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 дистанционного выполнения лабораторных работ по электротехнике

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА
К ВЫПУСКНОЙ КВАЛИФИКАЦИОННОЙ РАБОТЕ
ЮУРГУ-090301.2023.026 ПЗ ВКР

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

Автор работы,
студент группы КЭ-405
_____С.В. Ильин
«___» _____2023 г.

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

Челябинск-2023

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»

УТВЕРЖДАЮ
Заведующий кафедрой,
_____ Д.В. Топольский
«___» _____ 2023 г.

ЗАДАНИЕ
на выпускную квалификационную работу бакалавра
студенту группы КЭ-405
Ильину Сергею Владимировичу
обучающемуся по направлению
09.03.01 «Информатика и вычислительная техника»

1. Тема работы: «Разработка симулятора для дистанционного выполнения лабораторных работ по электротехнике» утверждена приказом по университету от 25.04.2023 г. №753-13/12

2. Срок сдачи студентом законченной работы: 01 июня 2023 г.

3. Исходные данные к работе:

Стенд «Электрические цепи» учебно-исследовательского комплекса «Электротехника – Электромеханика – 1,5 кВт»:

- модуль питания (напряжения +5 В, +12 В, -12 В);
- модуль резисторов (переменные резисторы R1, R2, R3);
- модуль измерительный (цифровые амперметры);
- модуль аналоговых измерителей (вольтметр, амперметр);
- модуль «Измеритель мощности».

4. Перечень подлежащих разработке вопросов:

Выпускная квалификационная работа (ВКР) должна содержать разработку следующих вопросов:

1. Аналитический обзор аналогов;
2. Выбор средств разработки и определение требований;
3. Разработка пользовательского интерфейса симулятора;
4. Реализация расчёта силы тока и напряжения скоммутированной цепи;
5. Отображение полученных результатов на измерительных устройствах.

Дата выдачи задания: «2» декабря 2022 г.

Руководитель работы_____/ К.А. Домбровский /

Студент _____/ С.В. Ильин /

КАЛЕНДАРНЫЙ ПЛАН

Этап	Срок сдачи	Подпись руководителя
Аналитический обзор аналогов	01.03.2023	
Выбор средств разработки и определение требований	10.03.2023	
Разработка пользовательского интерфейса симулятора	30.03.2023	
Реализация расчёта силы тока и напряжения скоммутированной цепи	30.04.2023	
Отображение полученных результатов на измерительных устройствах	10.05.2023	
Компоновка текста работы и сдача на нормоконтроль	22.05.2023	
Подготовка презентации и доклада	30.05.2023	

Руководитель работы_____/ К.А. Домбровский /

Студент _____/ С.В. Ильин /

АННОТАЦИЯ

С.В. Ильин. Разработка симулятора для дистанционного выполнения лабораторных работ по электротехнике. – Челябинск: ФГАОУ ВО «ЮУрГУ (НИУ)», ВШ ЭКН; 2023, 50 с., 23 ил., библиогр. список - 20 наим.

В рамках данной выпускной квалификационной работы разрабатывается симулятор для дистанционного выполнения лабораторных работ по электротехнике. В ходе работы был проведён аналитический обзор аналогов с выявлением их достоинств и недостатков, были выбраны необходимые для разработки средства и определены функциональные и нефункциональные требования.

В соответствии с установленными требованиями, был разработан пользовательский интерфейс симулятора, система управления мышью, реализован расчёт силы тока и напряжения скоммутированной пользователем цепи и способ отображения показаний измерительных модулей на индикаторах разных типов.

Результатом работы является симулятор, позволяющий пользователю собирать и симулировать произвольную электрическую цепь из компонентов модулей стенда.

ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ

Проект является заказом кафедры, читающей дисциплину «Электротехника».

