Министерство образования Республики Беларусь

Учреждение образования «Белорусский государственный университет  
информатики и радиоэлектроники»

Факультет компьютерных систем и сетей

Кафедра информатики

Дисциплина: Операционные системы и сети

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

к курсовому проекту на тему

«3D игра на Unity»

Выполнил:   
студент группы 753503 Шелдукова А.А.

Руководитель:

ассистент кафедры информатики

Шнейдер В. В.

Минск, 2020

# Содержание

[Введение 2](#_Toc40460090)

[1. Краткие теоретические сведения 3](#_Toc40460091)

[1.1 Обзор игрового движка Unity 3D 3](#_Toc40460092)

[1.2 Возможности Unity 4](#_Toc40460093)

[1.3 Достоинства и недостатки Unity 6](#_Toc40460094)

[2. Процесс разработки игры 8](#_Toc40460095)

[2.1 Описание игры «Racing» 8](#_Toc40460096)

[2.2 Описание графического интерфейса 9](#_Toc40460097)

[3. Демонстрация работы игры 11](#_Toc40460098)

[Заключение 13](#_Toc40460099)

[Список использованных источников 14](#_Toc40460100)

[Приложение 1. Код программы 15](#_Toc40460101)

# Введение

Компьютерные игры появились в середине 70-х годов и за несколько десятилетий набрали огромную популярность. Разработкой игр занимаются как крупные компании, так и небольшие фирмы, а также независимые частные сообщества.

В связи с быстрым ростом технологий у разработчиков появляется все больше возможностей для творчества. Первые игры представляли собой простейшие двухцветные 2D проекты, в настоящее время игровая индустрия достигла огромных высот. Современные 3D игры обладают настолько высоким уровнем реалистичности, что человек, играя в них, забывает, что находится в виртуальном мире. Так же в настоящее время активно ведется разработка игр для шлемов виртуальной реальности, которые позволяют пользователю с головой окунуться в игру.

Ни одна игра не обходится без игрового движка, который является основополагающим элементом для реализации проектов. Его основная задача – обеспечить базовыми технологиями, а также упростить работу над проектом для разработчика; игровой движок дает возможность кроссплатформенности (запуск и разработка игр на нескольких платформах одновременно). Обычно игровой движок включает в себя движок рендеринга, физический движок, звук, систему скриптов, анимацию, искусственный интеллект, сетевой код, управление памятью, многопоточность. Unity – это мультиплатформенный игровой движок для разработки двух- и трехмерных игр, работающих под операционными системами Windows OS X, Windows Phone, Android, Apple IOS, Linux, а так же на игровых приставках Wii, PS 3, PS 4, Xbox 360, Xbox One. Есть возможность создания браузерных приложений с помощью модулей Unity (Unity Web Player). Unity поддерживает 2 языка программирования JavaScript, C#. Для физических расчетов используется встроенный движок PhysX, разработанный компанией NVIDIA.

Целью работы является разработка игры на движке Unity 3D.

Тема является актуальной в связи с высоким ростом игровой индустрии в наше время. На рынке компьютерных технологий существуют как крупные компании (EA, Valve, Ubisoft), так и отдельные разработчики. В большинстве случаев крупные компании используют собственные движки для разработки проектов, а разработчики же предпочитают готовые, среди которых есть и бесплатные (Unity, Unreal Engine и т.д.)

В связи с поставленной целью в работе решаются следующие задачи:

* изучить интерфейс платформы Unity3D,
* описать сценарии видеоигры на языке программирования C#,
* провести окончательную сборку проекта.

# 1. Краткие теоретические сведения

## 1.1 Обзор игрового движка Unity 3D

В настоящее время существует большое количество инструментов для разработчика игр, для исследования был выбран один из самых новых и мощных движков – Unity 3D. У данного инструмента существуют как преимущества, так и недостатки.

Unity — межплатформенная среда разработки компьютерных игр. Unity позволяет создавать приложения, работающие под более чем 20 различными операционными системами, включающими персональные компьютеры, игровые консоли, мобильные устройства, интернет-приложения и другие. Выпуск Unity состоялся в 2005 году и с того времени идёт постоянное развитие.

Что мы можем в Unity: моделировать физическую среду, моделировать карты нормалей, моделировать преграждение окружающего света в экранном пространстве (SSAO), моделировать динамические тени и т. д.

Данный функционал существует во многих игровых движках, преимущества Unity в том, что в нем есть чрезвычайно производительный визуальный процесс и мощная межплатформенная поддержка.

Визуальный редактор Unity значительно упрощает взаимодействие объектов на сцене и процесс сборки проекта. Благодаря Unity разработчики избавляют себя от такой проблемы, как ограниченная и недостаточно гибкая поддержка возможности написания сценариев. Особенно полезен этот редактор при разработке проектов с последовательным улучшением, например, в циклах создания прототипов или тестирования.

