МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное автономное
образовательное учреждение высшего образования
«Южно-Уральский государственный университет
(национальный исследовательский университет)»

Высшая школа электроники и компьютерных наук
Кафедра «Электронные вычислительные машины»

ДОПУСТИТЬ К ЗАЩИТЕ
Заведующий кафедрой ЭВМ
_____ Д.В. Топольский
«___»_____ 2023 г.

Разработка симулятора для изучения теории графов

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА
К ВЫПУСКНОЙ КВАЛИФИКАЦИОННОЙ РАБОТЕ
ЮУРГУ-090301.2023.421 ПЗ ВКР

Руководитель работы,
к.т.н., доцент каф. ЭВМ
_____ К.А. Домбровский
«___»_____2023 г.

Автор работы,
студент группы КЭ-406
_____П.Е. Некрасов
«___»_____2023 г.

Нормоконтролёр,
ст. преп. каф. ЭВМ
_____С. В. Сяськов
«___»_____2023 г.

Челябинск-2023

АННОТАЦИЯ

П.Е. Некрасов. Разработка симулятора для изучения теории графов. - Челябинск: ФГАОУ ВО «ЮУрГУ (НИУ)», ВШ ЭКН; 2023, 51 с., 15 ил., библиогр. список – 19 наим.

В рамках выпускной квалификационной работы воспроизводится процесс разработки симулятора для изучения теории графов. В работе рассматриваются преимущества и недостатки существующих программных комплексов. Описывается процесс разработки симулятора, включая выбор технологических решений, проектирования архитектуры и функционала симулятора. Доказывается способность разрабатываемого программного продукта конкурировать с существующими программными продуктами.

В результате работы получается полноценный симулятор для изучения теории графов, который может быть использован в учебных целях.

ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ

Современные технологии дистанционного обучения широко применяются в различных образовательный учреждениях и обучающих центрах. Вместе с этим недостаточное развитие получила технология разработки симуляторов для проведения лабораторных работ в виртуальном режиме. Обучающая среда для дистанционного обучения в основном ограничивается текстовыми, графическими, анимационными и мультимедийными учебными материалами, представляемыми в электронном виде [1].

На рисунке 1 показано сравнение различных техник обучения, в том числе имитации реальной деятельности [2].

Рисунок 1 - Сравнение различных техник обучения

Граф – это дискретная математическая структура, которая представляет собой совокупность точек, попарно соединенных линиями или стрелками. Представление в виде графа позволяет наглядно продемонстрировать структуру сложного объекта или системы, взаимосвязи между ее элементами.

Теория графов имеет практическое применение почти во всех отраслях научных знаний: математике, физике, биологии, химии, истории, лингвистике, технике и т.д.

При обучении на IT-специальностях студенты сталкиваются со списками, стеками, очередям, бинарными деревьями и т. п. Все эти структуры можно

рассматривать как графы. Приложения теории графов — это фундаментальные свойства подобных структур.

Теория графов к настоящему времени содержит много эффективных инструментов для решения столь же широкого круга проблем.

Разрабатываемое приложение предоставит пользователю источник учебной информации, раскрывающей в доступной для учащихся форме предусмотренного образовательным стандартом содержания, а также предоставит пользователю интерактивную составляющую для улучшения процесса обучения.

Целью данной работы является создание симулятора для изучения теории графов.

Для выполнения поставленной перед нами цели, необходимо решить следующие задачи:

1. Обзор и анализ программного обеспечения для создания графов;
2. Анализ и выбор средств реализации проекта;
3. Архитектурное проектирование приложения;
4. Программная реализация и тестирование проекта.

Методы исследования, применяемые в работе:

– аналитический метод;

– метод моделирования;

– сравнительный метод.

1 ОБЗОР И АНАЛИЗ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ ДЛЯ СОЗДАНИЯ ГРАФОВ

В настоящее время для визуализации положений теории графов применяют редакторы графов и онлайн-редакторы графов.

Для теории графов существует такие программы:

– Graphviz [3];

– yEd Graph Editor [4];

– Graph Builder.

Graphviz - это программное обеспечение для визуализации графов, заданных в виде описания на языке DOT. Графическая визуализация - это способ представления структурной информации в виде диаграмм абстрактных графов и сетей. Разработан специалистами лаборатории AT&T, распространяется с открытыми исходными файлами по лицензии Eclipse Public License [3].

Программное обеспечение состоит из набора утилит командной строки и программ, способных обрабатывать файлы на языке DOT, а также из виджетов и библиотек, облегчающих создание графов и программ для построения графов.

Недостатки:

– отсутствует интерфейс в привычном понимании;

– для использования программы необходимо знать язык DOT.

yEd Graph Editor – это приложение, используемое для быстрого создания диаграмм и графов.

yEd защищена проприетарной лицензией, но распространяется бесплатно. Просмотреть программный код нельзя, однако пользоваться программой может любой желающий. Она работает на Windows, Mac OS, Linux и других операционных системах. Создатель программы — немецкая компания yWorks GmbH [5].

Недостатки:

– сложный пользовательский интерфейс;

– ограниченный набор функций;

– отсутствие возможности работы с большими объемами данных в автоматическом режиме.

Graph Builder – программа-визуализатор четырех наиболее популярных алгоритма на графах: Дейкстры, Флойда, Прима и Краскалла. Имеет возможность настройки цвета и толщины линий рёбер, цвета и размера вершин, времени задержки. Также обладает функцией загрузки графов на удалённый сервер для хранения и последующей выгрузки на компьютер пользователя.

Недостатки:

– малый функционал программы;

– небольшой список предоставленных алгоритмов.

В таблице 1 представлен сравнительный анализ программ для построения графов.

Таблица 1 – Сравнительный анализ программ для построения графов

	Широкий функционал	Удобный интерфейс	Поддержка последних версий windows
Graphviz	+	-	+
yEd Graph Editor	-	-	+
Graph Builder	-	+	-

Существует большое количество веб-построителей графов. Самые популярные среди них:

– проект Граф Онлайн [6];

– редактор графов от ООО «Новый семестр» [7];

– редактор графов веб-ресурса Programforyou [8];

– онлайн версия Graphviz [9].

Анализ рассмотренных источников показал, что существует большое количество онлайн-редакторов, они преобладают на рынке и имеют бесчисленное множество аналогов, по сравнению с офлайн-редакторами.

У онлайн-редакторов есть одно из основных достоинств – простота использования.

Однако у всех онлайн-редакторов, имеется один значительный недостаток – невозможность осуществлять работу приложения в офлайн режиме. Ни в одном из рассмотренных веб-построителей не присутствует представление графа в 3D пространстве, которое позволяет рассмотреть индивидуальные случаи.

2 АНАЛИЗ И ВЫБОР СРЕДСТВ РЕАЛИЗАЦИИ ПРОЕКТА

2.1 Функциональные требования

Симулятор должен предоставлять возможность:

– получения теоретической информации по теории графов;

– самостоятельного построения графов;

– изменения построенного графа;

– применения алгоритмов на построенном графе;

– переключения режима отображения графа;

– применения алгоритмов раскраска графа на карте;

– самостоятельного закрашивания карты с применением теории четырёх красок.

2.2 Нефункциональные требования

Симулятор должен соответствовать минимальным требованиям:

– Операционная система: Windows 8.1, 10;

– процессор: Intel Core 2-ядерный, аналогичный AMD или лучше;

– оперативная память: 4 ГБ;

– видеокарта: Nvidia GeForce GT 710, Intel HD Graphics 630 или лучше;

– место на диске: не более 2 ГБ.

Интерфейс симулятора необходимо реализовать на русском языке.

Разработанное приложение должно быть написано на языке программирования C#.

2.3 Среда разработки для реализации проекта

Среда разработки компьютерных игр - это программное обеспечение, которое обеспечивает основу для создания приложения. Он представляет собой ключевой компонент, который позволяет разработчикам реализовывать свои идеи и задумки в жизнь. Хорошая среда разработки компьютерных игр является центральным элементом, который обеспечивает функциональность и возможности для создания

виртуального пространства, с которым пользователи смогут свободно взаимодействовать.

Для достижения целей в рамках выпускной квалификационной работы было необходимо выбрать среду разработки, соответствующий основным критериям:

– хорошо разработанная документация;

– низкий порог вхождения – оценка необходимых знаний для работы с выбранной платформой;

– поддержка и написание скриптов на языке C#;

– широкий инструментарий.

Выбор среды разработки именно для компьютерных игр обусловлен тем, что такая среда разработки облегчают работу с объектами и графикой, что существенно упрощает процесс разработки.

В данной отрасли существует большое количество сред разработки компьютерных игр, среди которых наибольшее влияние оказали 4A Engine, Amazon Lumberyard, Anvil Engine, AppGameKit, Box2D, Clickteam Fusion, Cocos2d, Construct, Creation Engine, CryEngine, Dark Engine, Decima, Defold, Frostbite, GameMaker, Godot, Havok Physics / Destruction, Hero Engine, id Tech, Infinity Engine, IW Engine, LibGDX, Panda 3D, Phaser, RAGE, RPG Maker, Source, Spring, Unity, Unreal Engine, Urho3D.

Большинство перечисленных сред разработки являются узконаправленными и предназначены только для создания видеоигр. Также некоторые среды разработки не способны работать с трёхмерной графикой.

Для рассмотрения были выбраны CryEngine, Unity и Unreal Engine, так как они не ограничены узкой специализацией и могут использоваться для различных целей.

Unity и Unreal Engine – две наиболее часто используемых среды разработки для создания приложений или игр. Они были разработаны как для начинающих разработчиков, так и для профессионалов. Из-за этого они считаются отличными для начинающих и профессиональных разработчиков.

Unreal Engine (UE) – это среда разработки для создания игр. Изначально разработанный для компьютерных игр от первого лица, с тех пор он использовался в различных играх и был принят другими отраслями, в первую очередь киноиндустрией и телевизионной индустрией. В нём можно работать с персонажами, логикой, физикой и графикой игры.

UE разработала компания Epic Games для своей игры под названием Unreal, и после этого среда разработки стала популярной. Его основное отличие — хорошая оптимизация: UE создавался не как отдельный коммерческий продукт, а как рабочий инструмент, и ориентирован он на 3D-игры [10]. На рисунке 2 показан интерфейс Unreal Engine 5.