На данный момент времени разработка симуляторов лабораторного оборудования довольно актуальна, поскольку симуляция позволяет сократить расходы денег и материалов на производство и обслуживание аналогового оборудования, а также позволяет экономить и лучше распределять время для проведения лабораторных занятий, обеспечивая каждого студента своим личным стендом.

Помимо этого, виртуализация лабораторных работ упрощает учебный процесс для студентов, находящихся на заочном обучении, а также позволяет проводить лабораторные работы во время карантина.

1 АНАЛИТИЧЕСКИЙ ОБЗОР АНАЛОГОВ

Существующие на данный момент симуляторы электрических цепей, такие как Proteus, Jigrein, Spice Simulator и подобные им не соответствуют требованию заказчика, поскольку интерфейс этих программ не отражает внешний вид приборов стенда. Из-за этого студентам сложнее выполнять лабораторные работы, так как методические указания подразумевают работу с реальным стендом. Единственным аналогом является симулятор, разработанный компанией Лабсис [1].

Виртуальный лабораторный комплекс Лабсис имеет интерфейс, соответствующий внешнему виду реального учебного стенда «Электромеханика». Тренажёр обеспечивает выполнение лабораторных работ с цепями постоянного тока и позволяет студенту взаимодействовать с виртуальными приборами. Однако в построении цепей присутствует ограничение.

Данный симулятор даёт возможность запуска только тех цепей, которые присутствуют в методических указаниях и заранее заложены в код программы. Симуляция и расчёт произвольной цепи являются невозможными. Поэтому если появится необходимость расширить или изменить учебное пособие к лабораторным работам, то потребуется обновление программы.

Внешний вид виртуального аналога представлен на рисунке 1.1. Для отображения приборов используются трёхмерные модели. Это повышает минимальные требования к системе, на которой запускается симулятор. В связи с этим, студенты, обладающие маломощными компьютерами, могут испытывать трудности при работе.

Рисунок 1.1 – Виртуальный лабораторный комплекс

Основные преимущества:

– интерфейс соответствует внешнему виду реального стенда;
– встроенные методические указания.

Основные недостатки:

– невозможность симуляции произвольных цепей;
– сложность масштабирования;
– нагрузка графического процессора трёхмерной графикой;
– сложное управление.

2 ВЫБОР СРЕДСТВ РАЗРАБОТКИ И ОПРЕДЕЛЕНИЕ ТРЕБОВАНИЙ

2.1 Функциональные требования

– возможность коммутации клемм на стенде;

– расчёт силы тока, протекающего через заданный элемент цепи, и напряжения между двумя выбранными точками в созданной электрической цепи;

– результаты расчётов должны быть отображены на измерительных устройствах;

– управление положением камеры;

– изменение масштаба камеры для удобного снятия показаний с приборов;

– низкая нагрузка на ресурсы системы.

2.2 Нефункциональные требования

– внешний вид симулятора должен соответствовать реальному стенду;

– индикаторы стенда должны быть легко читаемы.

2.3 Выбор фреймворка для симуляции

Симуляция работы стенда подразумевает создание и расчёт электрических цепей с последующим выводом результатов. Для этого был выбран фреймворк SpiceSharp, основанный на Spice Simulator, разработанный в Калифорнийском университете Беркли.

SPICE является общепринятым стандартом в симуляции электрических схем, что делает его хорошим выбором [2]. В SpiceSharp были исправлены некоторые ошибки оригинальной библиотеки. Фреймворк имеет доступную документацию и обучающие материалы в достаточном количестве. Также библиотека имеет возможность симуляции цепей переменного тока [3].

2.4 Выбор среды разработки

Поскольку одним из требований заказчика является отображение в симуляторе модулей стенда, необходимо выбрать средство, позволяющее работать с графикой. Для этих задач существуют среды разработки компьютерных игр.

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. Данное программное обеспечение 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, соответствующую 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 для компьютерных игр обусловлен тем, что такое программное обеспечение облегчает работу с объектами и графикой, что существенно упрощает процесс разработки.

