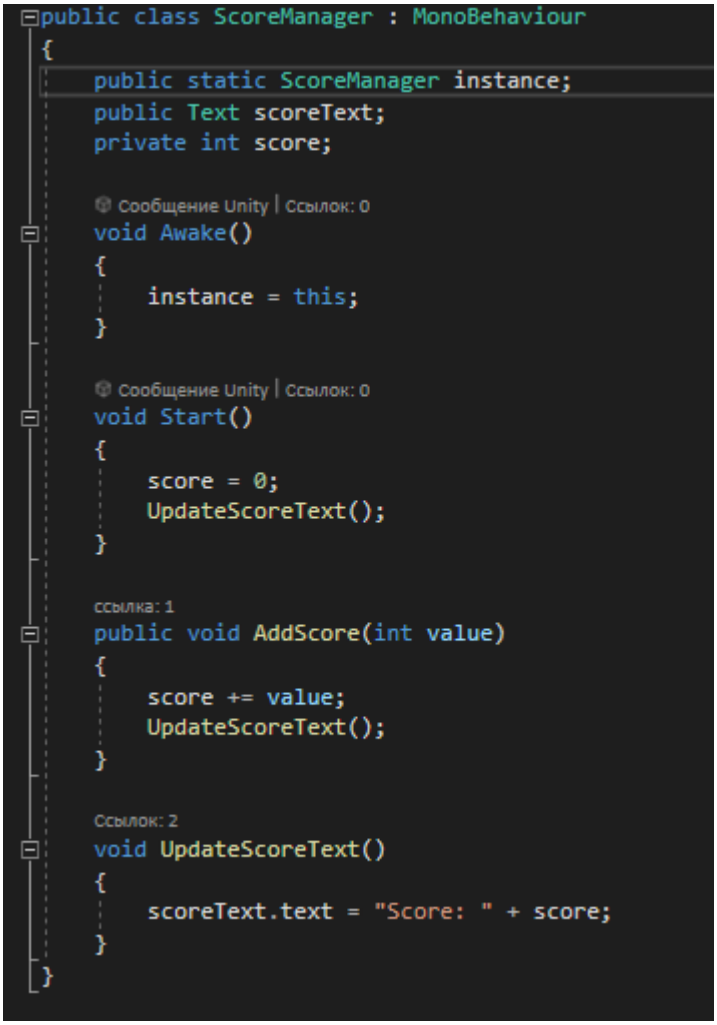
Создан префаб для зведочки (создать папку Prefabs и перетащить в нее звездочку, выбрав вариант «Original Prefab».) Префаб содержит модель звездочки и компонент Collider для обнаружения столкновений. Поставлен префаб на дороге в нужных позициях.

Набор очков при столкновении

Написан скрипт, который будет обрабатывать столкновения машины с звездочками и увеличивать счетчик очков.

```
public class StarScript : MonoBehaviour
{
  public int scoreValue = 10;

  void OnTriggerEnter(Collider other)
  {
   Debug.Log("Trigger");

    if (other.CompareTag("Player"))
    {
      ScoreManager.instance.AddScore(scoreValue);

            Destroy(gameObject);
    }
```

Создан новый скрипт (рис. 3.10) с названием , «ScoreManager» для управления счетом



```
public class ScoreManager : MonoBehaviour
{
    public static ScoreManager instance;
    public Text scoreText;
    private int score;

    Сообщение Unity | Ссылок: 0
    void Awake()
    {
        instance = this;
    }

    Сообщение Unity | Ссылок: 0
    void Start()
    {
        score = 0;
        UpdateScoreText();
    }

    ссылка: 1
    public void AddScore(int value)
    {
        score += value;
        UpdateScoreText();
    }

    Ссылок: 2
    void UpdateScoreText()
    {
        scoreText.text = "Score: " + score;
    }
}
```

Рисунок 3.10 – Скрипт

Присвоен тег «Player» машине с которым будет столкновение. Создан UI Text объект для отображения счета и присвоен скрипту управления счетом (ScoreManager)

Настройка машины

Создан скрипт (CarScript), который будет отвечать за движение машины и обработку столкновений.

```
        void OnTriggerEnter2D(Collider2D other)
{
    if (other.CompareTag("Star")) // Проверка столкновения с звездочкой
    {
        Debug.Log("Star collected!");
        // Дополнительные действия при сборе звездочки
    }
}
```

Затем присвоен тег «Star» звездочкам в сцене , чтобы обеспечить корректную обработку столкновений. Также этот добавлен скрипт к объекту машины.