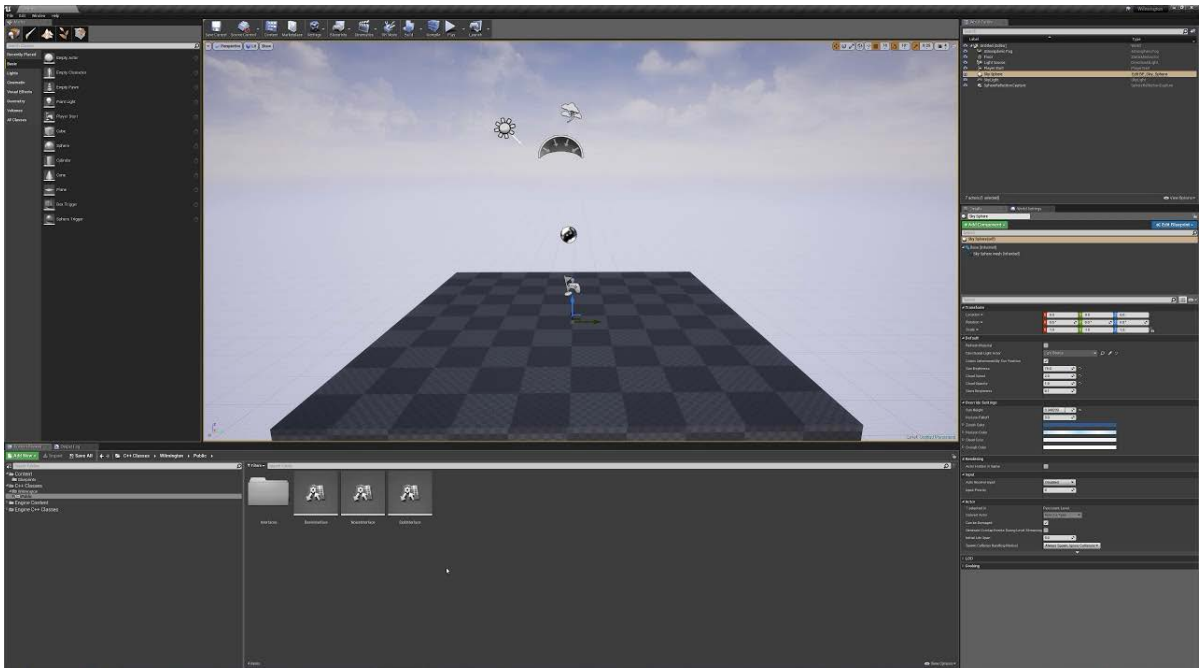

Рисунок 2 – Интерфейс Unreal Engine 5

Достоинства:

– большое сообщество;

– высокая кроссплатформенность;

– широкий ассортимент инструментов для различных целей;

– великолепное графическое решение.

Недостатки:

– высокие минимальные требования к ПК разработчика [11];

– высокий порог вхождения;

– малый выбор готового инструментария в официальном магазине.

Unity — среда разработки компьютерных игр, разрабатываемая и поддерживаемая компанией Unity Technologies. Среда разработки позволяет писать скрипты на языках JavaScript и C#.

Приложение Unity – это инструмент, которым ежедневно пользуются опытные разработчики, а также один из наиболее доступных инструментов для новичков. До недавнего времени человек, решивший научиться программированию игр, сразу же сталкивался с множеством серьезных препятствий, в то время как инструмент Unity позволил значительно облегчить жизнь начинающим разработчикам [12]. На рисунке 3 показан интерфейс Unity.

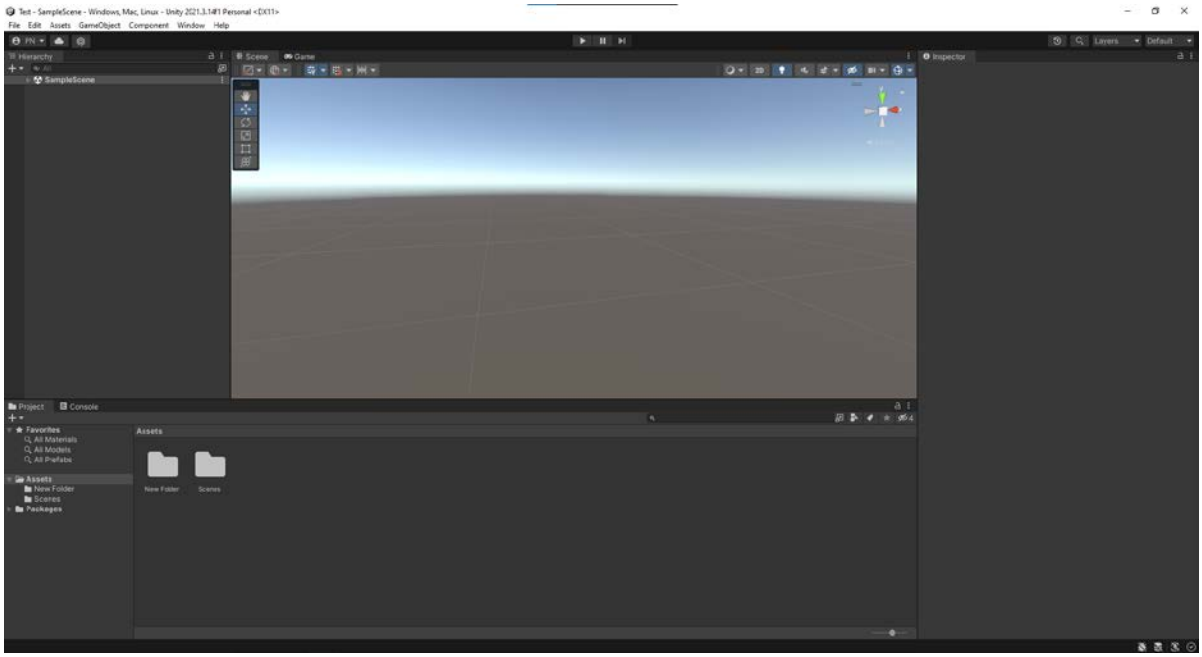

Рисунок 3 – Интерфейс Unity 2021

Достоинства:

– очень обширное сообщество;

– бесплатное учебное пособие на веб-сайте Unity;

– хранилище ресурсов Unity для бесплатных игровых ресурсов;

– низкий порог вхождения;

– возможность дополнения функционала;

– возможность использования систем контроля версий;

– чрезвычайно производительный визуальный рабочий процесс;

– высокая кроссплатформенность;

– низкие системные требования к персональному компьютеру [13];

– большой выбор коммерческих лицензий с фиксированной оплатой [14].

Недостатки:

– ограниченная поддержка пользователей стандартной лицензии;

– процесс создания приложения отнимает большое количество времени;

– не самое передовое графическое решение в сравнении с UE.

Помимо Unity и Unreal Enigne 5, есть чуть менее распространённый среда разработки компьютерных игр CryEngine.

CryEngine – среда разработки компьютерных игр, созданный немецкой компанией Crytek в 2002 году и первоначально используемый в игре от первого лица Far Cry [15].

Первоначально он был полностью платным, а бесплатная версия была доступна только обучающим учреждениям для обучения их учеников и студентов. Но в 2016 году стратегия распространения среды разработки поменялась. Теперь он и средства разработки игр от Crytek доступны абсолютно всем бесплатно до достижения доходов разработчиков с программного продукта свыше 5000$ [16]. На рисунке 4 показан интерфейс CryEngine 5.

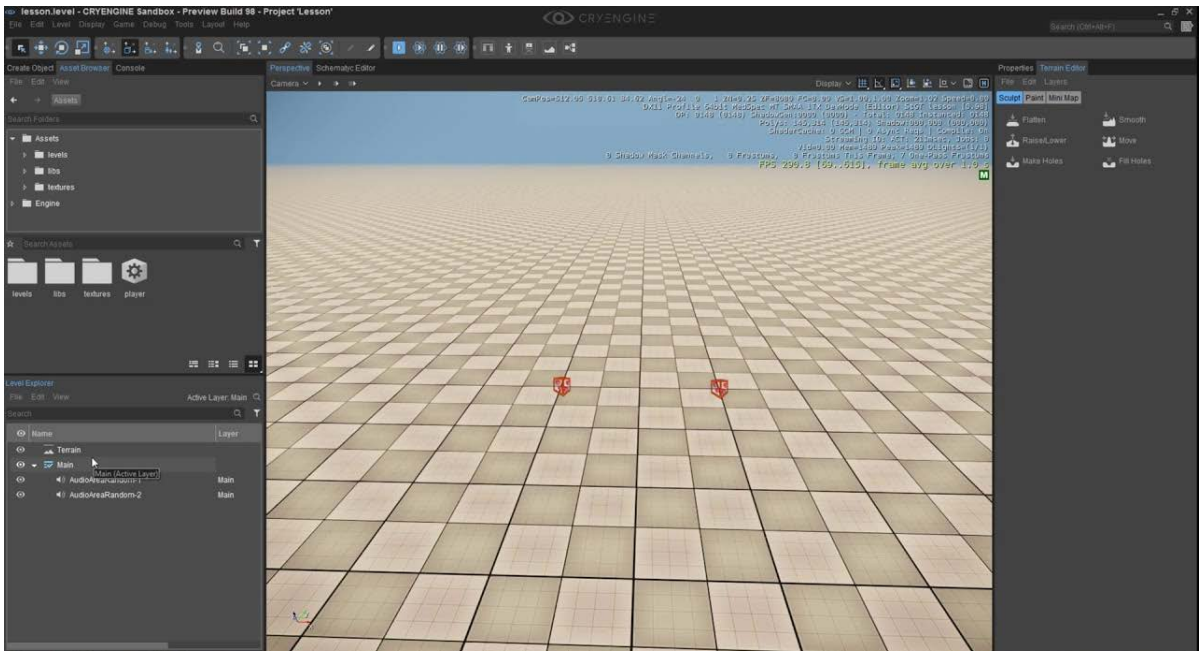

Рисунок 4 – Интерфейс CryEngine 5

Достоинства:

– многие инструменты доступны сразу после установки софта;

– программирование на языке C#;

– низкие системные требования [17];

– великолепное графическое решение.

Недостатки:

– небольшое сообщество;

– отсутствие хорошей техподдержки;

– сложен в освоении;

– трудности при работе с ним, возникающие из-за сложности сборки.

В таблице 2 представлен сравнительный анализ рассматриваемых сред разработки, на основе выдвинутых критериев.

Таблица 2 – Сравнительный анализ сред разработки

	Проработанная документация	Порог вхождения	Поддерживаемый язык	Широкий инструментарий
Unreal Engine 5	+	высокий	C++	+
Unity	+	низкий	C#	+
CryEngine	-	высокий	C++	+

Исходя из поставленных критериев больше всего подходит среда разработки Unity.

2.4 Интегрированная среда разработки

Средой разработки для написания скриптов было выбрано Visual Studio Community 2022 IDE.

Microsoft Visual Studio - это интегрированная среда разработки (IDE) от Microsoft. Он используется для разработки компьютерных программ, включая веб-сайты, веб-приложения, веб-службы и мобильные приложения. Visual Studio использует платформы разработки программного обеспечения Microsoft, такие как Windows API, Windows Forms, Windows Presentation Foundation, Windows Store и

Microsoft Silverlight. Он может создавать как собственный код, так и управляемый код [18].

Visual Studio включает редактор кода, поддерживающий IntelliSense, а также рефакторинг кода. Интегрированный отладчик работает как отладчик исходного кода, так и как отладчик машинного уровня.

3 АРХИТЕКТУРНОЕ ПРОЕКТИРОВАНИЕ ПРИЛОЖЕНИЯ

Разрабатываемое приложение будет построено на архитектуре Model, view, controller (MVC). На рисунке 5 представлена данная модель.