Также обзор игрового движка не обойдется без описания окна и интерфейса. Компонентами интерфейса Unity являются:

* вкладка Scene
* вкладка Game
* вкладка Hierarchy
* панель Inspector
* вкладки Project и Console

Верхнюю часть занимает панель инструментов, слева кнопки для осмотра и перемещения объектов, а в центре располагается кнопка Play. С правой стороны панель, отображающая информацию о выделенном в данный момент объекте.

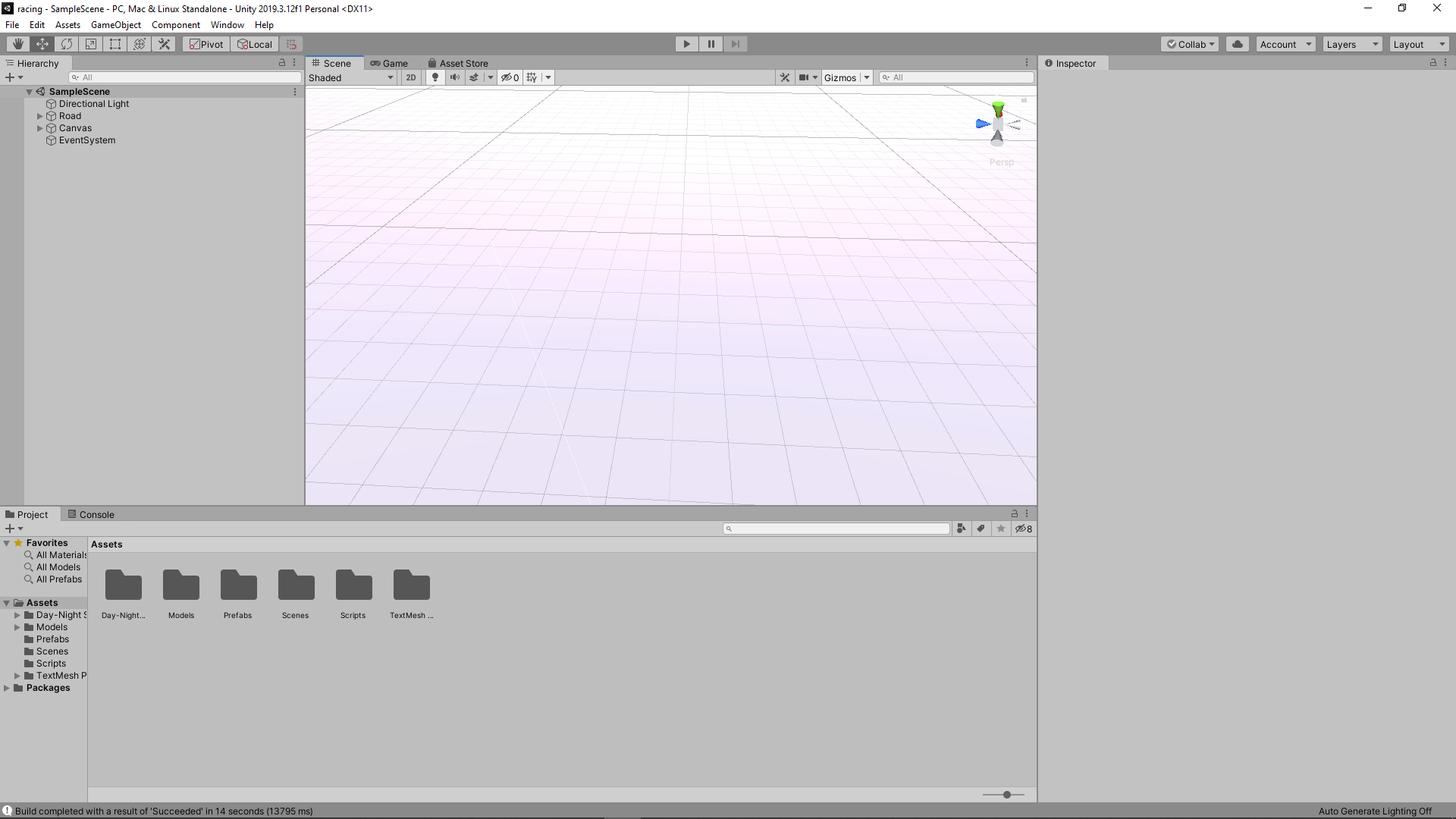


Рисунок 1.1.1 – Окно интерфейса Unity

На начальном этапе работы с движком необходимо выбрать язык, с помощью которого будет проходить разработка игры. В Unity поддерживаются всего 2 языка программирования JavaScript и C#. Язык C# имеет больше преимуществ чем недостатков перед языком JavaScript, особенно с точки зрения профессионального разработчика.