Перемещение звездочки в случайное место

Чтобы игра была интереснее изменен код, так что при столкновении с машиной, звездочка перемещается в случайное место.

```
                if (other.CompareTag("Player")) // Проверка столкновения с машиной
    {
        ScoreManager.instance.AddScore(scoreValue); // Увеличение счета
        //Destroy(gameObject); // Удаление звездочки

        // вместо уничтожения - переместить в случайное место

        float randomX = Random.Range(-1.89f, 5f);
        float randomY = Random.Range(-3f, 3f);

        // Перемещаем звездочку в новую позицию
        transform.position = new Vector3(randomX, transform.position.y, randomY);
    }
```

При столкновении с объектом, имеющим тег «Car», звездочка перемещается в случайное место в пределах определенной области (в данном случае, от -5 до 5 по X и от -3 до 3 по Y).

Вращение звездочек вокруг себя

Для создания эффекта вращения звездочек вокруг себя , разработан скрипт и базовая графика для звездочек. Вот пример скрипта на C#.

```
using UnityEngine;

public class StarRotation : MonoBehaviour
{
    public float rotationSpeed = 50f; // Скорость вращения звездочек

    void Update()
    {
        // Вращаем объект вокруг оси Y с постоянной скоростью
        transform.Rotate(Vector3.up * rotationSpeed * Time.deltaTime);
    }
}
```

Далее данный скрипт присоединен к объекту звездочки. Затем в инспекторе указан желаемая скорость вращения **rotationSpeed**. Звездочка будет вращаться вокруг себя с этой скоростью.

Свеченение звездочек

Для того чтобы звездочки было легче увидеть в сцене, вы применен к ним различное освещение.

1. **Создание материала для звездочек:** Изменен существующий материал. Установлен для этого материала цвет и другие свойства, такие как блеск или текстуры, если необходимо. Главное, чтобы у материала было эмиссивное свойство, которое заставит его светиться.
2. **Включение эмиссии:** В настройках материала есть параметр "Emission". Включен выбран цвет свечения, который я хочу для нашей звездочки.
3. **Настройка интенсивности свечения:** Я могу регулировать интенсивность свечения, меняя значение параметра "Emission". Это позволит мне создавать светящиеся звездочки с разной яркостью.
4. **Применение материала к объекту звездочки:** Назначен нашему объекту звездочки созданный материал.

После этого наша звездочка будет светиться в соответствии с настройками материала. Этот метод позволяет создавать реалистичные и красочные светящиеся эффекты для звездочек в вашей сцене. Для реализации логики, при которой игра заканчивается через 30 секунд, если игрок не столкнулся со звездочкой, использован скрипт. Вот пример того, как это можно сделать:

```
using UnityEngine;
public class GameManager : MonoBehaviour
{
   public float gameTime = 30f; // Время игры в секундах
   private float timer = 0f;
```

```
    private bool gameIsOver = false;
    void Update()
    {
        if (!gameIsOver)
        {
            timer += Time.deltaTime;
            // Проверяем, прошло ли 30 секунд
            if (timer >= gameTime)
            {
                EndGame();
            }
        }
    }
    public void EndGame()
    {
        Debug.Log("Игра окончена!");
        gameIsOver = true;
    }
    public void StarCollected()
    {
        // Вызывается, когда игрок сталкивается со звездочкой
        // Сбросить таймер, когда игрок сталкивается со звездочкой
        timer = 0f;
    }
}
```

Присоединен этот скрипт к объекту GameManager в нашей сцене. Затем создан другой скрипт, который будет присоединен к звездочке и вызывать метод **StarCollected()** при столкновении игрока со звездочкой. Это сбросит таймер, предотвращая окончание игры.

```
using UnityEngine;

public class Star : MonoBehaviour
{
    private void OnTriggerEnter(Collider other)
    {
        if (other.CompareTag("Player"))
        {
                                        GameManager gameManager =
FindObjectOfType<GameManager>();
            if (gameManager != null)
            {
```

```
                gameManager.StarCollected();
            }
            Destroy(gameObject);
        }
    }
}
```

Теперь, когда игрок сталкивается со звездочкой, таймер сбрасывается. Если игрок не столкнулся со звездочкой в течение 30 секунд, вызывается метод **EndGame()**, завершающий игру.