Рисунок 5 – Модель MVC

3.1 Графический план прецедентов

Для описания системы на концептуальном уровне была использована диаграмма прецедентов. На рисунках 6, 7 и 8 представлены диаграммы прецедентов для теоретического блока и блока «Редактор» соответственно.

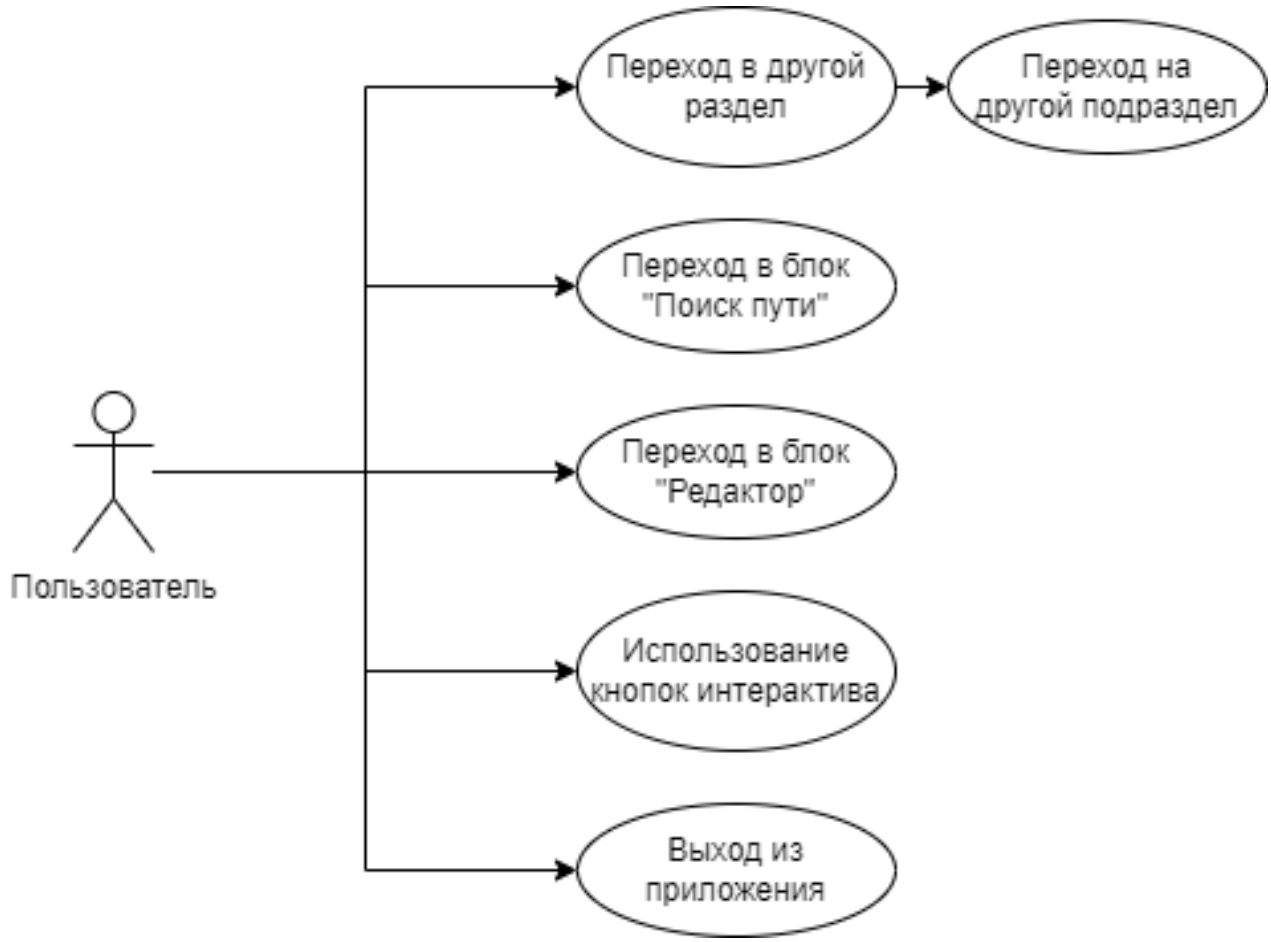

Рисунок 6 – Диаграмма прецедентов теоретического блока

Из представленных на рисунке 6 прецедентов можно выделить прецеденты для теоретического блока:

– переход по разделам;

– переход по подразделам;

– переход в блок «Редактор»;

– переход в блок «Поиск пути»;

– использование кнопок интерактива;

– выход из приложения.

Прецедент «Переход по разделам» позволяет пользователю переключаться между разделами теоретического блока. Позволяет перейти к следующим прецедентам: переход по подразделам.

Прецедент «Переход по подразделам» позволяет пользователю переключаться между подразделами внутри раздела теоретического блока.

Прецедент «Переход в блок «Редактор»» позволяет пользователю перейти в блок «Редактор».

Прецедент «Переход в блок «Поиск пути»» позволяет пользователю перейти в блок «Поиск пути».

Использование кнопок интерактива позволяет пользователю взаимодействовать, если это возможно, с объектами, находящимися на сцене.

Прецедент «Переход в главное меню» позволяет пользователю перейти в главное меню программы.

Из представленных на рисунке 7 прецедентов можно выделить основные прецеденты для блока «Редактор»:

– выбор инструмента;

– выбор алгоритма;

– настройка вершины;

– настройка ребра;

– изменение режима отображения графа;

– переход в теоретический блок;

– переход в блок «Поиск пути»;

– выход из приложения.

Прецедент «Выбор инструмента» позволяет пользователю выбрать инструмент для построения или изменения графа. Позволяет перейти к следующим прецедентам: перемещение вершин, создание вершин, создание рёбер, удаление отдельных объектов, удаление графа.

Прецедент «Выбор алгоритма» позволяет пользователю выбрать алгоритм для применения на графе. Позволяет перейти к следующим прецедентам: применение алгоритма, теория о четырёх красках.

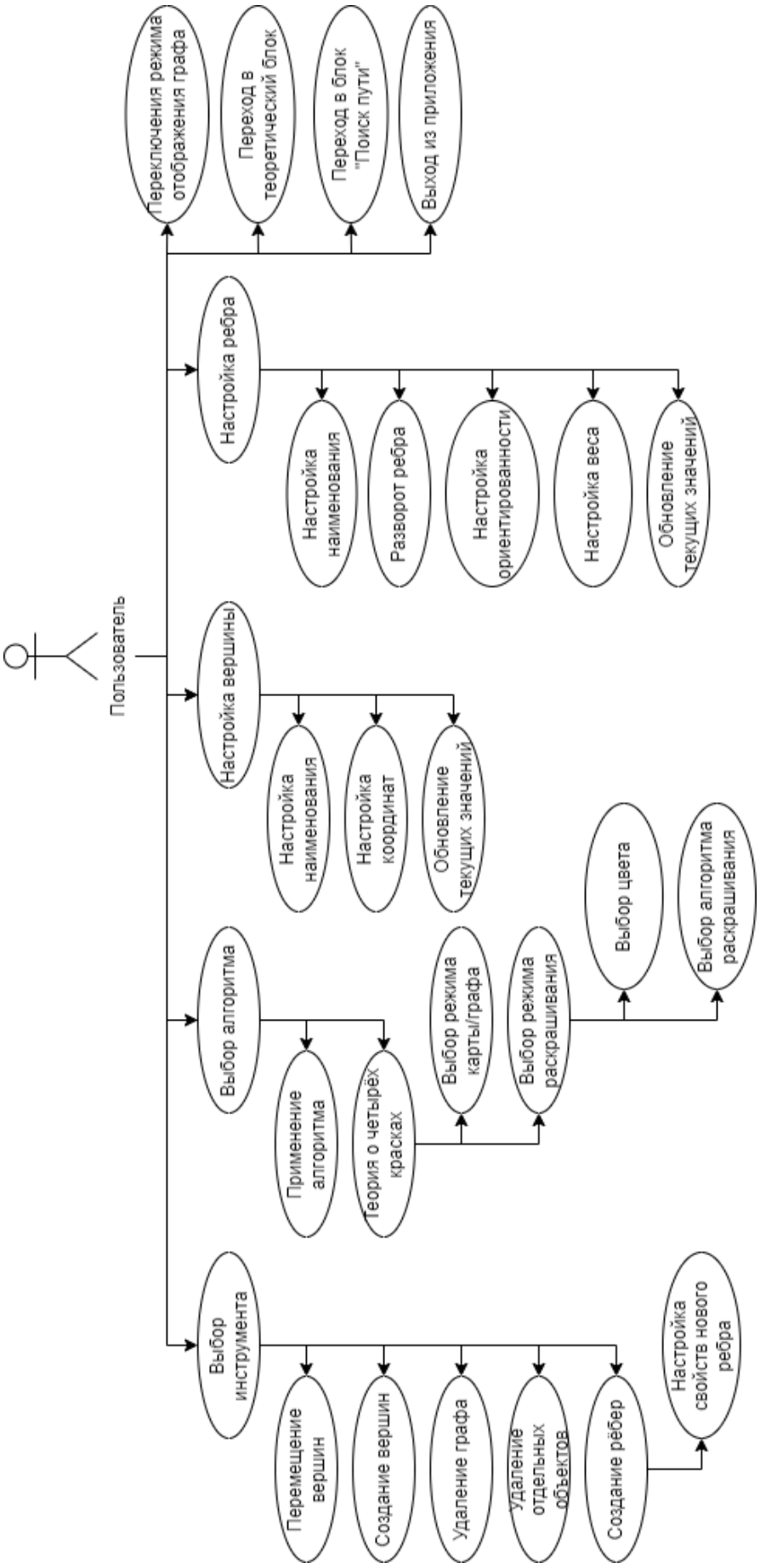

Рисунок 7 – Диаграмма прецедентов блока «Редактор»

Прецедент «Настройка вершины» позволяет пользователю настроить параметры выбранной вершины. Позволяет перейти к следующим прецедентам: настройка наименования, настройка координат, обновление значений.

Прецедент «Настройка ребра» позволяет пользователю настроить параметры выбранного ребра. Позволяет перейти к следующим прецедентам: настройка наименования, настройка ориентированности, настройка разворота, настройка веса, обновление значений.

Прецедент «Переключение режима отображения графа» позволяет пользователю изменить режим отображения графа с ортодоксального на перспективный и наоборот.

Прецедент «Переход в теоретический блок» позволяет пользователю перейти в конкретный раздел теоретического блока.

Прецедент «Переход в блок «Редактор»» позволяет пользователю перейти в блок «Редактор».

Прецедент «Выход из приложения» позволяет пользователю завершить работу приложения.

Из представленных на рисунке 8 прецедентов можно выделить основные прецеденты для блока «Поиск пути»:

– выбор режима;

– режим настройки;

– режим поиска пути;

– настройка перекрёстка;

– настройка дороги;

– настройка множителей;

– выбор начальной и конечной точек маршрута;

– изменение режима отображения карты;

– переход в теоретический блок;

– переход в блок «Редактор»;

– выход из приложения.

