МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное бюджетное образовательное учреждение
высшего профессионального образования
«Южно-Уральский государственный университет»
(национальный исследовательский университет)
Факультет «Высшая школа экономики и управления»
Кафедра «Информационные технологии в экономике»

РАБОТА ПРОВЕРЕНА

Рецензент, руководитель IT-отдела

_____/ П.П. Мартынюк /

« __ »_____ 20____ г.

ДОПУСТИТЬ К ЗАЩИТЕ

Зав. кафедрой, д.т.н., с.н.с.

_____/ Б.М. Суховилов/

« __ »_____ 20____ г.

Разработка симулятор авиртуальной реальности»

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА
ЮУрГУ–09.03.03.2017.087 ВКР

Консультант, доцент, к.т.н.
_____/_О.И. Галичин/
«_____»_____2017 г.

Руководитель, к.т.н., доцент
_____/А.Г. Палей/
«_____»_____2017 г.

Автор
студент группы ВШЭУ-436
_____/Я. Гасанов/
«_____»_____2017 г.

Нормоконтролер, доцент
_____/Е.А. Конова/
«_____»_____2017 г.

Челябинск 2017

АННОТАЦИЯ

Гасанов Я. Разработка симулятора виртуальной реальности – Челябинск: ЮУрГУ, ВШЭУ-436, 41 с, 15 ил., 3 табл., библиогр. список – 8 наимен., 0 прил.

Разработано приложение «Симулятор виртуальной реальности» для мобильных устройств.

Приведено краткое описание технологии виртуальной реальности, инструмента для разработки трехмерных приложений Unity, Sketchup, 3Ds max.

Описана архитектура приложения, представлена разработка приложения.

ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ 9
1. ПОСТАНОВКА ЗАДАЧИ 10
1.1 Определение требований к функциональности 10
1.2 Актуальность 10
1.3 Обзор аналогичных разработок 11
1.3 Сравнительный анализ программных продуктов 13
1.4 Обоснование разработки собственного решения 14
Выводы по первому разделу 14
2. ВИРТУАЛЬНАЯ РЕАЛЬНОСТЬ 15
Выводы по второму разделу 16
3. ПРОЕКТИРОВАНИЕ ПРИЛОЖЕНИЯ 17
3.1 Построение модели 17
3.2 Диаграмма классов 20
3.3 Выбор инструмента 21
Выводы по третьему разделу 22
4. РАЗРАБОТКА ПРИЛОЖЕНИЯ 23
4.1 Описание Unity как инструмента разработчика 23
4.2 Разработка игровых объектов 24
4.2.1 Игровой объект автомобиль 24
4.2.2 Игровой объект «компьютерный соперник» 29
4.2.3 Игровой объект « карта» 31
4.2.4 Игровой объект «GameProcess» 32
4.2.5 Игровой уровень 33
4.3 Sketchup 34
4.4 Autodesk 3ds Max 35
Выводы по четвертому разделу 36
5. ЭКОНОМИЧЕСКИЙ РАЗДЕЛ 37
Выводы по пятому разделу 38
ЗАКЛЮЧЕНИЕ 39
БИБЛИОГРАФИЧЕСКИЙ СПИСОК 40

ВВЕДЕНИЕ

Целью работы является разработка приложения симулятора виртуальной реальности, позволяющее управлять автомобилем. Приложение может быть использовано для получения навыков вождения или как игра.

Для достижения цели поставлены и решены следующие задачи:

- выполнен анализ предметной области;
- определены требования к наполнению и функциям приложения;
- разработана инфологическая модель: диаграмма прецедентов, классов;
- обоснован выбор инструментов для решения задачи;
- освоена среда разработки Unity;
- приложение реализовано и протестировано.

1. ПОСТАНОВКА ЗАДАЧИ

1.1 Определение требований к функциональности

Требуется разработать приложение «Симулятор виртуальной реальности» Приложение должно использовать технологии виртуальной реальности. В приложении должен быть реализован базовый функционал автосимулятора: перемещение автомобиля по игровому треку и игровой карте. Необходимо определить структуру приложения, разработать классы:

- обеспечивающие движение автомобиля по законам физики;
- обеспечивающие реализацию игрового искусственного интеллекта;
- управляющие процессом игры.

В приложении должны быть реализованы возможности расширения без изменения исходного кода приложения:

- добавление автомобилей с разными техническими характеристиками;
- добавление игровых уровней.

1.2 Актуальность

Согласно исследованию Компании J'son & Partners Consulting, рынок игр во всем мире является самым большим сегментом мирового рынка цифрового контента [6]. Доходы рынка игр в 2016 году составили 116млрд. долларов.

Самым быстрорастущим сегментом являются мобильные игры, совокупный среднегодовой темп роста которых составляет 23%. Доходы сегмента составили 36,9 млрд долл. Этому способствует распространение смартфонов и планшетов, улучшение их функционала, а также расширение покрытия 3G и 4G [7].

Согласно исследованию Statistica, к 2020 году индустрия виртуальной реальности будет оцениваться в 30 млрд. долларов, прибыль от программных продуктов виртуальной реальности к 2018 году вырастет почти в 60 раз, количество пользователей к этому же году возрастёт до 171 млн. человек[8].

1.3 Обзор аналогичных разработок

На рынке игр существуют приложения в жанре «автосимулятор», в обзоре представлены самые популярные приложения.

Asphalt Nitro

Главная заставка приложения Asphalt Nitro представлена на рисунке 1.

Рисунок 1 – Главная заставка приложения Asphalt Nitro

Машины разгоняются с отсутствием реалистичной физики при поворотах и прыжках. Физика – аркадная: автомобили переворачиваются и взлетают на огромную высоту.

Улучшение характеристик машин происходит благодаря игровой валюте, которую игрок получает, выиграв заезд. Она необходима для улучшений технических характеристик автомобиля. Имеются жетоны, которые нужно приобретать за реальную валюту. Жетоны используются для покупки авто и уровней.

Need for Speed Most Wanted

Главная заставка приложения Need for speed представлена на рисунке 2.

Рисунок 2 – Главная заставка приложения Need for speed

Лидер по популярности в жанре аркады (гонки). Плюсом игры является графическая составляющая и большое количество автомобилей. В связи с ограничением мобильной версии все трассы имеют достаточно простую геометрию – круговых заездов нет, только путепроводы, магистрали и т.д. Однообразные трассы, примитивная физика автомобиля и отсутствие круговых заездов – несомненный минус игры.

The Dragster

Главная заставка приложения The Dragster представлена на рисунке 3.

Рисунок 3 – Главная заставка приложения The Dragster

The Dragster, это классические драг-рейсинг гонки для Android от отечественного разработчика. Все заезды проводятся в режиме один на один, где проходит соревнование с компьютером на прямой.

Плюсом является большое количество автомобилей: 26 различных автомобилей, каждый из которых относится к своему классу, возможность изменения настроек авто. Игра распространяется на бесплатной основе, есть покупки внутриигровой валюты.

Минусом является отсутствие возможности в полной мере управлять автомобилем: автомобиль движется только вперед либо назад.

1.3 Сравнительный анализ программных продуктов

В таблице 1 приведен анализ схожих разработок по выбранным критериям.

Таблица 1 – Сравнительный анализ схожих разработок

Наименование	**Кроссплатформенность**	**Виртуальная реальность**	**Вид от первого лица**	**Распространение**
Дипломная работа	Полная (windows, linux, android, ios)	✓	✓	**Adware**
The Dragster	Отсутствует (android)	×	×	**Adware** Free-to-play
Need for Speed Most Wanted	Частичная (android, ios)	×	×	**Adware** Free-to-play
Asphalt Nitro	Частичная (android, ios)	×	×	**Adware** Free-to-play

Анализ выполнен по следующим критериям.

1. Кроссплатформенность – способность программного обеспечения работать более чем на одной аппаратной платформе и (или) операционной системе.

2. Виртуальная реальность – закрытая компьютерная симуляция некой среды вокруг пользователя.

3. Вид из салона.

4. Распространение программного обеспечения.

1.4 Обоснование разработки собственного решения

В результате анализа существующих на рынке предложений, выделены сильные и слабые стороны схожих решений. С учетом высоких темпов развития рынка и технологий виртуальной реальности принято решение о разработке симулятора виртуальной реальности в жанре «автосимулятор».

Программный продукт предлагается к распространению по бизнес-модели Adware – в приложение встраивается реклама.

Поддержка:

- виртуальная реальность;
- кроссплатформенность;
- режим камеры от первого лица;
- управление автомобилем с помощью контроллера – «джойстик».