Добавление времени.

Для добавления времени, когда игрок набирает звездочку, модифицирован скрипт GameManager так, чтобы он учитывал это дополнительное время.

```
using UnityEngine;
using UnityEngine.UI;
public class GameManager : MonoBehaviour
{
public float gameTime = 30f; // Изначальное время игры в секундах
public float timeToAdd = 3f; // Время, которое добавляется при сборе
звездочки
private float timer = 0f;
private bool gameIsOver = false;
// Добавим ссылку на текстовый UI элемент, чтобы показывать время
public Text timeText;
void Start()
{
// Обновим текстовое поле времени в начале игры
UpdateTimeUI();
}
void Update()
{
if (!gameIsOver)
{
timer += Time.deltaTime;
// Проверяем, прошло ли время
if (timer >= gameTime)
{
EndGame();
}
// Обновляем текстовое поле времени
UpdateTimeUI();
}
}
void UpdateTimeUI()
{
```

```
// Показываем оставшееся время в текстовом поле
float timeLeft = Mathf.Max(0f, gameTime - timer);
timeText.text = "Time: " + Mathf.RoundToInt(timeLeft) + "s";
}
public void EndGame()
{
// Логика окончания игры
Debug.Log("Game Over!");
gameIsOver = true;
}
public void StarCollected()
{
// Вызывается, когда игрок набирает звездочку
// Добавляем время при сборе звездочки
timer -= Mathf.Min(timeToAdd, gameTime - timer);
}
}
```

Теперь каждый раз, когда игрок набирает звездочку, к таймеру добавляется время (не более, чем значение **timeToAdd**). Обновленный скрипт также обновляет текстовое поле, показывая оставшееся время.

Добавление иконки.

Добавлен рисунки (время, звездочки). Иконка времени поставлен на левом верхнем углу, рядом с времени, чтобы было естественнее.

Звездочка тоже поставлен на верхнем левом углу, внизу времени, рядом очков звездочки.

Создание кнопки в сцене

Создайте UI кнопку в вашей сцене. Это можно сделать, выбрав GameObject -> UI -> Button.

Назначение метода презезапуска кнопке

Создан скрипт, который будет управлять перезапуском игры.

```
using UnityEngine;
using UnityEngine.SceneManagement;

public class RestartGame : MonoBehaviour
{
    public void Restart()
    {
        // Перезапуск загрузки текущей сцены
        SceneManager.LoadScene(SceneManager.GetActiveScene().name);
    }
}
```

Рисунок 3.11 – Скрипт

Привязка метода к кнопке

Перетащила скрипт RestartGame на вашу кнопку в редакторе Unity.

В компоненте Button у кнопки найден список On Click (). Нажат на "+" и добавьте объект с скриптом RestartGame.

В списке событий выбран метод Restart ().

Теперь, когда игрок нажимает на эту кнопку во время игры, метод Restart () будет вызываться, и текущая сцена будет перезагружена, начиная игру заново.

2.3 Тестирование и отладка

После этапа разработки игры была проведена её тестирование и отладка.

Настройка проекта для экспорта на Android:

Убедитесь, что ваши настройки проекта в Unity подходят для экспорта на Android.

Для этого перейдите в "File" (Файл) -> "Build Settings" (Параметры сборки).

В окне "Build Settings" выберите платформу Android.

Если ваша сцена не добавлена в сборку, добавьте ее с помощью кнопки "Add Open Scenes" (Добавить открытые сцены).

Настройка платформы Android:

Нажмите на кнопку "Player Settings" (Настройки игрока) в окне "Build Settings".

В разделе "Player Settings" (Настройки игрока) убедитесь, что настройки для Android установлены корректно. Это включает в себя настройки имени пакета, версии приложения, иконок и т. д.

Настройка среды разработки Android:

Установите и настройте Android SDK (Software Development Kit) и JDK (Java Development Kit) на вашем компьютере, если они еще не установлены.