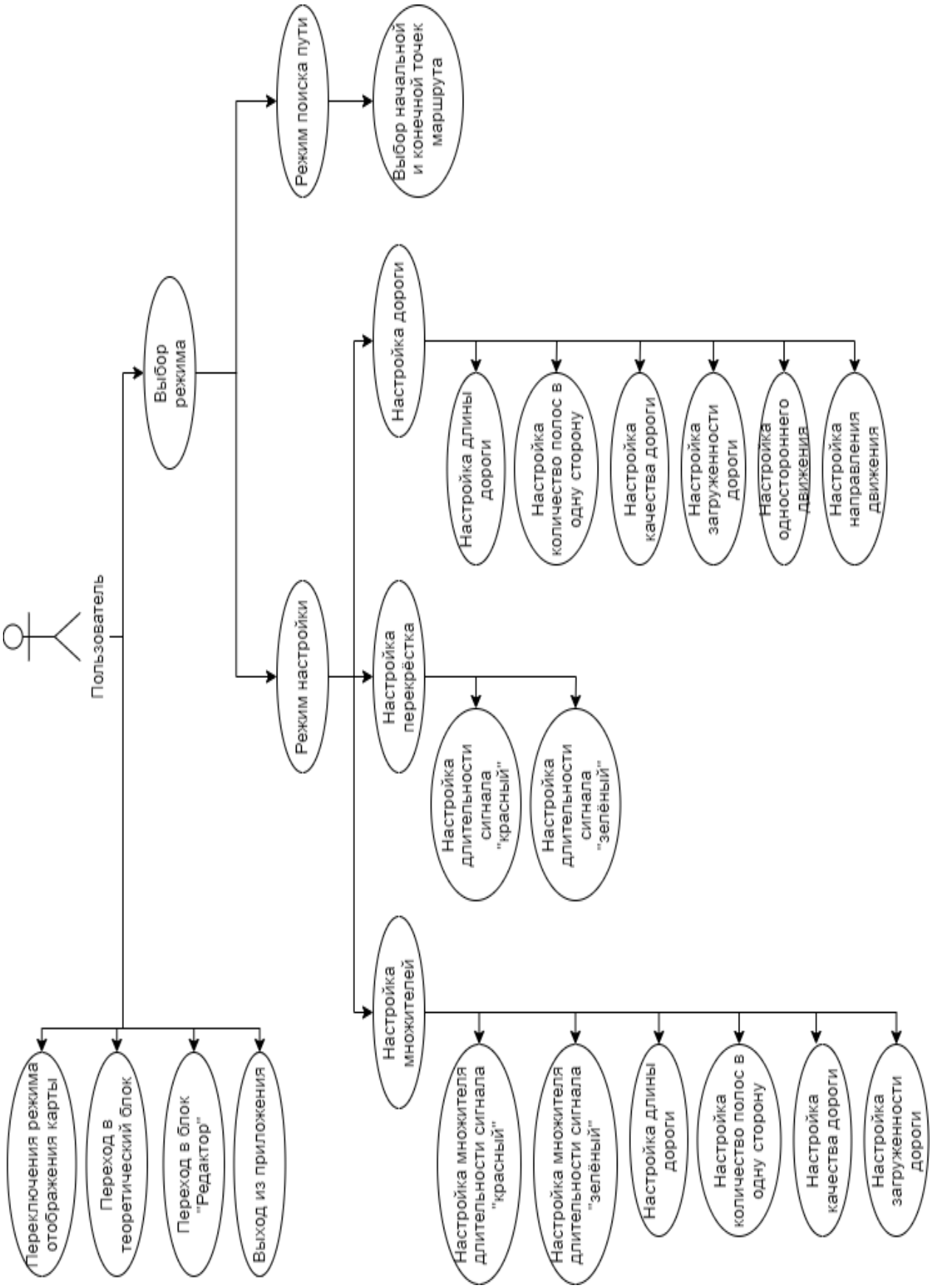

Рисунок 8 – Диаграмма прецедентов блока «Поиск пути»

Прецедент «Выбор режима» позволяет пользователю выбрать режим работы. Позволяет перейти к следующим прецедентам: режим настройки, режим поиска пути.

Прецедент «Режим настройки» позволяет перейти к следующим прецедентам: настройка перекрёстка, настройка дороги, настройка множителей. Не позволяет переходить к прецедентам, доступным из режима поиска пути.

Прецедент «Настройка перекрёстка» позволяет пользователю настроить параметры перекрёстка. Позволяет перейти к следующим прецедентам: настройка длительности сигналов «красный» и «зелёный».

Прецедент «Настройка дороги» позволяет пользователю настроить параметры дороги. Позволяет перейти к следующим прецедентам: настройка длины дороги, настройка количества полос в одну сторону, настройка качества дороги, настройка загруженности дороги, настройка одностороннего движения, настройка направления движения.

Прецедент «Настройка множителей» позволяет пользователю настроить параметры множителей параметров. Позволяет перейти к следующим прецедентам: настройка множителя длительности сигналов «красный» и «зелёный», настройка множителя длины дороги, настройка множителя количества полос в одну сторону, настройка множителя качества дороги, настройка множителя загруженности дороги.

Прецедент «Режим поиска пути» позволяет перейти к следующим прецедентам: выбор начальной и конечной точек маршрута. Не позволяет переходить к прецедентам, доступным из режима настройки.

Прецедент «Переключение режима отображения карты» позволяет пользователю изменить режим отображения карты с 2D-режима на 3D-режим и наоборот.

Прецедент «Переход в теоретический блок» позволяет пользователю перейти в конкретный раздел теоретического блока.

Прецедент «Переход в блок «Редактор»» позволяет пользователю перейти в блок «Редактор».

Прецедент «Выход из приложения» позволяет пользователю завершить работу приложения.

3.2 Концепция интерфейса

В теоретическом блоке было спланировано реализовать следующие элементы:

– панель с теоретической информацией;

– объекты, рисунки или видео, соответствующие подразделу;

– выпадающее меню «Разделы»;

– кнопки и выпадающее меню выбора подраздела;

– кнопки выбора объекта, рисунка или видео, соответствующего подразделу;

– кнопки «Интерактив»;

– кнопка «Выход».

Макет интерфейса теоретического блока представлен на рисунке 9.

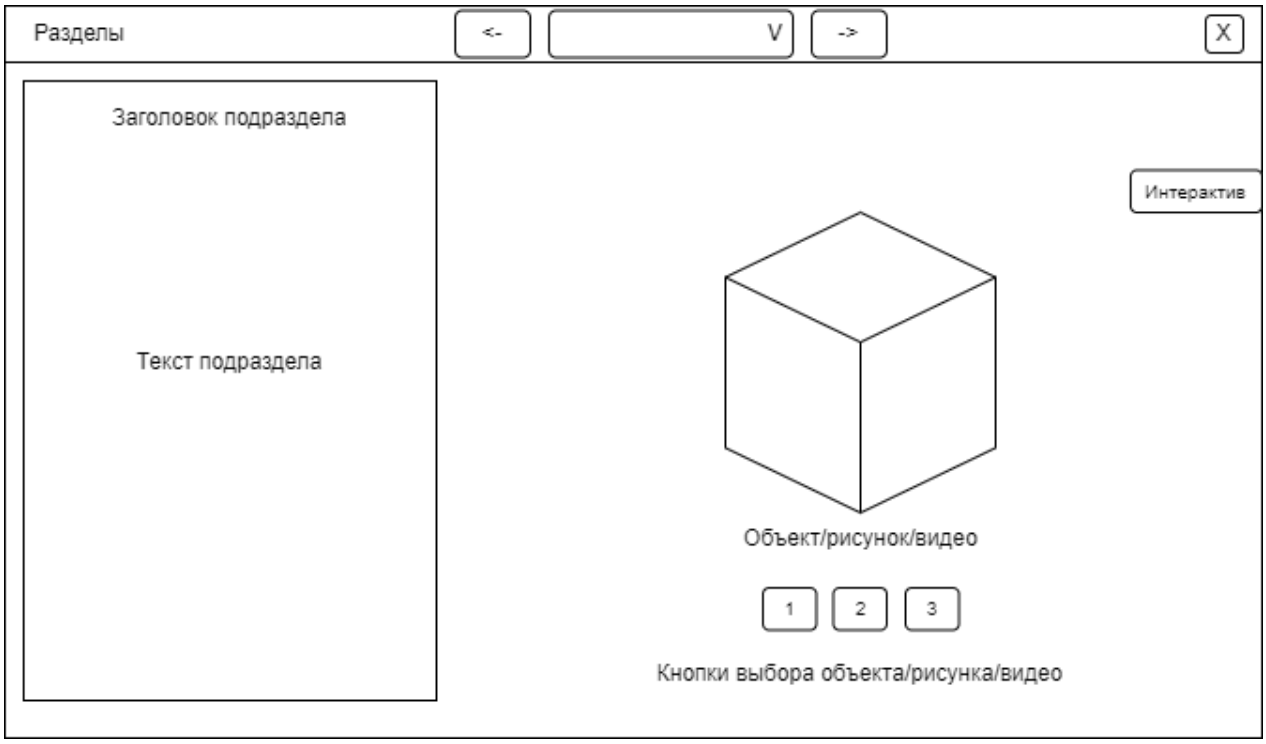

Рисунок 9 – Макет интерфейса теоретического блока

На панели с теоретической информацией предоставлен текст, соответствующий выбранному подразделу.

В области с объектом или рисунком предоставлен объект или рисунок, соответствующий выбранному подразделу.

Используя выпадающее меню «Разделы», пользователь получает возможность перехода на конкретный раздел теоретического блока, а также возможность перейти в блок «Редактор» или блок «Поиск пути».

Используя кнопки перехода по подразделам, пользователю предоставляется возможность перейти на предыдущий или следующий раздел теоретического блока.

Используя кнопки выбора объекта, рисунка или видео, пользователь переключается между объектами, рисунками или видео, соответствующих данному подразделу.

Используя кнопки «Интерактив», пользователь может получить доступ к интерактивным возможностям для взаимодействия с объектом.

Используя кнопку «Выход», пользователь может перейти в главное меню программы или полностью выйти из неё.

В блоке «Редактор» было спланировано реализовать следующие элементы:

– рабочее пространство;

– выпадающее меню «Разделы»;

– выпадающее меню «Инструменты»;

– выпадающее меню «Алгоритмы»;

– выпадающее меню «Вид»;

– панель свойств вершин/рёбер;

– панель свойств создаваемых рёбер;

– кнопка «Выход».

Макет интерфейса блока «Редактор» представлен на рисунке 10.