На Unity написаны тысячи игр, приложений, визуализации математических моделей, которые охватывают множество платформ и жанров. При этом Unity используется как крупными разработчиками, так и [независимыми студиями](https://ru.wikipedia.org/wiki/%D0%98%D0%BD%D0%B4%D0%B8-%D0%B8%D0%B3%D1%80%D0%B0).

### 1.2 Возможности Unity

Редактор Unity имеет простой [Drag&Drop](https://ru.wikipedia.org/wiki/Drag-and-Drop" \o "Drag-and-Drop) интерфейс, который легко настраивать, состоящий из различных окон, благодаря чему можно производить отладку игры прямо в [редакторе](https://ru.wikipedia.org/wiki/%D0%A0%D0%B5%D0%B4%D0%B0%D0%BA%D1%82%D0%BE%D1%80_%D1%83%D1%80%D0%BE%D0%B2%D0%BD%D0%B5%D0%B9). Движок использует для написания скриптов [C#](https://ru.wikipedia.org/wiki/C_Sharp). Ранее поддерживались также [Boo](https://ru.wikipedia.org/wiki/Boo" \o "Boo) (диалект [Python](https://ru.wikipedia.org/wiki/Python" \o "Python), поддержку убрали в 5-й версии) и модификация [JavaScript](https://ru.wikipedia.org/wiki/JavaScript" \o "JavaScript), известная как UnityScript (поддержка прекращена в версии 2017.1). Расчёты физики производит физический движок [PhysX](https://ru.wikipedia.org/wiki/PhysX" \o "PhysX) от [NVIDIA](https://ru.wikipedia.org/wiki/NVIDIA). Графический API - [DirectX](https://ru.wikipedia.org/wiki/DirectX" \o "DirectX) (на данный момент [DX 11](https://ru.wikipedia.org/wiki/DX11), поддерживается [DX 12](https://ru.wikipedia.org/wiki/Direct3D_12))

Проект в Unity делится на сцены ([уровни](https://ru.wikipedia.org/w/index.php?title=%D0%98%D0%B3%D1%80%D0%BE%D0%B2%D1%8B%D0%B5_%D1%83%D1%80%D0%BE%D0%B2%D0%BD%D0%B8&action=edit&redlink=1)) — отдельные файлы, содержащие свои игровые миры со своим набором объектов, сценариев, и настроек. Сцены могут содержать в себе как, собственно, объекты (модели), так и пустые игровые объекты — объекты, которые не имеют модели («пустышки»). Объекты, в свою очередь содержат наборы компонентов, с которыми и взаимодействуют скрипты. Также у объектов есть название (в Unity допускается наличие двух и более объектов с одинаковыми названиями), может быть тег (метка) и слой, на котором он должен отображаться. Так, у любого объекта на сцене обязательно присутствует компонент [Transform](https://ru.wikipedia.org/wiki/Transform" \o "Transform) — он хранит в себе координаты местоположения, поворота и размеров объекта по всем трём осям. У объектов с видимой геометрией также по умолчанию присутствует компонент [Mesh Renderer](https://ru.wikipedia.org/w/index.php?title=Mesh_Renderer&action=edit&redlink=1" \o "Mesh Renderer (страница отсутствует)), делающий модель объекта видимой.

К объектам можно применять коллизии (в Unity т. н. коллайдеры — collider), которых существует несколько типов.

Также Unity поддерживает физику твёрдых тел и ткани, а также физику типа [Ragdoll](https://ru.wikipedia.org/wiki/Ragdoll-%D1%84%D0%B8%D0%B7%D0%B8%D0%BA%D0%B0" \o "Ragdoll-физика) (тряпичная кукла). В редакторе имеется система наследования объектов; дочерние объекты будут повторять все изменения позиции, поворота и масштаба родительского объекта. Скрипты в редакторе прикрепляются к объектам в виде отдельных компонентов.

При импорте текстуры в Unity можно сгенерировать alpha-канал, mip-уровни, normal-map, light-map, карту отражений, однако непосредственно на модель текстуру прикрепить нельзя — будет создан материал, которому будет назначен [шейдер](https://ru.wikipedia.org/wiki/%D0%A8%D0%B5%D0%B9%D0%B4%D0%B5%D1%80), и затем материал прикрепится к модели. Редактор Unity поддерживает написание и редактирование шейдеров. Редактор Unity имеет компонент для создания анимации, но также анимацию можно создать предварительно в 3D-редакторе и импортировать вместе с моделью, а затем разбить на файлы.