Выводы по первому разделу

В разделе определены требования к функциональности приложения, актуальность игровых приложений. Проведен обзор аналогичных разработок, обоснование разработки собственного решения.

2. ВИРТУАЛЬНАЯ РЕАЛЬНОСТЬ

Для погружения в виртуальную реальность используются стереорежим, который предполагает обзор объекта с двух точек ракурсов, строго расположенных на горизонтальной линии на определенном расстоянии друг от друга. Таким образом, получается двухракурсное изображение, называемое стереопарой, что иллюстрирует рисунок 4.

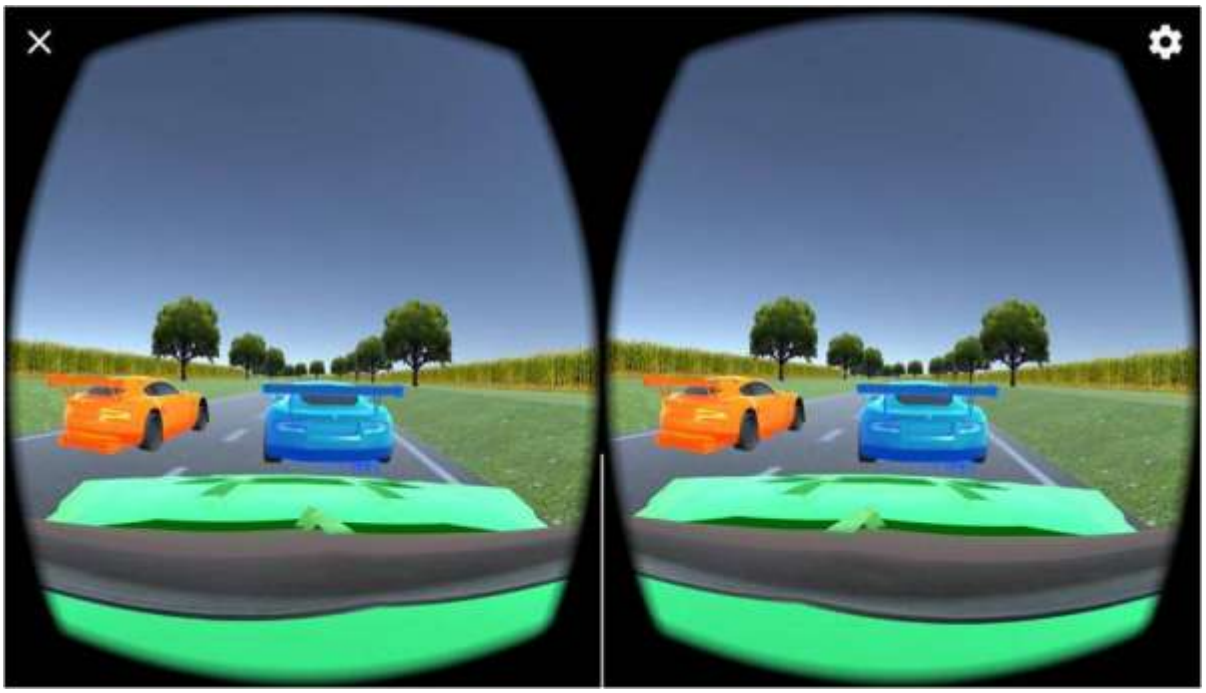

Рисунок 4 – Стереопара

Пользователь видит два ракурса правым и левым глазом, это необходимо для получения ощущения трехмерного восприятия.

Пространственное изображение, которое воспринимается в очках виртуальной реальности, иллюстрирует рисунок 5. Это результат работы мозга, оперирующего определенной разностью информации, заложенной в стереопаре, которую разглядывают глаза. Два изображения будут иметь горизонтальное смещение одноименных точек пространственного объекта.

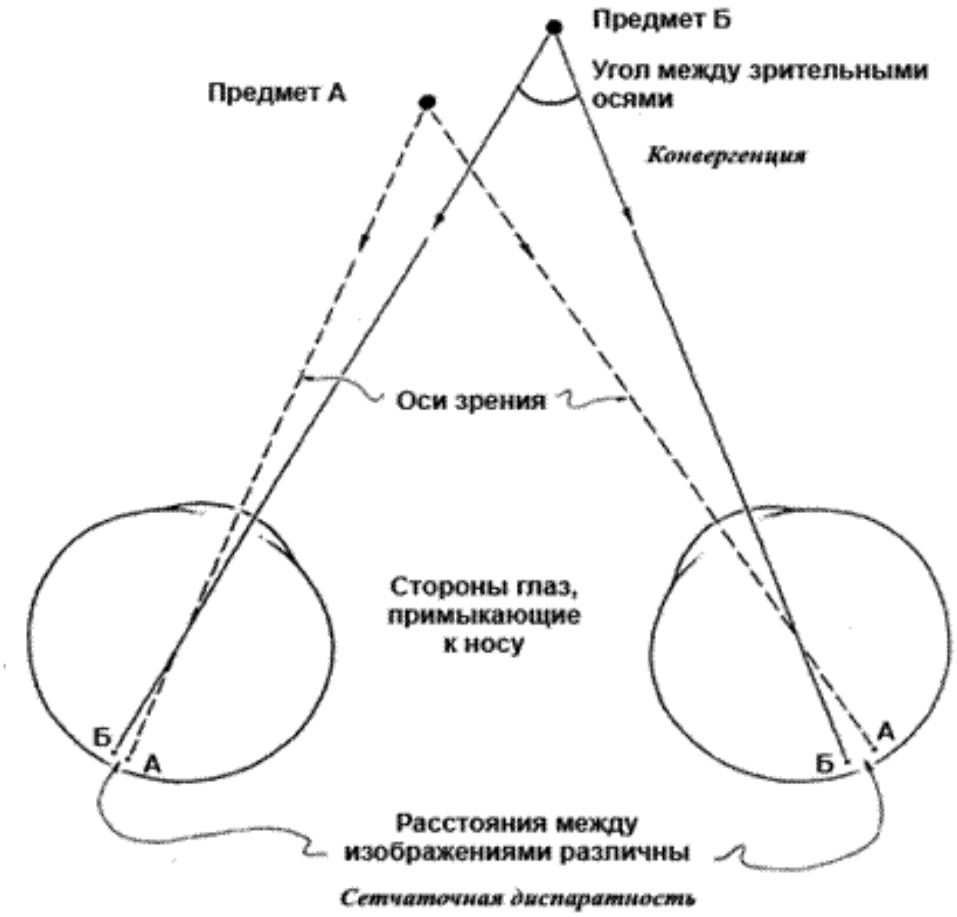

Рисунок 5 – Схема горизонтального смещения одноименных точек

Для корректного отображения картинки требуются акселерометр и гироскоп. С помощью их корректируется и перемещается 3D-изображение, в зависимости от положения головы пользователя.

Выводы по второму разделу

В разделе описана технология виртуальной реальности, каким образом она реализована.

3. ПРОЕКТИРОВАНИЕ ПРИЛОЖЕНИЯ

3.1 Построение модели

Диаграмма прецедентов разработана с помощью встроенного инструмента в Visual Studio. Диаграмма прецедентов изображена на рисунке 6.