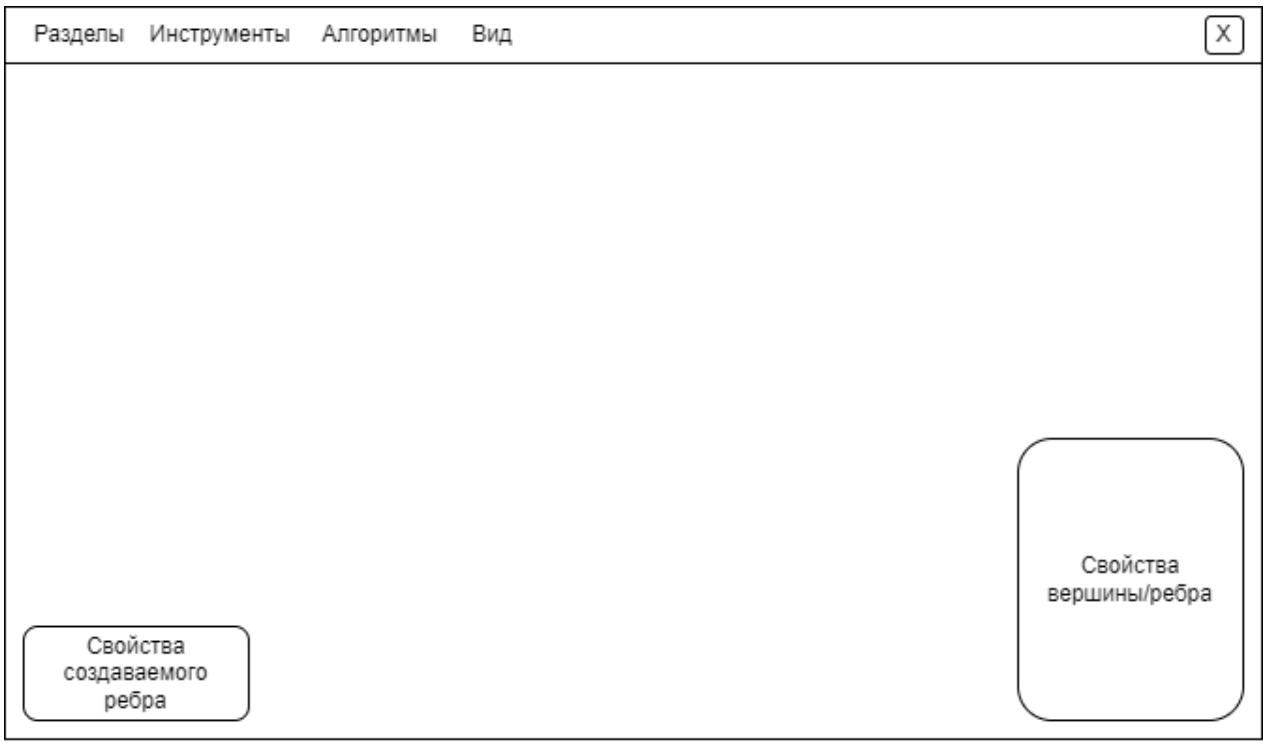

Рисунок 10 – Макет интерфейса блока «Редактор»

Рабочее пространство – пространство в котором пользователь может строить и настраивать граф.

Используя выпадающее меню «Разделы», пользователь получает возможность перехода на конкретный раздел теоретического блока или блок «Поиск пути».

Используя выпадающее меню «Инструменты», пользователь получает возможность выбора режима и кнопка «Удаление». Доступные режимы:

– режим «Перемещение»;

– режим «Создание вершин»;

– режим «Связь»;

– режим «Удаление».

В режиме «Перемещение» пользователь с помощью мыши может перемещать вершины графа.

В режиме «Создание вершин» пользователь может создать создавать вершины. Данный режим недоступен в 3D-режиме отображения.

В режиме «Связь» пользователь, выбрав две вершины графа, может создать между ними ребро. Данный режим недоступен в 3D-режиме отображения.

В режиме «Удаление» пользователь с помощью мыши может удалять выбранные элементы графа. Данный режим недоступен в 3D-режиме отображения.

При нажатии на кнопку «Удаление» пользователь отчистит рабочее пространство, удалив все объекты графа.

Используя выпадающее меню «Алгоритмы», пользователю предоставляется выбор алгоритмов для применения последующих на имеющемся графе. Выпадающее меню является недоступным, в случае, если в рабочем пространстве нет ни одной вершины графа.

Используя выпадающее меню «Вид», пользователь может переключиться между 2D и 3D-режимами отображения графа.

Используя панель свойств вершин/рёбер можно изменять следующие параметры у вершин:

– наименование вершины;

– координаты.

У рёбер:

– наименование ребра;

– вес ребра;

– сделать ребро дугой (ориентированным);

– развернуть дугу.

Используя панель свойств создаваемых рёбер можно заранее выбрать вес ребра и сделать его дугой (ориентированным).

Используя кнопку «Выход», пользователь может перейти в главное меню программы или полностью выйти из неё.

В блоке «Поиск пути» было спланировано реализовать следующие элементы:

– карта улиц города.

– выпадающее меню «Разделы».

– выпадающее меню «Режимы».

– выпадающее меню «Вид».

– панель «Множители».

– панель свойств перекрёстков/дорог.

– кнопка «Выход».

Макет интерфейса блока «Поиск пути» представлен на рисунке 11.

Рисунок 11 – Макет интерфейса блока «Поиск пути»

Рабочее пространство – пространство в котором пользователь может строить и настраивать граф.

Используя выпадающее меню «Разделы», пользователь получает возможность перехода на конкретный раздел теоретического блока или блок «Редактор».

Используя выпадающее меню «Режим», пользователь получает возможность выбора режима. Доступные режимы:

– режим «Настройка»;

– режим «Поиск пути».

– в режиме «Настройка» пользователь имеет возможность настраивать свойства перекрёстков/дорог, а также настраивать множители этих свойств.

– в режиме «Поиск пути» пользователь может применить алгоритм для поиска кратчайшего пути между выбранными точками.

Используя выпадающее меню «Вид», пользователь может переключиться между 2D и 3D-режимами отображения графа.

Используя панель свойств перекрёстков/дорог можно изменять следующие параметры у перекрёстков:

– длительность сигнала «красный»;

– длительность сигнала «зелёный».

У дорог:

– длина дороги;

– количество полос в одну сторону;

– качество дороги;

– загруженность дороги;

– сделать одностороннее движение по данной дороге;

– развернуть направление движения.

Используя панель «Множители» можно изменять множители параметров перекрёстков/дорог:

– множитель длительности сигнала «красный»;

– множитель длительности сигнала «зелёный»;

– множитель длины дороги;

– множитель количества полос в одну сторону;

– множитель качества дороги;

– множитель загруженности дороги.

Используя кнопку «Выход», пользователь может перейти в главное меню программы или полностью выйти из неё.

4 ПРОГРАММНАЯ РЕАЛИЗАЦИЯ И ТЕСТИРОВАНИЕ ПРОЕКТА

Во время реализации проекта использовалась платформа Unity 21, версии 2021.3.14f1, для разработки симуляторов с использованием дополнительных ресурсов из Unity Asset Store, а также сторонних источников.

Графическая составляющая создавалась с помощью редактора Paint.NET. Программная часть разрабатывалась на языке программирования C#.

4.1 Файловая структура проекта

Проект, разрабатываемого симулятора представляет собой список каталогов:

1. Animations – каталог, содержащий контроллеры и анимации для игровых объектов или объектов интерфейса.

2. Materials – каталог, содержащий наборы свойств объектов, с помощью который меняется внешний вид его поверхности, которые называются материалы. В данном каталоге содержатся подкаталоги:

– GraphPoint – подкаталог, содержащий в себе материалы, используемые для вершин графа;

– GraphLine – подкаталог, содержащий в себе материалы, используемые для рёбер графа;

– Map – подкаталог, содержащий в себе материалы, используемые для объектов карты из блока «Поиск пути».

3. Prefabs – каталог, содержащий шаблоны игровых объектов или объектов интерфейса. В данном каталоге содержатся подкаталоги:

– Algorithms – подкаталог, содержащий в себе шаблоны, используемые в визуализации работы алгоритмов;

– Buildings – подкаталог, содержащий в себе шаблоны строений на карте из блока «Поиск пути»;

– Pictures – подкаталог, содержащий в себе шаблоны объектов, используемых в качестве картинок;

– Prebuilds graphs – подкаталог, содержащий в себе шаблоны графов, используемых в теоретическом блоке;

– Roads – подкаталог, содержащий в себе шаблоны дорог и перекрёстков на карте из блока «Поиск пути»;

– UI – подкаталог, содержащий в себе шаблоны элементов интерфейса.

4. Scenes – каталог, содержащий игровые уровни, описывающие организацию объектов в памяти.

5. Scripts – каталог, содержащий скрипты для выполнения как отдельных так и наборов задач. В данном каталоге содержатся подкаталоги:

– Algorithms – подкаталог, содержащий в себе сценарии работы алгоритмов и визуализации их работы;

– GraphEditor – подкаталог, содержащий в себе сценарии работы блока «Редактор»;

– GraphTheory – подкаталог, содержащий в себе сценарии работы теоретического блока. Также имеет подкаталог Interactivity, содержащий в себе сценарии работы интерактивных элементов блока;

– PathFinder – подкаталог, содержащий в себе сценарии работы блока «Поиск пути»;

– UI – подкаталог, содержащий в себе сценарии работы элементов интерфейса.

6. Textures – каталог, содержащий изображения для использования по прямому назначению, либо для применения на игровых объектах и объектах интерфейса для изменения их внешнего вида. В данном каталоге содержатся подкаталоги:

– Icon – подкаталог, содержащий в себе изображения иконок, используемые в объектах интерфейса;

– Pictures – подкаталог, содержащий в себе изображения, используемые в проекте;

– RFRegions – подкаталог, содержащий в себе изображения карты Российской Федерации, используемые в алгоритме о четырёх красках.

7. Videos – каталог, содержащий видеоресурсы проекта.

На рисунке 12 представлена файловая структура проекта.