Unity 3D поддерживает систему [Level Of Detail](https://ru.wikipedia.org/wiki/Level_of_Detail" \o "Level of Detail) (сокр. LOD), суть которой заключается в том, что на дальнем расстоянии от игрока высокодетализированные модели заменяются на менее детализированные, и наоборот, а также систему Occlusion culling, суть которой в том, что у объектов, не попадающих в поле зрения камеры не визуализируется геометрия и коллизия, что снижает нагрузку на центральный процессор и позволяет оптимизировать проект. При компиляции проекта создается [исполняемый](https://ru.wikipedia.org/wiki/%D0%98%D1%81%D0%BF%D0%BE%D0%BB%D0%BD%D0%B8%D0%BC%D1%8B%D0%B9_%D0%BC%D0%BE%D0%B4%D1%83%D0%BB%D1%8C) (.exe) файл игры (для [Windows](https://ru.wikipedia.org/wiki/Windows" \o "Windows)), а в отдельной папке — данные игры (включая все игровые уровни и динамически подключаемые библиотеки).

Движок поддерживает множество популярных форматов. Модели, звуки, текстуры, материалы, скрипты можно запаковывать в формат .unityassets и передавать другим разработчикам, или выкладывать в свободный доступ. Этот же формат используется во внутреннем магазине Unity Asset Store, в котором разработчики могут бесплатно и за деньги выкладывать в общий доступ различные элементы, нужные при создании игр. Чтобы использовать Unity Asset Store, необходимо иметь аккаунт разработчика Unity. Unity имеет все нужные компоненты для создания мультиплеера. Также можно использовать подходящий пользователю способ контроля версий. К примеру, Tortoise SVN или Source Gear.

В Unity входит Unity Asset Server — инструментарий для совместной разработки на базе Unity, являющийся дополнением, добавляющим контроль версий и ряд других серверных решений.

### 1.3 Достоинства и недостатки Unity

Как правило, игровой движок предоставляет множество функциональных возможностей, позволяющих их задействовать в различных играх, в которые входят моделирование физических сред, карты нормалей, динамические тени и многое другое. В отличие от многих игровых движков, у Unity имеется два основных преимущества: наличие [визуальной среды разработки](https://ru.wikipedia.org/wiki/RAD_(%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5)) и [межплатформенная](https://ru.wikipedia.org/wiki/%D0%9A%D1%80%D0%BE%D1%81%D1%81%D0%BF%D0%BB%D0%B0%D1%82%D1%84%D0%BE%D1%80%D0%BC%D0%B5%D0%BD%D0%BD%D0%BE%D1%81%D1%82%D1%8C) поддержка. Первый фактор включает не только инструментарий визуального моделирования, но и [интегрированную среду](https://ru.wikipedia.org/wiki/%D0%98%D0%BD%D1%82%D0%B5%D0%B3%D1%80%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%BD%D0%B0%D1%8F_%D1%81%D1%80%D0%B5%D0%B4%D0%B0_%D1%80%D0%B0%D0%B7%D1%80%D0%B0%D0%B1%D0%BE%D1%82%D0%BA%D0%B8), [цепочку сборки](https://ru.wikipedia.org/wiki/%D0%90%D0%B2%D1%82%D0%BE%D0%BC%D0%B0%D1%82%D0%B8%D0%B7%D0%B0%D1%86%D0%B8%D1%8F_%D1%81%D0%B1%D0%BE%D1%80%D0%BA%D0%B8), что направлено на повышение производительности разработчиков, в частности, этапов создания прототипов и тестирования. Под межплатформенной поддержкой предоставляется не только места развертывания (установка на персональном компьютере, на мобильном устройстве, консоли и т. д.), но и наличие инструментария разработки (интегрированная среда может использоваться под Windows и Mac OS).

Третьим преимуществом называется модульная система компонентов Unity, с помощью которой происходит конструирование игровых объектов, когда последние представляют собой комбинируемые пакеты функциональных элементов. В отличие от механизмов [наследования](https://ru.wikipedia.org/wiki/%D0%9D%D0%B0%D1%81%D0%BB%D0%B5%D0%B4%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5_(%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5)), объекты в Unity создаются посредством [объединения функциональных блоков](https://ru.wikipedia.org/wiki/%D0%9F%D1%80%D0%B8%D0%BC%D0%B5%D1%81%D1%8C_(%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5)), а не помещения в узлы дерева наследования. Такой подход облегчает создание прототипов, что актуально при разработке игр.