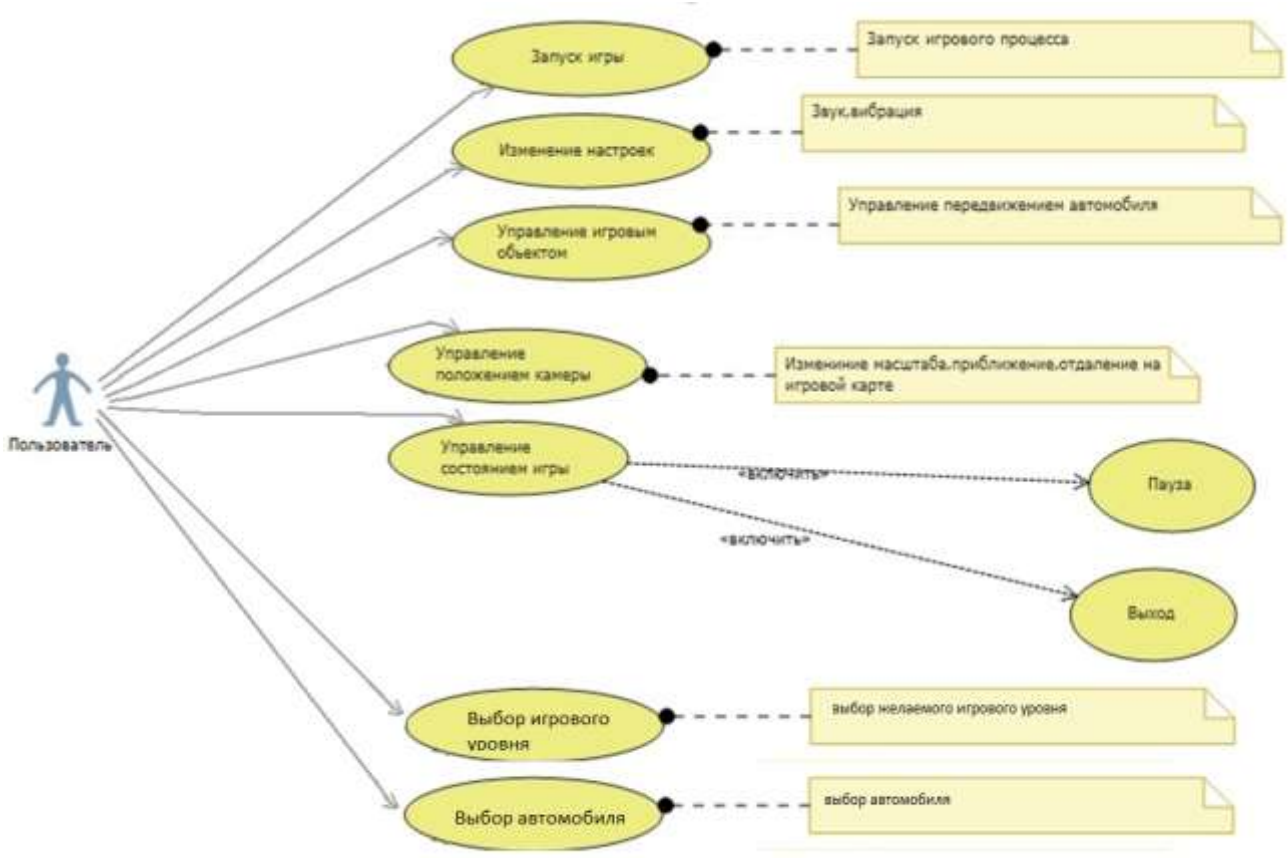

Рисунок 6 – Использование приложения пользователем

На диаграмме один актер **Пользователь**, и прецеденты, описывающие функционал моделируемой системы. Пользователь может запускать игру, выбирать игровой уровень и необходимый автомобиль, а также изменять настройки и положение обзора камеры. Прецедент «Управление состоянием игры» включает в себя возможность паузы и выхода из игрового уровня в меню основного функционала.

Диаграмма вариантов использования приложения разработчиком изображена на рисунке 7.

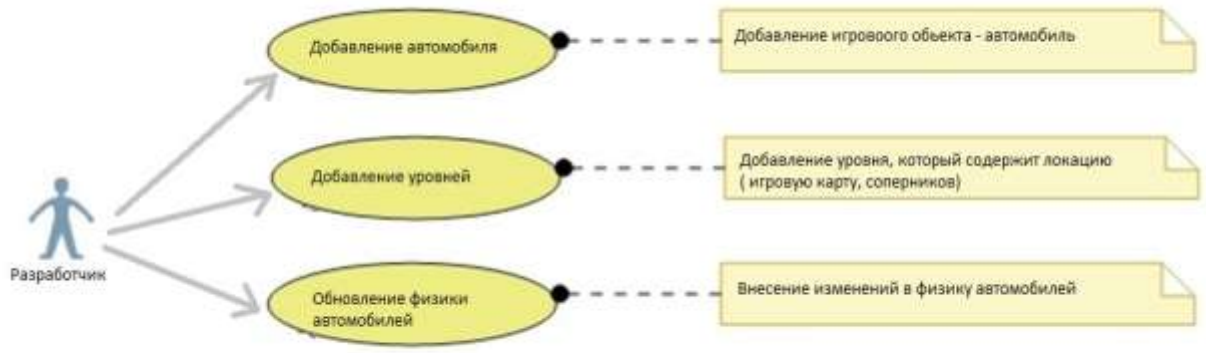

Рисунок 7 Варианты использования приложения разработчиком

Актер **Разработчик** может:

- добавлять игровые объекты типа «Автомобиль»;
- добавлять **Уровень,** который включает в себя игровую карту и соперников;
- вносить изменения в физику автомобилей, изменять их характеристики.

Диаграммы последовательностей разработаны с помощью встроенного инструмента в Visual Studio. Диаграмма последовательностей «Запуск приложения» изображена на рисунке 8.

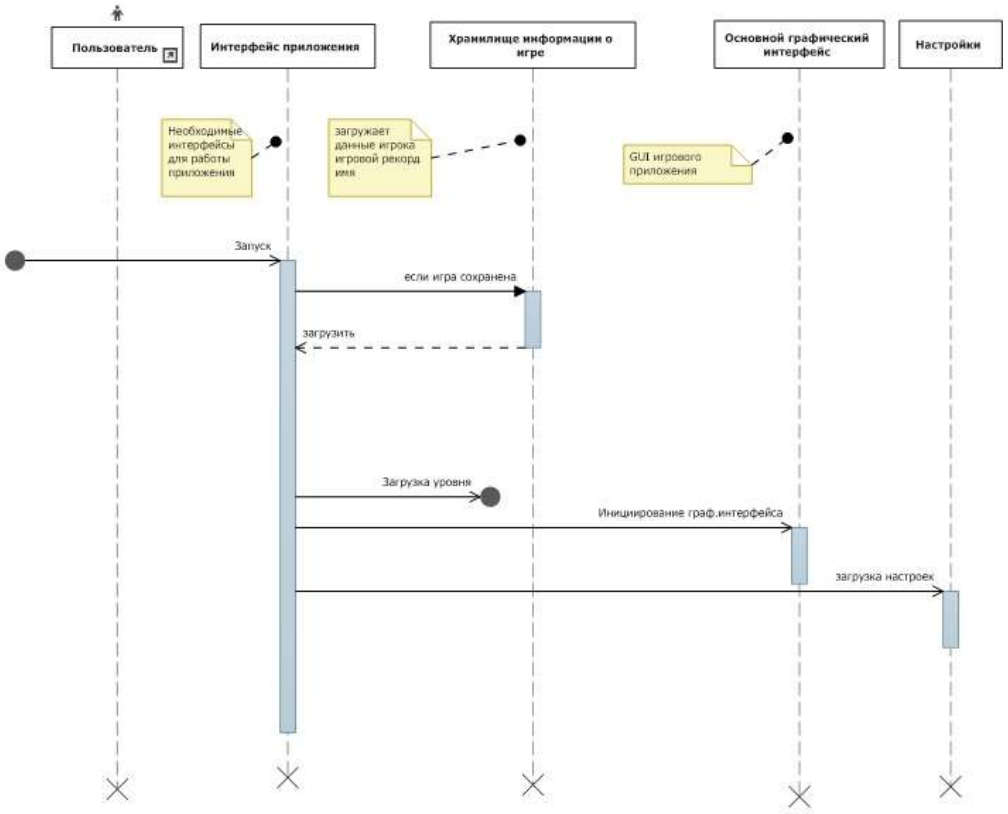

Рисунок 8 – Запуск приложения

На диаграмме отражена последовательность запуска приложения, запуска интерфейса приложения, загрузки данных пользователя, настроек, списка уровней, автомобилей и инициирование графического интерфейса.

Диаграмма последовательностей «Запуск игры» изображена на рисунке 9.