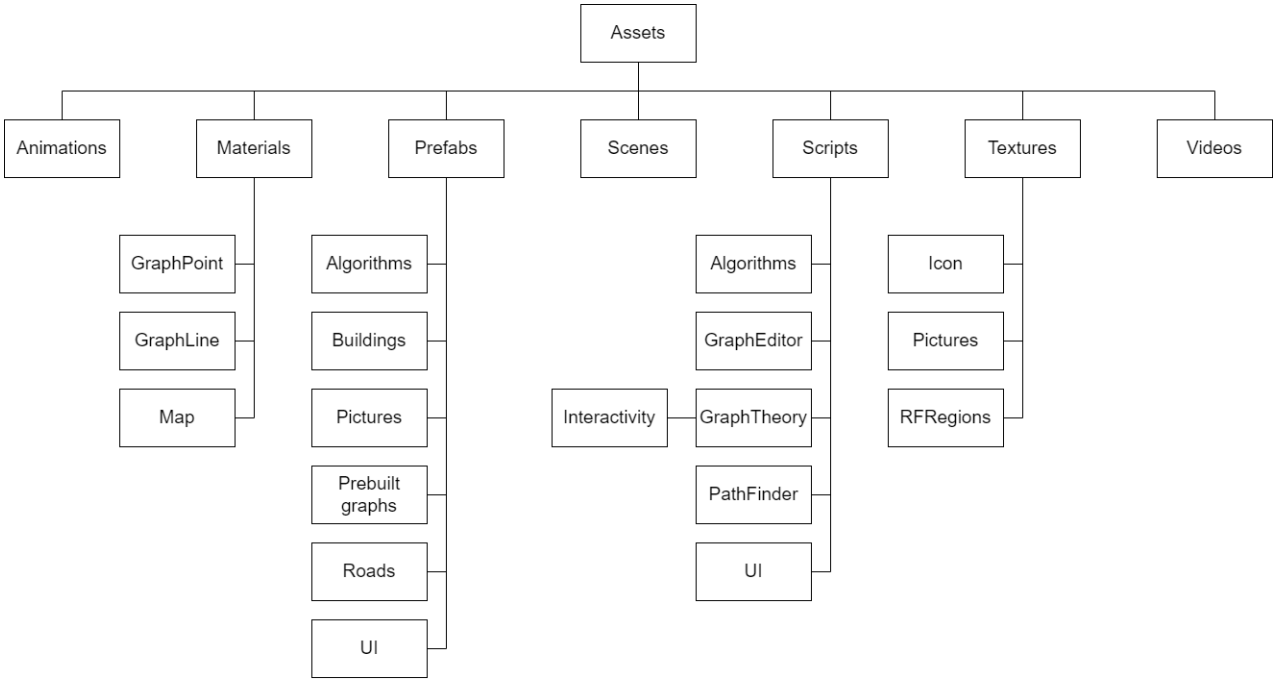

Рисунок 12 – Файловая структура проекта

4.2 Процесс создания графа

Процесс создания графа происходит на сцене GraphTheoryProg. Создание графа начинается с вершины. Используя выпадающее меню «Инструменты», находящееся на верхнем горизонтальном меню, пользователю необходимо выбрать инструмент «Добавить вершину». После чего разместить вершину, следующую за его курсором в рабочем пространстве. Код создания вершины представлен в листинге 1 приложения А.

При нажатии на кнопку «Добавить вершину» происходит выполнение метода StartPlacingPoint(). В процессе выполнения создаётся шаблон вершины, которому присваивается переменная flyingPoint.

В методе Update происходит проверка переменной flyingPoint. Переменная flyingPoint, после выполнения StartPlacingPoint() имеет значение отличное от null, поэтому выполняется RayCastFlying().

Метод RayCastFlying() отвечает за следование вершины за курсором мыши.

При нажатии левой кнопки мыши, пользователь вызывает метод LBM_performed(). В методе происходит проверка переменной flyingPoint.

Переменная flyingPoint, после выполнения StartPlacingPoint() имеет значение отличное от null, поэтому выполняется PlaceFlyingPoint().

В ходе выполнения метода PlaceFlyingPoint() в лист кортежей pointsCoordinate добавляется новый кортеж, в который вносятся: идентификатор вершины, его координаты, имя и ссылка на объект. После чего список сортируется по идентификаторам. Общая переменная graphPointID инкрементируется, чтобы при создании следующей вершины последняя имела индивидуальный идентификатор. Переменной flyingPoint присваивается значение null. Метод StartPlacingPoint() повторно вызывается для создания следующей вершины.

Процесс создания вершин представлен на рисунке 13.

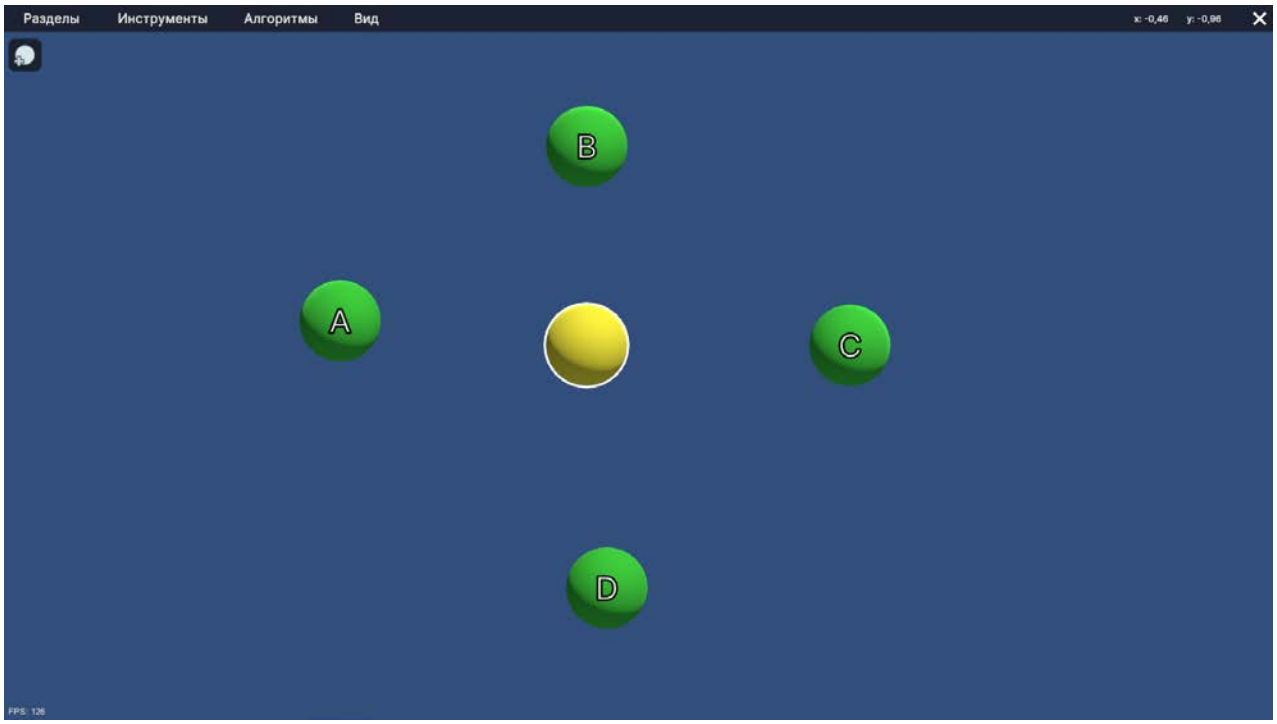

Рисунок 13 – Процесс создания вершин

Для создания рёбер пользователю необходимо, используя выпадающее меню «Инструменты», выбрать инструмент «Добавить ребро». После чего выбрать начальную и конечную вершины. Код создания ребра представлен в листинге 2 приложения А.

При нажатии на кнопку «Добавить ребро» происходит выполнение метода StartPlacingLine(). В процессе выполнения создаётся шаблон ребра, которому присваивается переменная flyingLine.

В методе Update происходит проверка переменной flyingLine. Переменная flyingLine, после выполнения StartPlacingLine() имеет значение отличное от null, поэтому выполняется RayCastFlying().

Метод RayCastFlying() отвечает за следование ребра за курсором мыши.

При нажатии левой кнопки мыши, пользователь вызывает метод LBM_performed(). В методе происходит проверка переменной flyingLine. Переменная flyingLine, после выполнения StartPlacingLine() имеет значение отличное от null, поэтому выполняется PlaceFlyingLine().

В ходе выполнения метода PlaceFlyingLine() в лист кортежей linesCoordinate добавляется новый кортеж, в который вносятся: идентификатор ребра, ссылка на начальную вершину, имя начальной вершины, ссылка на конечную вершину, имя конечной вершины, имя линии и ссылка на объект. После чего список сортируется по идентификаторам. Общая переменная graphLineID инкрементируется, чтобы при создании следующей вершины последняя имела индивидуальный идентификатор. Переменной flyingLine присваивается значение null. Метод PlaceFlyingLine() повторно вызывается для создания следующего ребра.

Процесс создания рёбер представлен на рисунке 14.

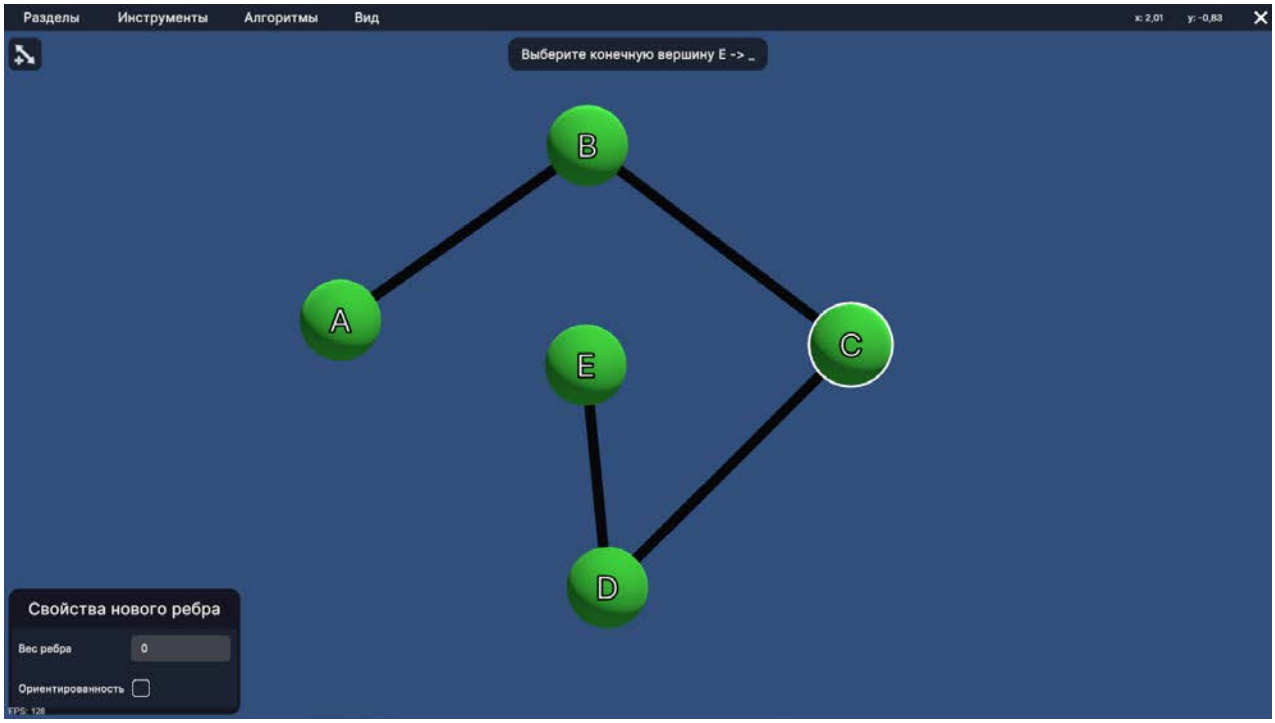

Рисунок 14 – Процесс создания рёбер

4.3 Работа алгоритма Дейкстры

Интерфейс работы с алгоритмами находится на сцене GraphTheoryProg. Для его вызова необходимо навести мышку на выпадающее меню «Алгоритмы», находящееся на верхнем горизонтальном меню, и выбрать конкретный алгоритм. Также необходимым условием является наличие на сцене хотя бы одной вершины. На рисунке 15 представлен интерфейс работы с алгоритмом Дейкстры.