Имея большое количество преимуществ, в Unity присутствуют и недостатки. В качестве недостатков приводятся ограничение визуального редактора при работе с многокомпонентными схемами, когда в сложных сценах визуальная работа затрудняется. Вторым недостатком называется отсутствие поддержки Unity ссылок на внешние [библиотеки](https://ru.wikipedia.org/wiki/%D0%91%D0%B8%D0%B1%D0%BB%D0%B8%D0%BE%D1%82%D0%B5%D0%BA%D0%B0_(%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5)), работу с которыми программистам приходится настраивать самостоятельно, и это также затрудняет командную работу. Ещё один недостаток связан с использованием шаблонов экземпляров ([англ.](https://ru.wikipedia.org/wiki/%D0%90%D0%BD%D0%B3%D0%BB%D0%B8%D0%B9%D1%81%D0%BA%D0%B8%D0%B9_%D1%8F%D0%B7%D1%8B%D0%BA) prefabs). С одной стороны, эта концепция Unity предлагает гибкий подход визуального редактирования объектов, но с другой стороны, редактирование таких шаблонов является сложным. Также, [WebGL](https://ru.wikipedia.org/wiki/WebGL" \o "WebGL)-версия движка, в силу специфики своей архитектуры (трансляция кода из C# в С++ и далее в JavaScript), имеет ряд нерешённых проблем с производительностью, потреблением памяти и работоспособностью на мобильных устройствах.

Ещё один недостаток – это потеря точности [float](https://ru.wikipedia.org/wiki/%D0%A7%D0%B8%D1%81%D0%BB%D0%BE_%D1%81_%D0%BF%D0%BB%D0%B0%D0%B2%D0%B0%D1%8E%D1%89%D0%B5%D0%B9_%D0%B7%D0%B0%D0%BF%D1%8F%D1%82%D0%BE%D0%B9" \o "Число с плавающей запятой) в [компоненте Transform](https://docs.unity3d.com/ru/530/Manual/class-Transform.html) по всем трем осям. Начинается это проявляться на координатах 10000 и больше.

### 

# 2. Процесс разработки игры

# 2.1 Описание игры «Racing»

Данная игра представляет собой простую 3D-гонку.

В игре реализованы:

* меню;
* управление машиной;
* другие машины;
* аварии;
* бесконечная дорога;
* обработка выезда за пределы дороги;
* очки.

Игра начинается со сцены Menu, где есть две кнопки: Play (для начала игры и перехода к сцене Main) и Exid (для выхода из игры).

В этой гонке игрок управляет машиной. Его цель – проехать как можно большее расстояние от начала, не столкнувшись по пути с другими машинами. Другие машины случайным образом генерируются на дороге с определенной вероятностью и едут с постоянной скоростью. В случае столкновения с одной из них машина игрока разбивается (у нее отваливаются колеса), игра заканчивается, открывается меню.

Управление своей машиной игрок осуществляет с помощью клавиш:

* ↑ или W – увеличение скорости машины (но не больше максимально возможного);
* → или D – движение в правый бок;
* ← или A – движение в левый бок.

Бесконечная дорога генерируется по мере движения игрока добавлением новых дорожных блоков (roadBloks). Перед машиной будет добавляться столько дорожных блоков, сколько нужно, чтобы не было видно пропасти. Когда машина проезжает достаточное расстояние от определённого блока, он удаляется, чтобы освободить оперативную память. Выезд машины за пределы дороги предотвращают коллайдеры (Box Collider), стоящие по бокам от дороги.

Очки (scores) игрока равны проеханному им расстоянию по координате x. После смерти очки игрока сравниваются с сохраненным в игре максимальным результатом (highScore). В случае, если игрок набрал больше очков, его результат становится новым highScore. Во время игры максимальный результат и текущий результат игрока отображаются в левом верхнем углу экрана.

## 2.2 Описание графического интерфейса

В качестве фона в игре мы видим небо. Всего в игре 10 вариантов изображения неба, они заранее скачаны с бесплатного сегмента Assets Store (официального сайта для разработчиков Unity) и занесены в массив. В каждой игре фон выбирается случайным образом.

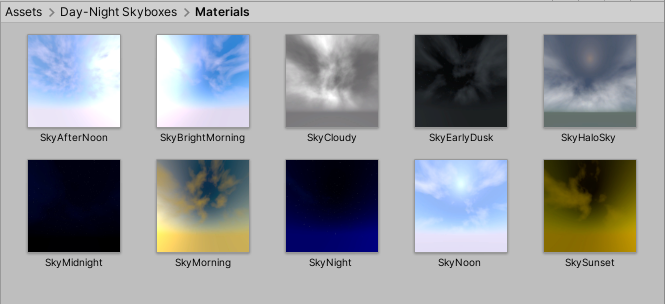


Рисунок 2.2.1 – Все варианты изображения неба в игре

В игре также используются модели машины, сломанной машины и участка дороги. Их изображения представлены ниже.