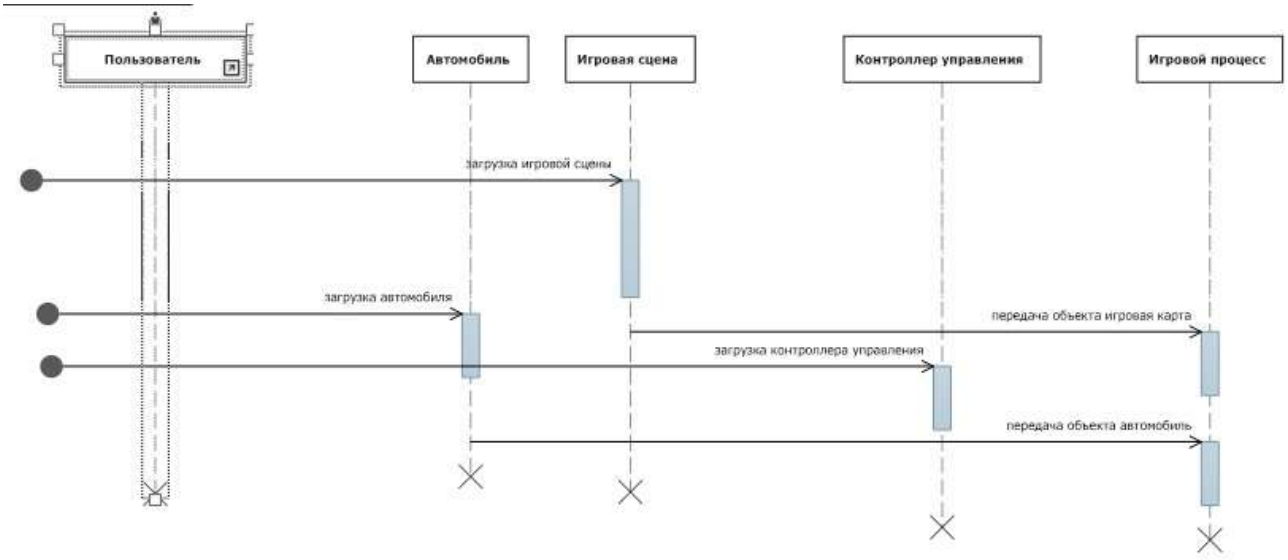

Рисунок 9 – Диаграмма запуска игры

Диаграмма последовательности запуска игры отражает: загрузка игровой сцены, загрузка автомобилей, контроллера управления автомобилем и передача этих объектов в игровой процесс.

3.2 Диаграмма классов

Диаграмма классов представлена на рисунке 10.

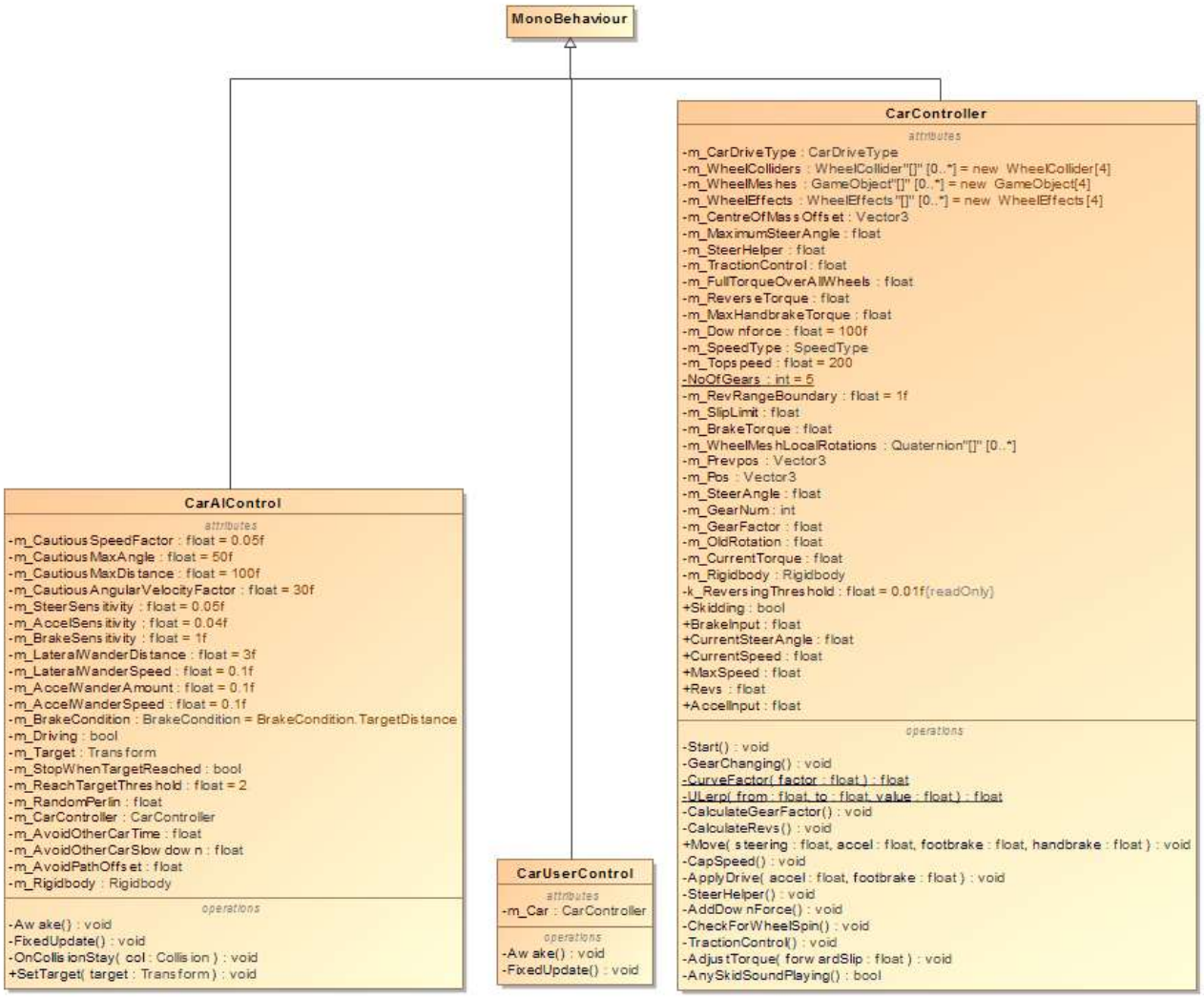

Рисунок 10 – Диаграмма классов

Описание диаграммы классов.

1. **MonoBehaviour** – это базовый класс игрового движка Unity, от которого наследуются все скрипты.

2. **CarController** содержит переменные и обращается к объектам **WheelCollider** (объект «колесо») для взаимодействия с ними.

Содержит функции: движения, коробки передач, определение текущей передачи, определение заноса и крутящего момента колес.

3. **CarUserControl** обращается к классу CarController и передает ему значения для движения, которые были получены с устройства ввода.

4. **CarAiControl** обращается к классу CarController и вызывает функции движения, в зависимости от ближайшего TargetPoint (ключевой точки для направления автомобиля)

3.3 Выбор инструмента

Инструменты, поддерживающие разработку приложений, взаимодействующих с технологиями виртуальной реальности и кроссплатформенную разработку под платформы Android, Windows:

- Unity [1];
- UnrealEngine4 [2].

Unity – самый распространенный движок для разработки приложений под мобильные платформы. Инструмент разработан, в первую очередь, для мобильных платформ и лучше оптимизирован для мобильных устройств. Системные характеристики приложений ниже, чем у UnrealEngine4. Сборка и установочный файл приложения, построенного в Unity, занимает на устройстве гораздо меньше места. Компактный размер особенно важен в магазине приложений Google Play Store, где APK приходится делить на части, если этот файл оказывается крупнее 50 мегабайт.

UnrealEngine4 – движок, целевыми платформами которого являются персональные компьютеры, в связи с этим у движка более высокие системные требования для устройств, на котором запускается готовое приложение, но он так же поддерживает разработку под мобильные платформы.

В UnrealEngine4 используется язык программирования C++. В Unity3d: C# и JavaScript. Выбор языка программирования основан на личных предпочтениях.

UnrealEngine4 и Unity3d являются условно бесплатными. Полная версия UnrealEngine4 бесплатна, пока доход от игры составляет меньше $3 000 в квартал, а если больше, то 5% от доходов.

Выбор сделан в пользу Unity, так как приложение разрабатывается в первую очередь под мобильные устройства.

Выводы по третьему разделу

В данном разделе представлено проектирование приложения, выбор инструмента для разработки игрового приложения.

4. РАЗРАБОТКА ПРИЛОЖЕНИЯ

4.1 Описание Unity как инструмента разработчика

Проект в Unity делится на сцены – отдельные файлы, содержащие свои игровые миры с определенным набором объектов, сценариев, и настроек. Сцены могут содержать объекты контейнеры (например, автомобиль), так и пустые игровые объекты – объекты, которые не имеют модели (например, источник света).

Объекты, в свою очередь, содержат наборы компонентов и скрипты, определенные разработчиком. У объектов есть название, может быть указан тег. У всех объектов на сцене присутствует компонент Transform, который хранит в себе координаты местоположения, поворота и размеров объекта по всем трём осям.

У объектов с видимой геометрией по умолчанию присутствует компонент Mesh Renderer с определенной текстурой, делающей модель объекта видимой.

Для определения формы для физических столкновений определены коллайдеры. Существует несколько типов коллайдеров.