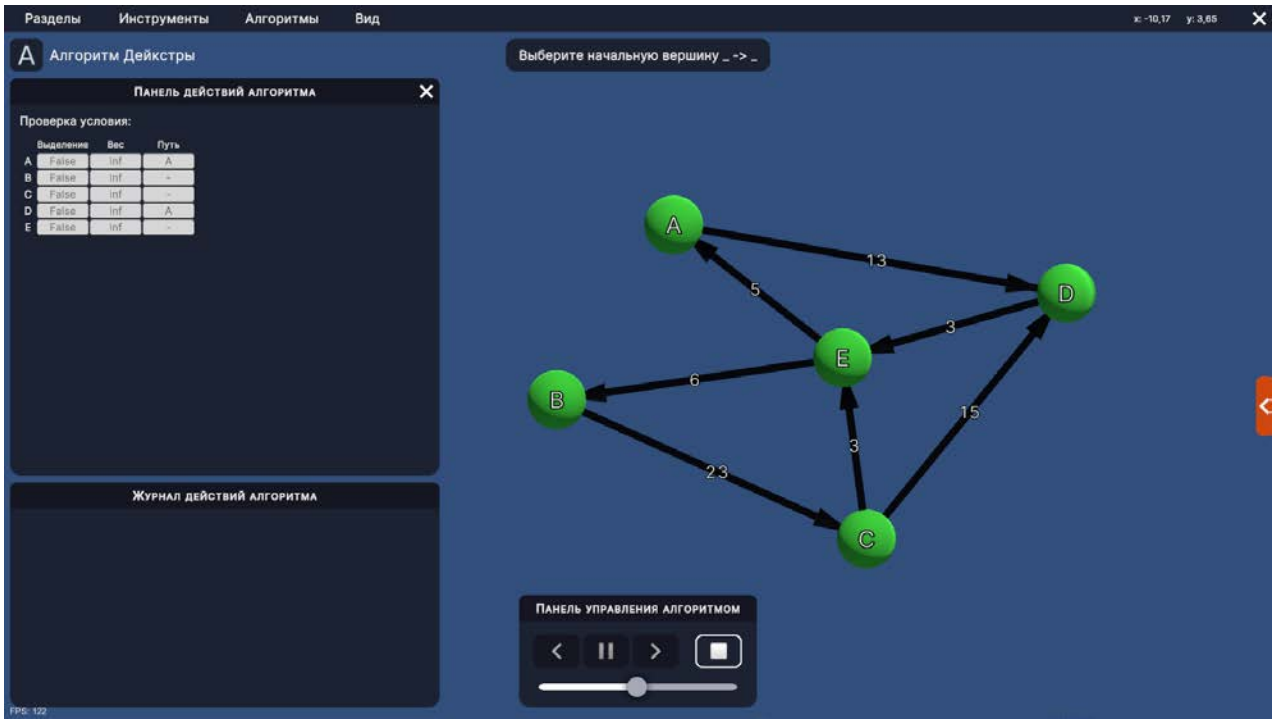

Рисунок 15 – Интерфейс работы с алгоритмом Дейкстры

Алгоритм Дейкстры - это алгоритм нахождения кратчайшего пути во взвешенном графе от одной вершины до всех остальных. Алгоритм был придуман нидерландским учёным Эдсгером Вибе Дейкстрой в 1959 году. Он работает следующим образом:

1. Создаётся список вершин, которые еще не были посещены, и устанавливается для каждой вершины бесконечное расстояние от начальной вершины.

2. Устанавливается расстояние от начальной вершины до самой себя равным нулю.

3. Находится вершина с наименьшим расстоянием из списка непосещённых вершин и помечается ее как посещенную.

4. Для каждой соседней вершины, которая еще не была посещена, вычисляется расстояние от начальной вершины через текущую вершину и сравнивается с текущим расстоянием до этой вершины. Если новое расстояние меньше, то значение расстояния обновляется.

5. Повторяются шаги 3 и 4, пока все вершины не будут посещены.

6. После завершения алгоритма, расстояние до каждой вершины будет минимальным возможным.

Алгоритм Дейкстры не применим к графам, имеющим отрицательный вес рёбер. Код работы алгоритма Дейкстры представлен в листинге 3 приложения Б.

4.4 Тестирование приложения

Для тестирования приложения был использован метод функционального тестирования.

Функциональное тестирование рассматривает заранее указанное поведение и основывается на анализе спецификации компонента или системы в целом, т.е. проверяется корректность работы функциональности приложения [19].

В таблице 3 показано тестирование симулятора.

Таблица 3 – Тестирование

№	Назначение	Действие	Ожидаемый результат	Исходный результат	Итоги
1	Проверка на корректное отображение текста, формул, и изображений теоретического блока	Используя кнопки переходов по разделам и подразделам проверить весь текст, формулы и изображения теоретического блока	Весь текст, формулы и изображения теоретического блока будет корректно отображаться	Весь текст, формулы и изображения теоретического блока корректно отображается	Пройдено
2	Проверка на корректное создание элементов графа	Используя инструменты редактора создать вершины и рёбра графа	Граф будет создан в соответствии с действиями пользователя	Граф был создан в соответствии с действиями пользователя	Пройдено
3	Проверка на корректное изменение свойств элементов графа	Используя панель изменения свойств изменить свойства вершин и рёбер графа	Свойства вершин и рёбер графа будут изменены в соответствии с внесёнными правками	Свойства вершин и рёбер графа были изменены в соответствии с внесёнными правками	Пройдено

Продолжение таблицы 3

4	Проверка на корректную визуализацию работы алгоритмов	Используя панель выбора алгоритмов проверить работу всех алгоритмов	Все алгоритмы будут визуализированы пошагово и выдадут корректный результат работы	Все алгоритмы были визуализированы пошагово и выдали корректный результат работы	Пройдено
5	Проверка на корректное переключение режима отображения графа/карты	Используя кнопку режима отображения проверить корректность отображения графа/карты	Граф/карта корректно будет отображаться во всех режимах камеры, а функционал будет соответствовать режимам	Граф/карта корректно отображаются во всех режимах камеры, а функционал соответствует режимам	Пройдено
6	Проверка на корректное изменение рейтинга дорог при изменении параметров дороги и смежных перекрёстков	Используя панель изменения параметров дороги и перекрёстков изменить параметры	Рейтинг дороги будет меняться в соответствии с внесёнными правками по исходной формуле	Рейтинг дороги поменялся в соответствии с внесёнными правками по исходной формуле	Пройдено
7	Проверка на корректное выполнение алгоритмов раскраски графа на карте	Используя панель теории о четырёх красках проверить работу алгоритмов	Карта будет раскрашена в соответствии с выбранным алгоритмом	Карта была раскрашена в соответствии с выбранным алгоритмом	Пройдено
8	Проверка корректного закрашивания карты с условием	Используя панель раскраски перекрасить карту с условием	Карта будет раскрашиваться если выполнено условие	Карта раскрашивается при выполнении условия	Пройдено
9	Проверка на корректную работу главного меню	Перейти в разные блоки программы	Будет произведён переход на соответствующий нажатой кнопке блок программы	Был произведён переход на соответствующий нажатой кнопке блок программы	Пройдено

Тестирование программы показало, что проект соответствует функциональным требованиям, выделенным в подзаголовке 3.1.

ЗАКЛЮЧЕНИЕ

В рамках выполнения выпускной квалификационной работы был разработан симулятор для изучения теории графов на платформе Unity.

Для достижения поставленной цели были выполнены следующие задачи:

5. Проведён обзор и анализ программного обеспечения для создания графов.
6. Проведён анализ и выбор средств реализации проекта.
7. Проведено архитектурное проектирование приложения.
8. Проведены программная реализация и тестирование проекта.

В ходе реализации проекта были приобретены навыки работы со средой разработки компьютерных игр Unity. Создание скриптов повысило навыки программирования на языке C#.

После полного завершения проекта предполагается его использование на кафедре ЭВМ.

БИБЛИОГРАФИЧЕСКИЙ СПИСОК

1 Гурин Н. И., Сахонь Е. С. Технология разработки симуляторов лабораторных установок для дистанционного обучения // Труды БГТУ. Сер. 3, Физико-математические науки и информатика. 2021. № 1 (242). С. 48–53.

2 Тренажеры в виртуальной реальности [Электронный ресурс] // Режим доступа: https://likevr.ru/blog/trenazhery-v-virtualnoy-realnosti (дата обращения: 10.05.2023).

3 Официальный сайт Graphviz [Электронный ресурс] // Режим доступа: https://graphviz.org (дата обращения: 21.02.2023).

4 Официальный сайт yEd Graph Editor [Электронный ресурс] // Режим доступа: https://www.yworks.com (дата обращения: 21.02.2023).

5 yEd Graph Editor [Электронный ресурс] // Режим доступа: https://media.contented.ru/glossary/yed-graph-editor (дата обращения: 22.02.2023).

6 Работа с графами онлайн / [Электронный ресурс] // Режим доступа: https://graphonline.ru (дата обращения: 21.02.2023).

7 Создание графа [Электронный ресурс] // Режим доступа: https://www.semestr.online/graph/graph.php (дата обращения: 22.02.2023).

8 ProgramForYou [Электронный ресурс] // Режим доступа: https://programforyou.ru/graph-redactor (дата обращения: 22.02.2023).

9 Graphiz Online [Электронный ресурс] // Режим доступа: https://dreampuf.github.io/GraphvizOnline (дата обращения: 22.02.2023).

10 Unreal Engine [Электронный ресурс] // Режим доступа: https://blog.skillfactory.ru/glossary/unreal-engine (дата обращения: 22.02.2023).

11 Hardware and Software Specifications [Электронный ресурс] // Режим доступа: https://docs.unrealengine.com/5.0/en-US/hardware-and-software-specifications-for-unreal-engine (дата обращения: 22.02.2023).

12 Хокинг, Д.Р. Unity в действии. Мультиплатформенная разработка на C# / Пер. с англ. И. Рузмайкиной. — СПб.: Питер, 2016. — 20 с.

13 Unity Manual. System requirements for Unity 2021 LTS [Электронный ресурс] // Режим доступа: https://docs.unity3d.com/Manual/system-requirements.html (дата обращения: 22.02.2023).

14 Unity Store. Plans and pricing [Электронный ресурс] // Режим доступа: https://store.unity.com/front-page?check_logged_in=1#plans-enterprise (дата обращения: 22.02.2023).

15 Серьёзные ошибки в коде CryEngine V [Электронный ресурс] // Режим доступа: https://habr.com/ru/company/pvs-studio/blog/325600 (дата обращения: 22.02.2023).