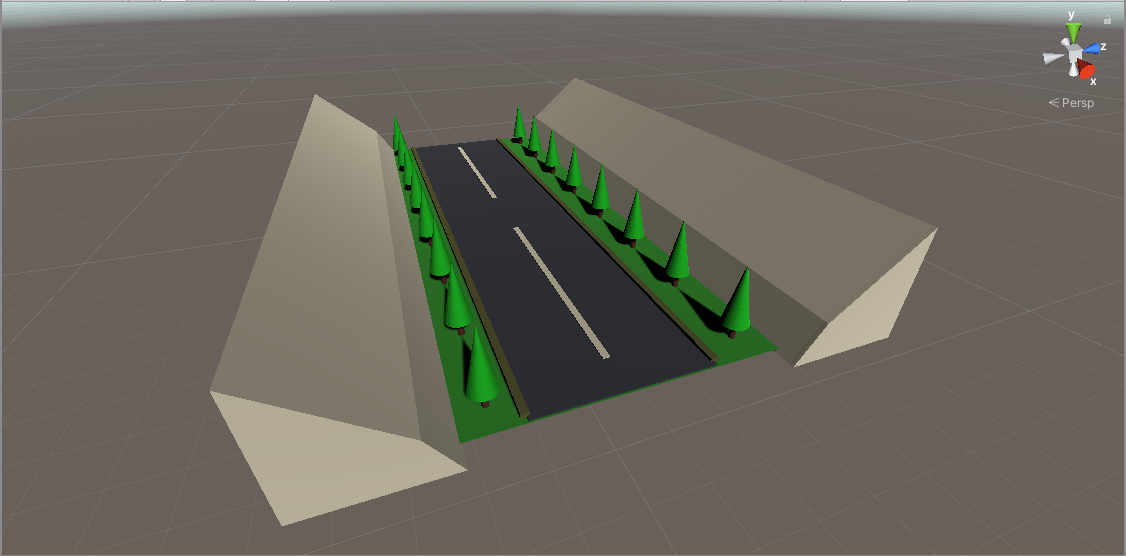


Рисунок 2.2.2 – Модель дорожного блока



Рисунок 2.2.3 – Модель машины

# 3. Демонстрация работы игры

При запуске игры пользователю открывается меню.