- Character controller – вид физической модели, созданный под использование его для игровых персонажей;
- Box collider – физическая модель куб;
- Sphere collider – физическая модель сфера;
- Capsule collider – физическая модель капсула. В отличие от предыдущего типа размеры можно менять и по одной, и по трём осям сразу;
- Mesh collider – физическая модель полностью повторяет реальную геометрию объекта;
- Wheel collider – физическая модель колеса;
- Terrain collider – тип физической модели, созданный специально для использования на объекте типа Terrain – земля, генерируемая редактором Unity с возможностями построения и окрашивания местности.

На сцену можно добавлять объекты типа GameObject:

- система частиц;

- камера;
- GUI текст;
- GUI текстура;
- 3D текст;
- точечный свет;
- направленный свет;
- освещение территории;
- источник света, имитирующий солнце;
- стандартные примитивы;
- деревья;
- terrain (земля).

Компонент, отвечающий за физику Rigidbody (твёрдое тело) – это основной компонент, подключающий физическое поведение для объекта. С прикреплённым Rigidbody, объект немедленно начнёт реагировать на гравитацию. Если добавлен один или несколько компонентов Collider, то при коллизиях (столкновениях), объект будет передвигаться.

Компонент Rigidbody управляет перемещением объекта, к которому он прикреплён. Твёрдые тела управляются силами и вращением. Добавление силы/вращения к твёрдому телу позволит изменить позицию и вращение игрового объекта.

4.2 Разработка игровых объектов

В Unity используется модульная система компонентов, необходимая для конструирования игровых объектов. Игровой объект в такой системе представляет собой контейнер, содержащий необходимые для функциональности компоненты. Игровые объекты создаются путем объединения различных компонентов.

4.2.1 Игровой объект автомобиль

Игровой объект содержит следующие компоненты.

1. Коллайдеры для физических столкновений и 3D модель автомобиля (кузов и колеса).

2. Встроенный компонент Rigidbody (твердое тело) – необходимый для подключения физического поведения, содержащий параметры: масса игрового объекта, сила и центр массы.

3. Встроенный компонент Wheel Collider содержит: коллайдер для определения столкновений, масса колеса, диаметр колеса, расстояние подвески колеса, сила трения колеса, параметр силы торможения, крутящий момент движения колеса, крутящий момент торможения, и угол поворота колеса.

4. Объект Камера со стереопарой.

5. Класс CarUserControl для управления автомобилем с помощью устройств ввода.

6. Класс CarController содержит реализацию физики и звуков автомобиля. В классе определены параметры, от которых зависит поведение автомобиля:

- максимальная скорость;
- крутящий момент переднего хода авто;
- крутящий момент заднего хода авто;
- колеса и для пары передних их углы разворота;
- мощность торможения;
- количество передач.

Автомобиль приводится в движение за счет подачи крутящего момента на все колеса. Управление поворотом авто осуществляется за счет изменения углов поворота передних колес. Центр тяжести авто рассчитывается в зависимости от массы физических компонентов авто.

Функция движения автомобиля Move() принимает значения :

- steering – значение поворота колеса;
- accel – значение педали газа;
- Footbrake – значение педали тормоза;
- Handbrake – значение ручного тормоза.

Код функции движения приведен в **Ошибка! Источник ссылки не найден.**.

Листинг 1 – Код функции движения

```
public void Move(float steering, float accel, float footbrake, float handbrake)
        {
            for (int i = 0; i < 4; i++)
            { //позиции текстур(3д моделей колес) в зависимости от позиций
              //физических моделей колес
                Quaternion quat;
                Vector3 position;
                m_WheelColliders[i].GetWorldPose(out position, out quat);
                m_WheelMeshes[i].transform.position = position;
                m_WheelMeshes[i].transform.rotation = quat;
            }
            // clamp= диапазоны если -1 меньше steering то возвращаем -1, иначе 1
            //если этого не сделать, то колеса не смогут разворачиваться обратно,
            //тормозить, если будет разгоняться авто
            // например колесо будет всегда направо, или налево
            steering = Mathf.Clamp(steering, -1, 1);
            AccelInput = accel = Mathf.Clamp(accel, 0, 1);
            BrakeInput = footbrake = -1 * Mathf.Clamp(footbrake, -1, 0);
            handbrake = Mathf.Clamp(handbrake, 0, 1);
            // Установили рулевое управление на передние колеса.
            // Предполагая, что колеса 0 и 1 являются передними колесами.
            // задали угол поворота колес
            m_SteerAngle = steering * m_MaximumSteerAngle;
            m_WheelColliders[0].steerAngle = m_SteerAngle;
            m_WheelColliders[1].steerAngle = m_SteerAngle;
            //задаем крутящий момент колесам
            //в зависимости от параметра педали газа
            thrustTorque = accel * (m_CurrentTorque / 4f);
            for (int i = 0; i < 4; i++)
            {   //задаем крутящий момент каждому колесу
                m_WheelColliders[i].motorTorque = thrustTorque;
            }
            for (int i = 0; i < 4; i++)
            {
          if (CurrentSpeed>5 && Vec-
tor3.Angle(transform.forward,m_Rigidbody.velocity)< 50f)
                {   //задаем торможение если необходимо
                    m_WheelColliders[i].brakeTorque = m_BrakeTorque * footbrake;
                }
                //ручной тормоз на задние колеса если необходимо
                if (handbrake > 0f)
                {
                    var hbTorque = handbrake * m_MaxHandbrakeTorque;
                    m_WheelColliders[2].brakeTorque = hbTorque;
                    m_WheelColliders[3].brakeTorque = hbTorque;
                }

                CalculateRevs();//звук двигателя
```

```
            GearChanging();//смена передачи
            CheckForWheelSpin();//эффекты для колес
            TractionControl();//корректировка крутящего момента
        }
    }
```

Функция смены передачи автомобиля, которая в зависимости от текущей скорости, повышает или понижает передачу автомобиля.

Код функции переключения коробки передач приведен в листинге 2.

Листинг 2 – Код функции переключения коробки передач

```
private void GearChanging()
    {   //f= текущая скорость/максимальная
        float f = Mathf.Abs(CurrentSpeed / MaxSpeed);
        float upgearlimit = (1 / (float)CountGears) * (m_GearNum + 1);
        float downgearlimit = (1 / (float)CountGears) * m_GearNum;
        //в зависимости от текущей скорости понижаем или повышаем передачу
        //и коэф. скоростей которые зависят от количества передач
        if (m_GearNum > 0 && f < downgearlimit)
        {
            m_GearNum--;
        }

        if (f > upgearlimit && (m_GearNum < (CountGears - 1)))
        {
            m_GearNum++;
        }
    }
```

Настройка InputManager для корректного получения значений с клавиатуры и беспроводного джойстика отражена на рисунке 11.

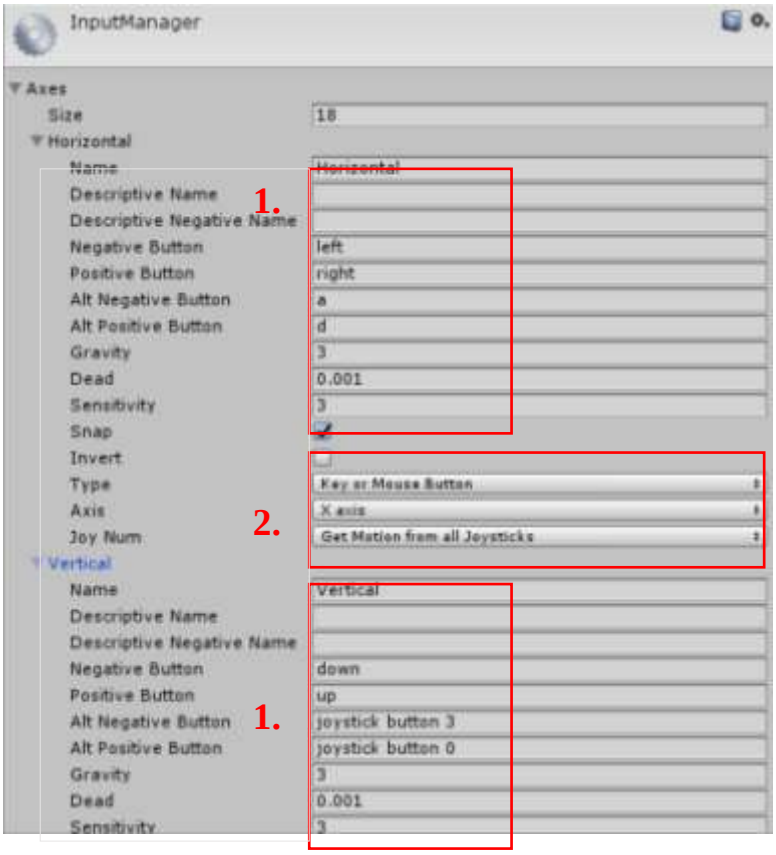

Рисунок 11 – Настройка InputManager

Под номером 1 указываются наименования клавиш на устройстве ввода. Для поворота автомобиля налево и направо выбрано наименование ввода Horizontal и указаны:

- клавиши клавиатуры a, d;
- для джойстика left right.