16 Игровой движок CryEngine от компании Crytek [Электронный ресурс] // Режим доступа: https://almaguz.ru/igrovoy-dvizhok-cryengine-ot-kompanii-crytek (дата обращения: 23.02.2023).

17 CryEngine. System requirements [Электронный ресурс] // Режим доступа: https://www.cryengine.com/support/view/system-requirements (дата обращения: 23.02.2023).

18 Визуальная студия [Электронный ресурс] // Режим доступа: https://wikipredia.net/ru/Visual_studio (дата обращения: 23.02.2023).

19 Теория тестирования ПО просто и понятно [Электронный ресурс] // Режим доступа: https://habr.com/ru/articles/587620/ (дата обращения: 10.04.2023)

ПРИЛОЖЕНИЯ

ПРИЛОЖЕНИЕ А

Исходный код создания вершин и рёбер графа

Листинг 1 – GraphController

```
using System;
using System.Linq;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.InputSystem;
public class GraphController: MonoBehaviour
{
    public static GraphController _graphController;
    public static GraphPoint flyingPoint;
    public static GraphLine flyingLinePoint;
    public static List<Tuple<int, float, float, float, string, Transform>> pointsCoordinate = new List<Tuple<int, float, float, float, string, Transform>>();
    public static List<Tuple<int, Transform, string, Transform, string, string, Transform>> linesCoordinate = new List<Tuple<int, Transform, string, Transform, string, string, Transform>>();
    public static int graphPointID = 0;
    public static int graphLineID = 0;
    public static GameObject workingSpace_points;
    public static GameObject workingSpace_lines;
    [SerializeField] private GraphPoint pointPrefab;
    [SerializeField] private GraphLine linePrefab;
    private CameraInputs inputs;

    private void OnEnable()
    {
        inputs.Enable();
        inputs.Mouse.LBM.performed += LBM_performed;
    }

    private void OnDisable()
    {
        inputs.Mouse.LBM.performed -= LBM_performed;
        inputs.Disable();
    }

    private void Awake()
```

```
        {
            inputs = new CameraInputs();
            _graphController = this;
        }

        private void Update()
        {
            if(flyingPoint != null || flyingLinePoint != null)
                RayCastFlying();
        }

        private void LBM_performed(InputAction.CallbackContext obj)
        {
            if(flyingPoint != null)
                PlaceFlyingPoint();
        }

        private void RayCastFlying()
        {
            var groupPlane = new Plane(new Vector3(0, 0, 1f), Vector3.zero);
            Ray ray = Camera.main.ScreenPointToRay(Input.mousePosition);
            if(groupPlane.Raycast(ray, out float position))
            {
                Vector3 worldPosition = ray.GetPoint(position);
                if(flyingPoint != null)
                {
                    flyingPoint.transform.position  =  new  Vector3(worldPosition.x,
worldPosition.y, 0);

flyingPoint.SetTransparent(flyingPoint.GetComponentInChildren<GraphPointCollider>().
repositionAvailable);
                }
                if(flyingLinePoint != null)
                    flyingLinePoint.transform.position = new Vector3(worldPosition.x,
worldPosition.y, 0);
            }
        }

        public static void StartPlacingPoint(GraphPoint pointPrefab)
        {
```

```
            flyingPoint = Instantiate(pointPrefab, new Vector3(Input.mousePosition.x,
Input.mousePosition.y, 0), Quaternion.identity);
            flyingPoint.transform.parent = workingSpace_points.transform;
        }

        private void PlaceFlyingPoint()
        {
            pointsCoordinate.Add(new Tuple<int, float, float, float, string,
Transform>(graphPointID, flyingPoint.transform.position.x,
flyingPoint.transform.position.y, flyingPoint.transform.position.z, flyingPoint.name,
flyingPoint.transform));
            pointsCoordinate.OrderBy(x => x.Item1).ToList();
            flyingPoint.ID = graphPointID;
            graphPointID++;
            flyingPoint = null;
            StartPlacingPoint(pointPrefab);
        }

        public static void StartPlacingLine()
        {
            flyingLinePoint = Instantiate(_graphController.linePrefab);
            flyingLinePoint.transform.parent = workingSpace_lines.transform;
        }

        public static void PlaceFlyingLine(GameObject firstPoint, GameObject
secondPoint)
        {
            linesCoordinate.Add(new Tuple<int, Transform, string, Transform, string,
string,      Transform>(graphLineID,      firstPoint.transform,      firstPoint.name,
secondPoint.transform,            secondPoint.name,            flyingLinePoint.name,
flyingLinePoint.transform));
            linesCoordinate.OrderBy(x => x.Item1).ToList();
            flyingLinePoint.transform.position = Vector3.zero;
            flyingLinePoint.SetPoints();
            flyingLinePoint.ID = graphLineID;
            graphLineID++;
            flyingLinePoint = null;
        }
    }
```

Листинг 2 – GraphPointCollider

```
using System;
using System.Linq;
using UnityEngine;
public class GraphPointCollider: MonoBehaviour
{
    [NonSerialized] public GameObject firstPoint;

    private void OnMouseOver()
    {
        if(GraphController.flyingLinePoint != null)
        {
            bool available = true;
            if(available)
                available = false;
            if(available && Input.GetMouseButtonDown(0))
            {
                GraphController.PlaceFlyingLine(firstPoint,
transform.parent.gameObject);
            }
        }
        else if(Input.GetMouseButtonDown(0) && GraphController.flyingLinePoint ==
null)
        {
            GraphController.StartPlacingLine();
            firstPoint = transform.parent.gameObject;
        }
    }
}
```

ПРИЛОЖЕНИЕ Б

Исходный код работы алгоритма Дейкстры

Листинг 3 – AlgorithmDijkstra

```
using System;
using UnityEngine;
using UnityEngine.InputSystem;
public class AlgorithmDijkstra : MonoBehaviour
{
    [NonSerialized] public GameObject startPoint;
    [NonSerialized] public GameObject endPoint;
    private Transform[] pathTable;
    private int[,] graph;
    private int[] distance;
    private bool[] shortestPathTreeSet;
    private int verticesCount;
    private int startIndex;
    private int endIndex;
    private CameraInputs inputs;

    private void OnEnable()
    {
        inputs.Enable();
        inputs.Mouse.LBM.performed += LBM_performed;
        startPoint = null;
        endPoint = null;
    }

    private void OnDisable()
    {
        inputs.Mouse.LBM.performed -= LBM_performed;
        inputs.Disable();
    }

    private void Awake()
    {
        inputs = new CameraInputs();
    }
    private void LBM_performed(InputAction.CallbackContext obj)
    {
        Ray _ray = Camera.main.ScreenPointToRay(Input.mousePosition);
```

```
            if(Physics.Raycast(_ray, out RaycastHit _raycastHit))
            {
                GameObject selectedObject;
                if(_raycastHit.collider.gameObject.tag == "Point")
                {
                    selectedObject                                                    =
_raycastHit.collider.transform.parent.gameObject;
                    AlgorithmMain(selectedObject);
                }
            }
        }

        private void AlgorithmMain(GameObject graphPoint)
        {
            if(startPoint == null)
                startPoint = graphPoint;
            else if(endPoint == null && startPoint != null)
            {
                endPoint = graphPoint;
                graph = ToMatrix._toMatrix.MatrixMain();
                verticesCount = graph.GetLength(0);
                startIndex = int.MaxValue;
                endIndex = int.MaxValue;
                GameObjectIntoInt(startPoint, endPoint);
                distance = new int[verticesCount];
                shortestPathTreeSet = new bool[verticesCount];
                for(int i = 0; i < verticesCount; ++i)
                for {
                    distance[i] = ToMatrix.noLine;
                    shortestPathTreeSet[i] = false;
                }
                distance[startIndex] = 0;
                shortWays = new Transform[verticesCount][];
                CreatePathTable();
                Dijkstra();
            }
        }
        private void Dijkstra()
        {
            for(int count = 0; count < verticesCount; ++count)
```

```
            {
                int u = MinimumDistance();
                shortestPathTreeSet[u] = true;
                for(int v = 0; v < verticesCount; ++v)
                {
                    if(!shortestPathTreeSet[v] && graph[u, v] < ToMatrix.noLine &&
distance[u] < ToMatrix.noLine && distance[u] + graph[u, v] < distance[v])
                    {
                        distance[v] = distance[u] + graph[u, v];
                        SetWay(u, v);
                    }
                }
            }
        }

        private int MinimumDistance()
        {
            int min = ToMatrix.noLine;
            int minIndex = 0;
            for(int v = 0; v < verticesCount; ++v)
            {
                if(shortestPathTreeSet[v] == false && distance[v] <= min){
                    min = distance[v];
                    minIndex = v;
                }
            }
            return minIndex;
        }

        private void SetWay(int u, int v)
        {
            pathTable[v] = ToMatrix.pointAllTransformsMatrix[u, u][0];
            int tmpCount = 50;
            Transform[] shortWays_tmp = new Transform[tmpCount];
            int n = 0;
            int _v = v;
            while(true)
            {
                if(pathTable[_v] == null) break;
                shortWays_tmp[n] = pathTable[_v];
```

```
                n++;
                if(ToMatrix.pointAllTransformsMatrix[startIndex,  startIndex][0]  ==
pathTable[_v]) break;
                bool stop = false;
                for(int y = 0; y < verticesCount; y++)
                {
                    if(stop) break;
                    for(int x = 0; x < verticesCount; x++)
                    {
                        if(stop) break;
                        if(ToMatrix.pointAllTransformsMatrix[y, x][0] == pathTable[u]
&& ToMatrix.pointAllTransformsMatrix[y, x][1] == pathTable[_v])
                        {
                            _v = x;
                            stop = true;
                        }
                    }
                }
            }
            shortWays[v] = new Transform[n+1];
            for(int i = tmpCount-1, j = 0; i >= 0; i--)
            {
                if(shortWays_tmp[i] != null){
                    shortWays[v][j] = shortWays_tmp[i];
                    j++;
                }
            }
            shortWays[v][n] = ToMatrix.pointTransformsMatrix[v, v][0];
        }
        private  void  GameObjectIntoInt(GameObject  startGraphPoint,  GameObject
endGraphPoint)
        {
            if(startGraphPoint != null && endGraphPoint != null)
            {
                for(int i = 0; i < verticesCount; i++)
                {
                    if(ToMatrix.pointAllTransformsMatrix[i,         0][0]         ==
startGraphPoint.transform)
                        startIndex = i;
```

```
                    if(ToMatrix.pointAllTransformsMatrix[i,          0][0]          ==
endGraphPoint.transform)
                        endIndex = i;
                    if(startIndex != int.MaxValue && endIndex != int.MaxValue) break;
                }
            }
        }
        private void CreatePathTable()
        {
            pathTable = new Transform[verticesCount];
            for(int y = 0; y < verticesCount; y++)
            {
                if(graph[startIndex, y] < ToMatrix.noLine)
                    pathTable[y] = ToMatrix.pointTransformsMatrix[startIndex, y][0];
            }
        }
    }
```