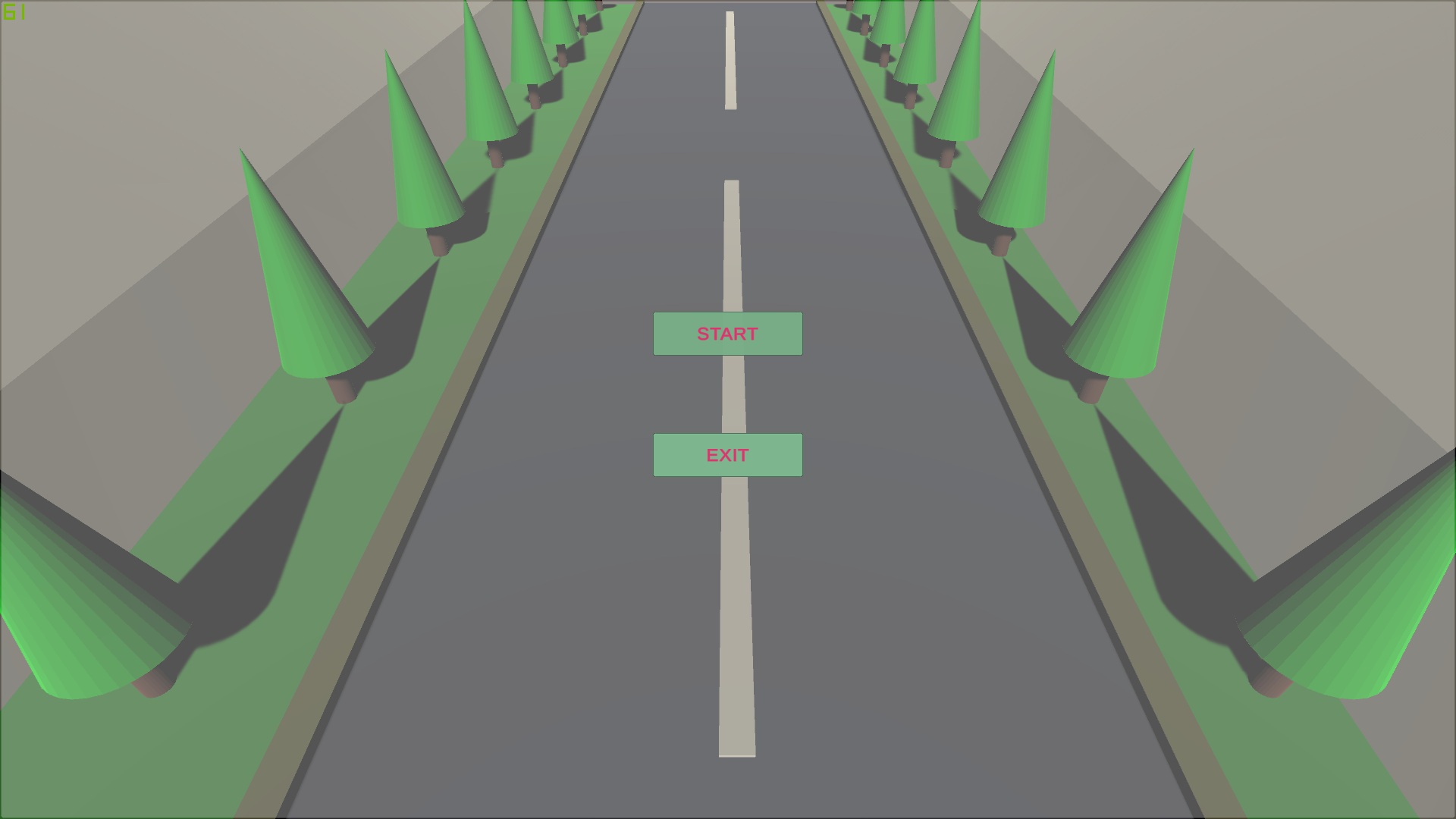


Рисунок 3.1 – Меню игры



Рисунок 3.2 – Скриншот игры



Рисунок 3.3 – Скриншот игры при попытке выехать с дороги



Рисунок 3.4 – Скриншот игры при аварии

# Заключение

При выполнении данной работы разработана игра «Racing» на платформе Unity 3D, а также описан сам процесс разработки.

Данная игра не является законченным продуктом, и по своей сути является прототипом, стоящим примерно на уровне ранней альфа-версии. Вызвано это было тем, что создание качественного продукта занимает колоссально количество времени. Также можно использовать любой из элементов данной игры в любом другом проекте Unity, или же выложить их в магазин Unity, чтобы поделиться с сообществом.

Были рассмотрены основные аспекты разработки игр в целом и на Unity в частности. Платформа показала себя как очень гибкий, удобный и простой инструмент для создания игр, не требующий установки дополнительного программного обеспечения (для разработки игры использовался только Unity), при этом значительных недостатков при работе с платформой замечено не было. Unity постоянно развивается и довольно часто выходят обновления, в которых появляются новые возможности, а также улучшаются старые, поэтому некоторые аспекты данной работы могут не совпадать с версией новейшей платформы.

# Список использованных источников

1. Официальная документация Unity [Электронный ресурс]. — Электронные данные. — Режим доступа: <https://docs.unity3d.com/Manual/index.html>

2. Официальный сайт Unity [Электронный ресурс]. — Электронные данные. — Режим доступа: https://unity.com/

3. Бонд Джереми Гибсон. Unity и С#. Геймдев от идеи до реализации. 2-е издание. Питер, 2019. – 928 с.

4. Алан Торн. Основы анимации в Unity: ДМК-СПб, 2016. – 176 с.

5. Мэннинг Джон, Батфилд-Эддисон Пэрис. Unity для разработчика: Питер, 2018. – 352 с.

# Приложение 1. Код программы

**Menu.cs:**

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

using UnityEngine.SceneManagement;

public class Menu : MonoBehaviour

{

public void Play()

{

SceneManager.LoadScene("SampleScene");

}

public void Exit()

{

Application.Quit();

}

}

**RandBG.cs:**

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

public class RandBG : MonoBehaviour

{

public Material[] materials;

void Start()

{

GetComponent <Skybox> ().material = materials[Random.Range(0, 9)];

}

void Update()

{

}

}

**RoadBlock.cs:**

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

public class RoadBlock : MonoBehaviour

{

public bool Fetch(float x)

{

bool result = false;

if(x > transform.position.x + 100f)

{

result = true;

}

return result;

}

public void Delete()

{

Destroy(gameObject);

}

}

**Road.cs:**

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

public class Road : MonoBehaviour

{

public List<GameObject> blocks;

public GameObject player;

public GameObject roadPrefab;

public GameObject carPrefab;

public GameObject carPrefab2;

public GameObject coinPrefab;

private System.Random rand = new System.Random();

void Update()

{

float x = player.GetComponent<Moving>().rb.position.x;

var last = blocks[blocks.Count - 1];

if(x > last.transform.position.x - 24.69f \* 10f)

{

var block = Instantiate(roadPrefab, new Vector3(last.transform.position.x + 24.69f, last.transform.position.y, last.transform.position.z), Quaternion.identity);

block.transform.SetParent(gameObject.transform);

blocks.Add(block);

if(rand.Next(0, 100) > 80)

{

float side = rand.Next(1, 3) == 1 ? -1f : 1f;

var car = Instantiate(carPrefab2, new Vector3(last.transform.position.x + 24.69f, last.transform.position.y + 0.20f, last.transform.position.z + 1.30f \* side), Quaternion.Euler(new Vector3(0f, 90f, 0f)));

car.transform.SetParent(gameObject.transform);

}

}

int flag = 1;

List<GameObject> toRemove = new List<GameObject>();

foreach (GameObject block in blocks)