Под номером 2 выбраны настройки под тип устройства ввода – клавиши, мышь и все кнопки джойстика по оси X.

Для движения вперед выбрано наименование ввода – Vertical с указанием наименований клавиш клавиатуры и джойстика.

Класс CarUserControl, содержит функционал для управления автомобилем с устройств ввода. Включает в себя объект класса CarController, и обращается к его функции Move(), передавая ей значения с устройства ввода.

Код функции управлении с устройств ввода приведен в листинге 3.

Листинг 3 – Код функции управления игровым объектом с устройств ввода

```
{
            m_Car     =     GetComponent<CarController>();//получаем     компонент
"CarController" игрового объекта
        }
        private void FixedUpdate()  //Каждый кадр
        {
            //Получаем значения с устройства ввода
            float h = CrossPlatformInputManager.GetAxis("Horizontal");
            float v = CrossPlatformInputManager.GetAxis("Vertical");
            float handbrake = CrossPlatformInputManager.GetAxis("Jump");
            //Передаем значения компоненту CarController
            m_Car.Move(h, v, v, handbrake);
         }
```

4.2.2 Игровой объект «компьютерный соперник»

Содержит компоненты игрового объекта «автомобиль»:

- встроенный компонент WheelCollider;
- класс CarController;
- 3D модель автомобиля, текстуры 3D модели, коллайдеры для физических столкновений;
- компонент CarComputer для реализации функционала компьютерного соперника.

В классе CarComputer определены параметры чувствительности педали газа, тормоза, рулевого управления и массив точек, по которым передвигается компьютерный соперник. Определены методы вычисления расстояния автомобиля и определения угла до ключевой точки. Класс содержит объект класса CarController и вызывает функцию Move(), передавая необходимые параметры.

Часть кода реализации компьютерного игрока приведена в листинге 4.

Листинг 4 – Часть кода скрипта для движения автомобиля по точкам

```
private void FixedUpdate()
{
if (m_Target == null || !m_Driving)
{
      // точки нет и не нужно двигаться – стоим
      m_CarController.Move(0, 0, -1f, 1f);
}
else
{
      Vector3 fwd = transform.forward;
      if (m_Rigidbody.velocity.magnitude > m_CarController.MaxSpeed * 0.1f)
```

```
        {
            fwd = m_Rigidbody.velocity;
        }
        //необходимая скорость которой придерживаемся
        float desiredSpeed = m_CarController.MaxSpeed;
        // необходимо ли останавливаться
        switch (m_Brake)
        {
        case ReqBrake.BrakingTurn:
            {
            // автомобиль будет тормозить в соответствии с предстоящим изменением направления цели.
            // проверяем угол нашей цели по сравнению с текущим направлением автомобиля
            float CorAngle = Vector3.Angle(m_Target.forward, fwd);
            // если необходимо уменьшить угол направления
            float cRequired = Mathf.InverseLerp(0, m_CMaxAngle,CorAngle);
            desiredSpeed = Mathf.Lerp(m_CarController.MaxSpeed, m_CarController.MaxSpeed * m_CoefCareSpeed, cRequired);
            break;
            }
        case ReqBrake.PointDistance:
        {// автомобиль будет тормозить, когда он приближается к своей цели, независимо от направления цели.
            // проверяем расстояние до цели
            Vector3 delta = m_Target.position - transform.position;
            float distanceCautiousFactor = Mathf.InverseLerp(100f, 0, delta.magnitude);
            // если необходимо уменьшить угол направления
            float cRequired = Mathf.InverseLerp(0, m_CMaxAngle,CorAngle);
            desiredSpeed = Mathf.Lerp(m_CarController.MaxSpeed, m_CarController.MaxSpeed * m_CoefCareSpeed,
                                                        cRequired);
            break;
        }
        case ReqBrake.FullForward:
            break;
        }
        // целевая позиция направления
        Vector3 offsetTargetPos = m_Target.position;
        // использовать различную чувствительность в зависимости от того, ускорение или торможение:
        float acclBrakeSens = (desiredSpeed < m_CarController.CurrentSpeed)
                                                            ? m_BrakeSens
                                                            : m_AccelerationSens;
// принимаем фактическое ускорение / тормоза для достижения желаемой скорости.
        float accel = Mathf.Clamp((desiredSpeed - m_CarController.CurrentSpeed) * acclBrakeSens, -1, 1);
        // вычисляем локально-относительное положение цели, чтобы
        Vector3 localTarget = transform.InverseTransformPoint(offsetTargetPos);
        // выработать локальный угол к цели
        float targetAngle = Mathf.Atan2(localTarget.x, localTarget.z) * Mathf.Rad2Deg;
// получить сумму рулевого управления, необходимую для движения автомобиля к цели
```

```
        float steer = Mathf.Clamp(targetAngle * m_SteerSensitivity, -1, 1) *
Mathf.Sign(m_CarController.CurrentSpeed);
        // подавать значения на контроллер автомобиля.
        m_CarController.Move(steer, accel, accel, 0f);
        // если необходимо, остановить движение, когда мы будем достаточно близко к
цели.
        if (m_StopWhenTargetReached && localTarget.magnitude <
m_ReachTargetThreshold)
        {
            m_Driving = false;
        }
    }
}
```

4.2.3 Игровой объект « карта»

Объект игровая карта содержит:

- 3D модель карты;
- текстуры 3D модели;
- коллайдеры для физических столкновений;
- точки направлений для авто.

На рисунке 12 изображен состав компонентов объекта «игровая карта».

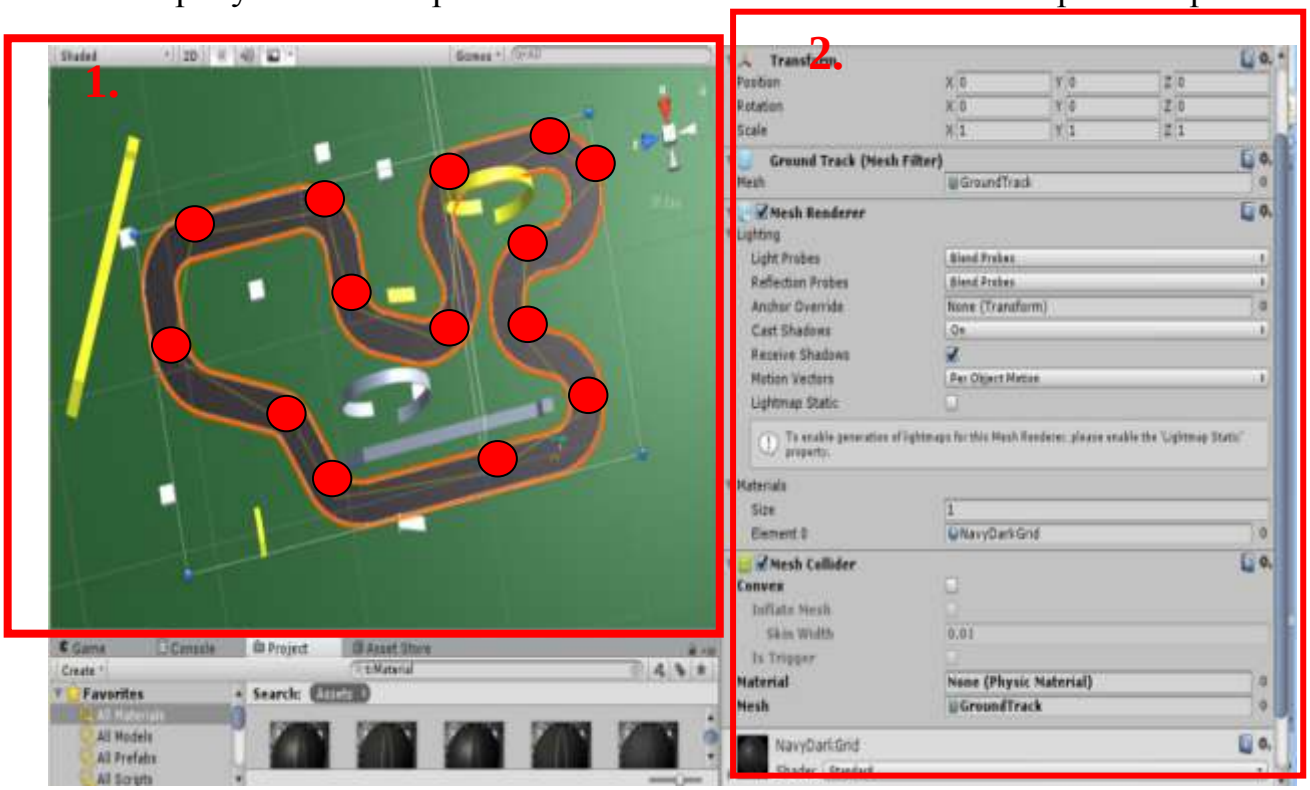

Рисунок 12 – Объекты игровой карты

В области под номером 1 указан готовый контейнер игровая карта, с точками на определенных координатах для передвижения компьютерных соперников. В области под номером 2 указан состав компонентов трассы :

- Transform – содержит координаты объекта.
- Mesh Renderer – для визуализации объекта, содержит 3D модель карты.
- Mesh Collider – физическая модель карты, при добавлении этого компонента Unity задействует физику игрового движка для данного объекта.