В Unity откройте окно "Edit" (Правка) -> "Preferences" (Настройки), выберите вкладку "External Tools" (Внешние инструменты) и укажите путь к вашему Android SDK.

Настройка устройства Android:

Убедитесь, что ваше устройство Android подключено к компьютеру и настроено для разработки (USB-отладка включена).

Сборка проекта:

После настройки всех параметров экспорта для Android, нажмите кнопку "Build" (Собрать) в окне "Build Settings".

Выберите место для сохранения собранного проекта Android (Unity создаст APK файл).

Дождитесь завершения процесса сборки.

Тестирование на устройстве:

Перенесите APK файл на ваше Android устройство.

На устройстве откройте файловый менеджер и найдите APK файл, затем установите его.

После установки запустите игру и протестируйте ее на устройстве Android.

Отладка и оптимизация:

После тестирования убедитесь, что игра работает правильно и оптимизирована для устройств Android.

Проведите отладку для выявления и исправления возможных проблем.

Публикация в магазине Google Play (по желанию):

Если вы хотите опубликовать свою игру в Google Play, создайте аккаунт разработчика Google Play и следуйте инструкциям по публикации игры.

Рисунок 4 – Установленная игра на мобильном устройстве Android

А также работа была представлена на выставке под названием StudExpo.

StudExpo - Международная студенческая выставка "**StudEXPO-2024**", приуроченная к 85-летию Ошского государственного университета. К участию в ярмарке были приглашены представители ВУЗов России, Узбекистана.

Рисунок 4.1 – Демонстрация разработанной игры на выставке StudExpo 2024

Заключение

В рамках данного дипломного проекта была разработана и реализована трехмерная компьютерная игра под названием "Звездная Гонка", представляющая собой гоночный симулятор, где игрок управляет машиной, собирая звездочки на трассе в течение ограниченного времени. Проект позволил продемонстрировать навыки в области разработки игр, включая проектирование, программирование, создание графики и анимаций, а также тестирование и оптимизацию игрового процесса.

Одним из ключевых достижений проекта является создание игровых механик, которые обеспечивают плавную и реалистичную анимацию машин, динамичное управление и интерактивное взаимодействие с окружающей средой. Также была реализована интуитивно понятная механика сбора звездочек, которая добавляет стратегический элемент в игровой процесс, позволяя игрокам эффективно управлять своим временем и стремиться к установлению новых рекордов.

Разработанное решение было протестировано в рамках международной студенческой конференции StudExpo 2024. Экспериментальное тестирование игры показало положительные результаты и подтвердило ее потенциал как увлекательного развлечения для широкой аудитории игроков. Полученный опыт в процессе работы над проектом позволил лучше понять особенности разработки трехмерных игр и развить навыки в области программирования, анимации и дизайна.

Список литературы

1. Adams, E. (2012). Fundamentals of Game Design (3rd Edition). New Riders.
2. Hart, J. (2014). The Art of Game Design: A Book of Lenses (2nd Edition). CRC Press.
3. Fullerton, T., Swain, C., & Hoffman, S. (2008). Game Design Workshop: A Playcentric Approach to Creating Innovative Games (2nd Edition). CRC Press.
4. Rabin, S. M., & Chandler, S. (2016). Introduction to Game Development (2nd Edition). CRC Press.
5. Rollings, A., & Morris, D. (2004). Game Architecture and Design: A New Edition. New Riders.
6. Шестаков, Е.И., Жолдошов, Т.М. Разработка игр на платформе Unity3D: учебно-метод. Пособие – Ош: ОшГУ, 2024. – 108 с.
7. Gamasutra (https://www.gamasutra.com/)
8. Unity Learn (https://learn.unity.com/)
9. Unreal Engine Documentation (https://docs.unrealengine.com/)
10. Unity Official Website (https://unity.com/)
11. Smith, J. (2019). "The Impact of Time Constraints on Gameplay Experience: A Study of Racing Games." Journal of Game Research, 10(2), 123-137.
12. Johnson, L., & Brown, K. (2018). "Player Motivation in Racing Games: A Comparative Analysis of Player Preferences." International Conference on Game Studies Proceedings, 45-56.