{

bool fetched = block.GetComponent<RoadBlock>().Fetch(x);

if(fetched && flag == 0)

{

toRemove.Add(block);

block.GetComponent<RoadBlock>().Delete();

}

flag = 0;

}

foreach(var a in toRemove)

{

blocks.Remove(a);

}

}

}

**Controls.cs:**

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

using System;

using System.IO;

using System.Text;

using UnityEngine.SceneManagement;

using UnityEngine.Audio;

public class Controls : MonoBehaviour

{

public float speed = 10f;

public float maxSpeed = 30f;

public float minSpeed = 5f;

public float sideSpeed = 0f;

public float scores = 0f;

public float highScore = 0f;

void Start()

{

}

void Update()

{

using (FileStream fstream = File.OpenRead($"D:/highscore.txt"))

{

byte[] array = new byte[fstream.Length];

fstream.Read(array, 0, array.Length);

string textFromFile = System.Text.Encoding.Default.GetString(array);

Debug.Log(textFromFile);

highScore = float.Parse(textFromFile);

}

float moveSide = Input.GetAxis("Horizontal");

float moveForward = Input.GetAxis("Vertical");

if(moveSide != 0)

{

sideSpeed = moveSide \* -0.1f;

}

if(moveForward != 0)

{

speed += 0.1f \* moveForward;

}

else

{

if(speed > 0)

{

speed -= 0.1f;

}

else

{

speed += 0.1f;

}

}

if(speed > maxSpeed)

{

speed = maxSpeed;

}

if(speed < minSpeed)

{

speed = minSpeed;

}

}

}

**Moving.cs:**

using System.Collections;

using System.Collections.Generic;

using System;

using System.IO;

using System.Text;

using UnityEngine;

using UnityEngine.SceneManagement;

using UnityEngine.Audio;

public class Moving : MonoBehaviour

{

public Rigidbody rb;

public GameObject car;

public GameObject brokenPrefab;

public GameObject modelHolder;

public Controls control;

private float speed = 0.1f;

private float maxSpeed;

private float minSpeed;

private bool isAlive = true;

private bool isKilled = false;

public List<GameObject> wheels;

private System.Random rand = new System.Random();

void Start()

{

rb = GetComponent<Rigidbody>();

maxSpeed = control.maxSpeed;

minSpeed = control.minSpeed;

}

void Update()

{

if(isAlive)

{

float newSpeed = speed;

float sideSpeed = 0f;

if(control != null)

{

newSpeed += control.speed;

sideSpeed = control.sideSpeed;

control.scores += (newSpeed - minSpeed) / 100;

if(newSpeed > maxSpeed)

{

newSpeed = maxSpeed;

}

if(newSpeed < minSpeed)

{

newSpeed = minSpeed;

}

rb.velocity = new Vector3(newSpeed, 0, newSpeed \* sideSpeed);

} else {

newSpeed = 0.1f;

transform.position = new Vector3(transform.position.x + newSpeed, transform.position.y, transform.position.z);

}

if(control != null)

{

control.sideSpeed = 0f;

}

if(wheels.Count > 0)

{

foreach (var wheel in wheels)

{

wheel.transform.Rotate(-3f, 0f, 0f);

}

}

if(tag == "Car")

{

if(transform.position.y < -50f)

{

Destroy(gameObject.GetComponent<BoxCollider>());

Destroy(gameObject);

}

}

}

}

void OnTriggerEnter(Collider other)

{

if(other.tag == "Car" || other.tag == "Wall")

{

isAlive = false;

if(car != null)

{

if(!isKilled)

{

rb.velocity = new Vector3(0, 0, 0);

Destroy(car);

var broken = Instantiate(brokenPrefab, transform.position, Quaternion.Euler(new Vector3(0f, -270f, 0f)));

broken.transform.SetParent(modelHolder.transform);

isKilled = true;

StartCoroutine("Die");

}

}

}

}

IEnumerator Die()

{

if(Math.Floor(control.highScore) < Math.Floor(control.scores))

{

string text = Math.Floor(control.scores).ToString();

using (FileStream fstream = new FileStream($"D:/highscore.txt", FileMode.OpenOrCreate))

{

byte[] array = System.Text.Encoding.Default.GetBytes(text);

fstream.Write(array, 0, array.Length);

}

}

yield return new WaitForSeconds(4f);

SceneManager.LoadScene("Menu");

}

}

**Scores.cs:**

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

using TMPro;

using System;

public class Scores : MonoBehaviour

{

private TextMeshProUGUI text;

private Controls controls;

public GameObject player;

void Start()

{

text = GetComponent<TextMeshProUGUI>();

controls = player.GetComponent<Controls>();

}

void Update()

{

if(controls != null)

{

text.text = $"Highscore: {Math.Floor(controls.highScore)} | Scores: {Math.Floor(controls.scores)}";

}

}

}