4.2.4 Игровой объект «GameProcess»

Игровой объект GameProcess содержит класс LapSystem, который обеспечивает реализацию игрового процесса: старт, прохождение ключевых точек, финиш.

Код класса LapSystem приведен в листинге 5.

Листинг 5 – Код класса LapSystem

```
public class LapSystem : MonoBehaviour {

	public int no_checks;
	public int curr_check;
	public int no_laps;
	public int curr_lap;
	public GameObject go_ui;

	void Start () {
		no_checks = GameObject.Find ("Checkpoints").transform.childCount;
		curr_check = 1;// текущий чекпоинт
		no_laps = 3;//количество кругов
		curr_lap = 1;//текущий КРУГ
	}
	void Update () {
		if (curr_check > no_checks) {
			curr_lap++;
			curr_check = 1;
		}
		if (curr_lap > no_laps) {

			go_ui.SetActive (true);// объект сцена с результатами
		}
	}
	void OnTriggerEnter (Collider check_col)
	{
		if (check_col.name == curr_check.ToString ())
		{
```

```
                curr_check++;
            }
        }
}
```

4.2.5 Игровой уровень

Игровой уровень – сцена, заключительный контейнер, который содержит все необходимые игровые объекты.

Состав компонентов изображен на рисунке 13.

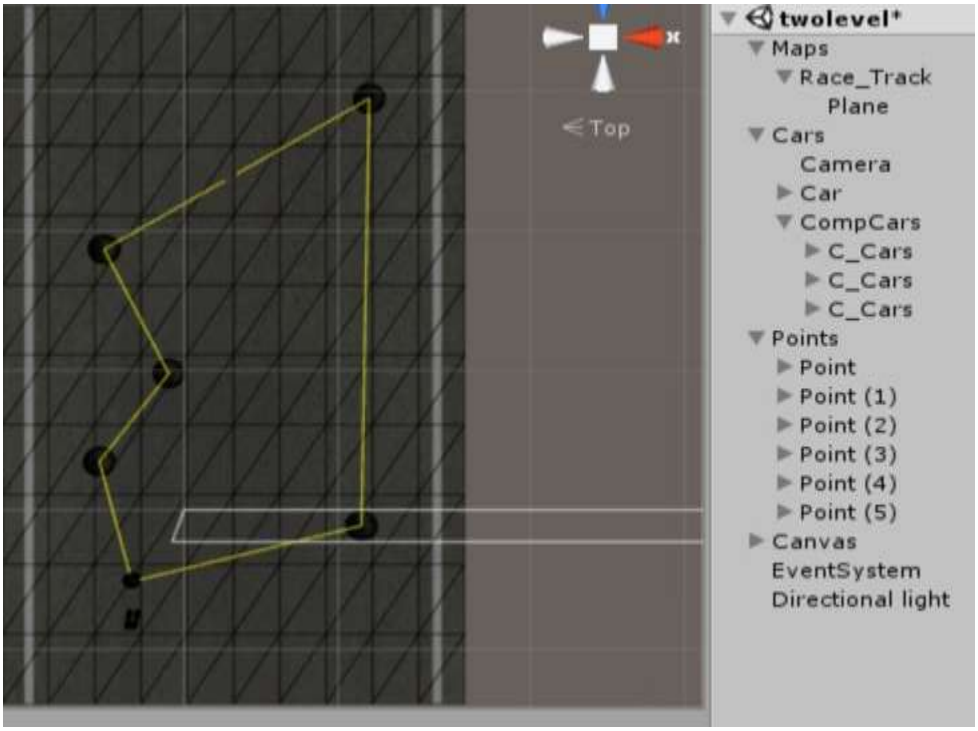

Рисунок 13 – Сцена с игровыми объектами

Сцена содержит:

- Maps – объект игровая карта;
- Cars – игровой автомобиль, камеру, которая необходима для отображения игрового процесса, CompCars – содержит 3 объекта типа «компьютерный соперник»;
- Points – содержит точки, с координатами для направлений компьютерных соперников;
- Canvas – игровой объект, содержащий GUI – кнопки, label;

- EventSystem – система событий, способ отправки событий к объектам в приложении, основанный на вводе с клавиатуры;
- Directional light – источник света на сцене.

4.3 Sketchup

SketchUp – программа для моделирования относительно простых трёхмерных объектов – строений, мебели, интерьера. Основной идеей SketchUp является простота интерфейса. Программа реализует концепцию прямого моделирования геометрии, в рамках которой пользователь сначала стоит плоский контур из имеющихся примитивов, затем вытягивает его с целью создания или вычитания объема, после чего придает модели нужную форму посредством перетаскивания ее элементов (вершин, ребер и граней) с помощью указателя мыши.

SketchUp поддерживает экспорт и импорт различных форматов трехмерной и растровой графики. В программе имеются библиотеки компонентов, которые можно пополнять своими элементами, и библиотека материалов. Поставляется в двух вариантах – бесплатном и коммерческом.

Построение круговой трассы изображено на рисунке 14.

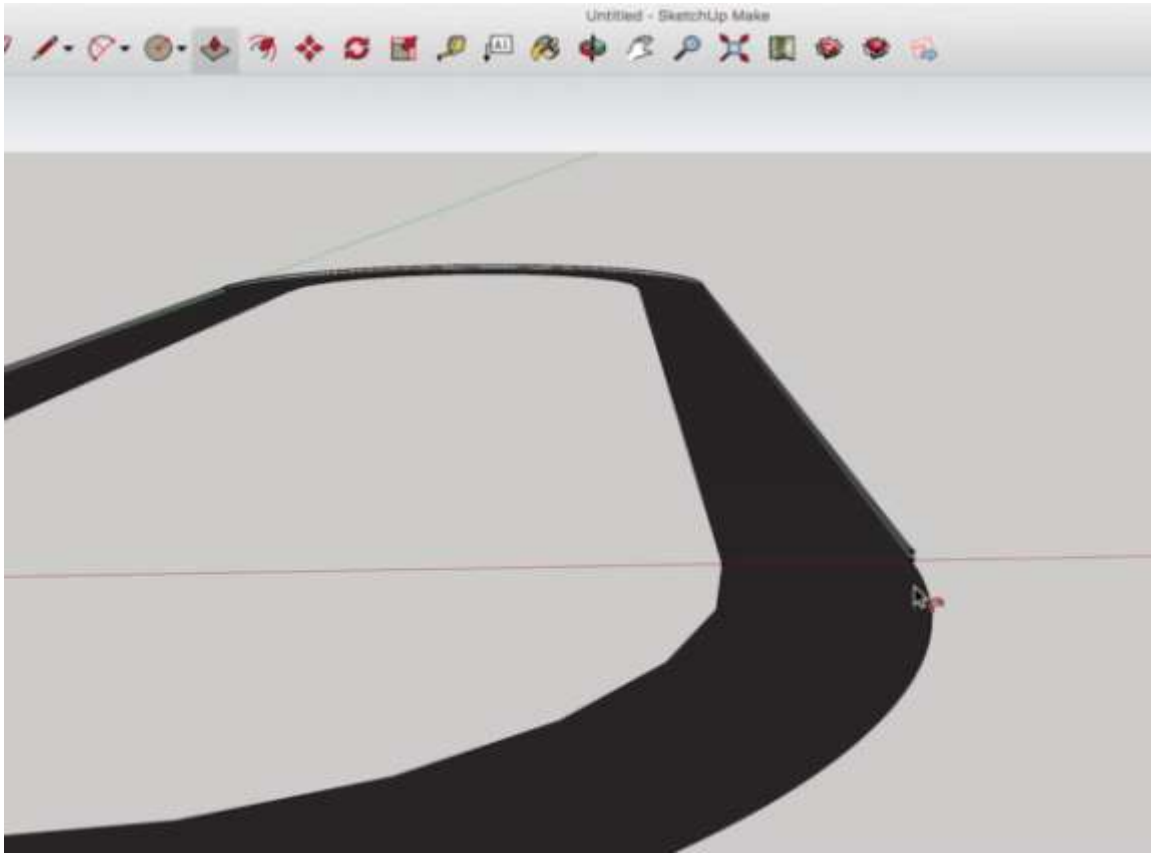

Рисунок 14 – Создание круговой трассы

После сохранения 3D модели карты, ее необходимо импортировать в Unity с помощью вкладки Assets Unity > Import New Assets, которая поддерживает импорт моделей из всех популярных 3D редакторов.

4.4 Autodesk 3ds Max

Autodesk 3ds Max (ранее 3D Studio MAX) – полнофункциональная профессиональная программная система для создания и редактирования трёхмерной графики и анимации, доразработанная компанией Autodesk.

Autodesk 3ds Max доступен в двух лицензионных версиях: студенческая – бесплатная (требуется регистрация на сайте Autodesk), и полная (коммерческая) версия.

3ds Max располагает обширными средствами для создания разнообразных по форме и сложности трёхмерных компьютерных моделей, реальных или фанта-

стических объектов окружающего мира, с использованием разнообразных техник и механизмов.

Autodesk 3ds Max использовалась как для создания, так и для редактирования моделей. На рисунке 15 изображена готовая модель автомобиля.

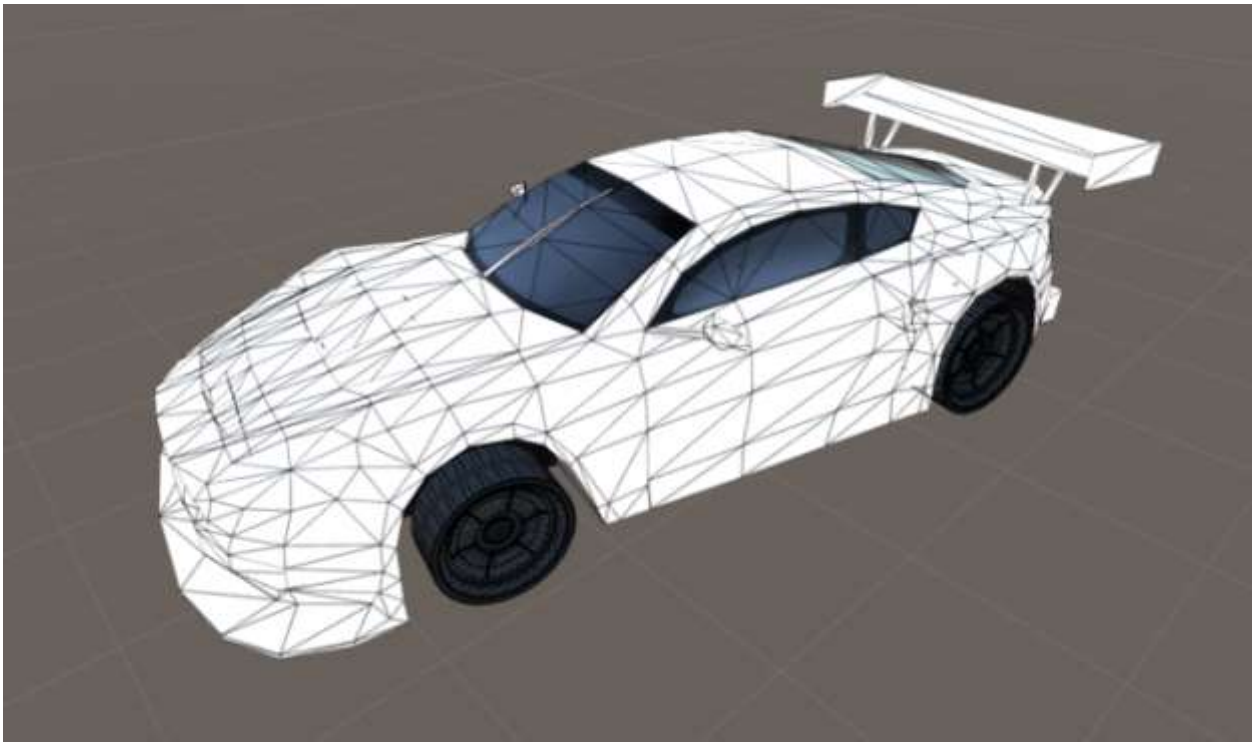

Рисунок 15 – Модель авто

После сохранения 3D модели автомобиля, ее необходимо импортировать в Unity с помощью вкладки Assets Unity > Import New Assets, которая поддерживает импорт моделей из всех популярных 3D редакторов.

Выводы по четвертому разделу

В данном разделе описаны инструменты для разработки игровых приложений и создания 3D моделей, описан состав игровых компонентов и игровой сцены.

5. ЭКОНОМИЧЕСКИЙ РАЗДЕЛ

Основная заработная плата программиста за время разработки, постановки и написания самого проекта вычисляется по формуле (1)

$$\text{ОЗП}=\frac{\text{ЗП}_{\text{мес.}}}{\text{ФРВ}} \times t_{\text{разработки,}} \quad (1)$$

где ЗПмес. = средняя заработная плата программиста;

ФРВ – месячный фонд рабочего времени;

t разработки – время затраченное на разработку программы.

ЗПмес. = 22000 рублей;

ФРВ = 168 часов;

t разработки = 2 месяца = 336 часов.

Таким образом,

$$\text{ОЗП}=\frac{22000}{168} \times 336 = 44000 \text{ рублей}$$

Отчисления на социальные нужды по единому социальному налогу (ЕСН) составляют 26% от заработной платы.

ОТ = 44000 * 0,26 = 11440 рублей.

Расчет затрат на материалы, необходимые при разработке проекта, производится в таблице 2. (Цены указаны на июнь 2017 г.)

Таблица 2 – Затраты на 3D модели

Наименование	**Количество**	**Ед. изм.**	**Цена**
Стоимость модели автомобиля	3	шт.	~600 р.
Стоимость игровой трассы	3	шт.	~600 р.
Итого			1200 р

Общая сумма расходов на материалы (МЗ) в составит около 1200,00 рублей.

Сумма прочих расходов рассчитывается как 3% от основных расходов:

ПР = (ОЗП + ОТ + МЗ) * 0,03 = (44000 + 11400 + 1200) * 0,03= 1698 руб.

Статьи проектных затрат представлены в сводной таблице 2.

Таблица 2 – Статьи проектных затрат

№ п/п	Статья расхода	Сумма
1.	Проектные затраты	
2.	Основная зарплата	44000
3.	Материальные затраты	1200
4.	Прочие расходы	1698
Итого		46898

Итого, общие проектные затраты (ПЗ) составляют 46 898 рублей.

Выводы по пятому разделу

Рассчитана стоимость создания приложения, включающая в себя затраты на разработку и покупку 3D моделей.

ЗАКЛЮЧЕНИЕ

Разработано приложение для мобильных устройств – симулятор виртуальной реальности. Проведен обзор аналогичных разработок. Разработаны диаграммы прецедентов и последовательностей, разработана архитектура приложения. Создан основной функционал приложения, разработаны игровые объекты, игровые уровни, модели игровых объектов.

БИБЛИОГРАФИЧЕСКИЙ СПИСОК

1 Официальный сайт инструмента Unity: https://unity3d.com/ru

2 Официальный сайт инструмента Unreal Engine4: https://www.unrealengine.com/

3 Руководство Unity: https://docs.unity3d.com/ru

4 Д.Хокинг Unity в действии. Мультиплатформенная разработка на C#, 2015

5 Google VR: https://vr.google.com/cardboard/

6 Рынок игр: https//json.tv/ict_telecom_analytics_view/analiz-rynka-igr-v-rossii-i-mire-2014-2016-gg-tekuschaya-situatsiya-prognozy-igroki-proekty-i-tendentsii-20150724054917.

7 Рынок игр в России и в мире: http://json.tv/ict_telecom_analytics_view/issledovanie-mirovogo-i-rossiyskogo-rynka-igr-2016-god-20170502014806.

8 Анализ рынка виртуальной реальности 2016:https://habrahabr.ru/post/318868/