В данной отрасли существует огромное множество сред разработки, 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 программ являются узконаправленными и предназначены только для создания видеоигр. Также некоторые среды 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 используемые среды 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 программа для создания игр. Изначально разработанная для компьютерных игр от первого лица, с тех пор она использовалась в различных играх и была принятя другими отраслями, в первую очередь киноиндустрией и телевизионной индустрией. В ней можно работать с персонажами, логикой, физикой и графикой игры.

UE разработала компания Epic Games для своей игры под названием Unreal, и после этого программа стала популярна. Её основное отличие — хорошая оптимизация: UE создавался не как отдельный коммерческий продукт, а как рабочий инструмент, и ориентирован он на 3D-игры [4]. На рисунке 2.1 показан интерфейс Unreal Engine 5.

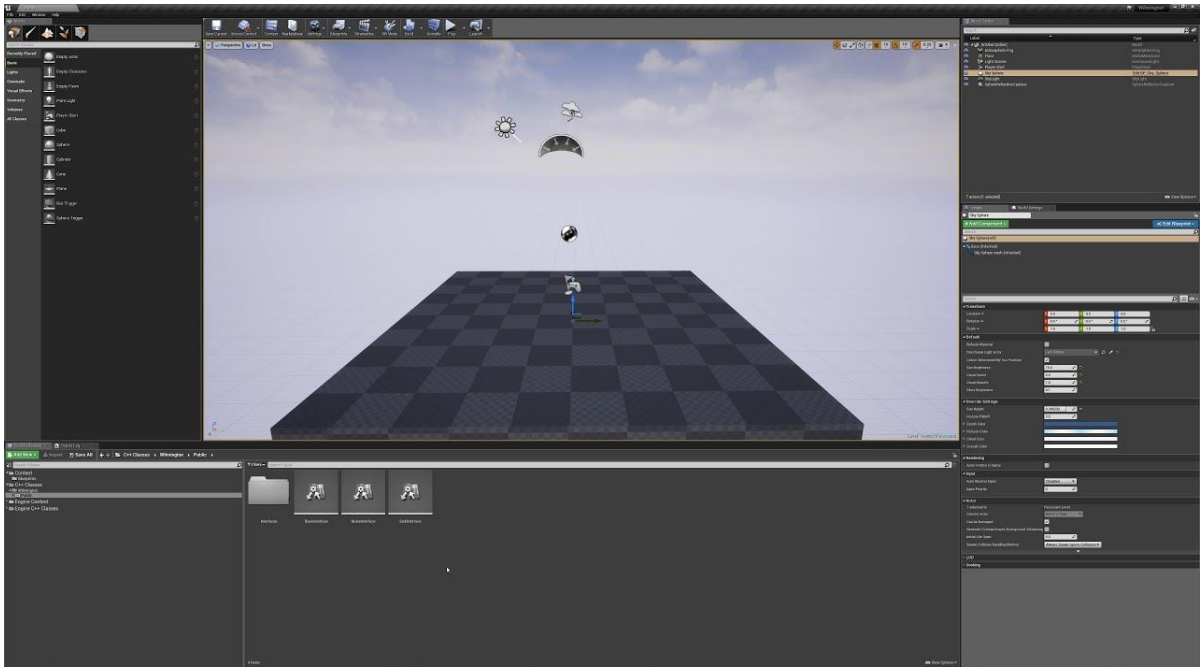

Рисунок 2.1 – Интерфейс Unreal Engine

Достоинства:

– большое сообщество;
– высокая кроссплатформенность;
– широкий ассортимент инструментов для различных целей;
– великолепное графическое решение.

Недостатки:

– высокие минимальные требования к ПК разработчика [5];
– высокий порог вхождения;
– малый выбор готового инструментария в официальном магазине.

Unity — среда разработки компьютерных игр, разрабатываемая и поддерживаемая компанией Unity Technologies. Программа 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 [6]. На рисунке 2.2 показан интерфейс Unity.

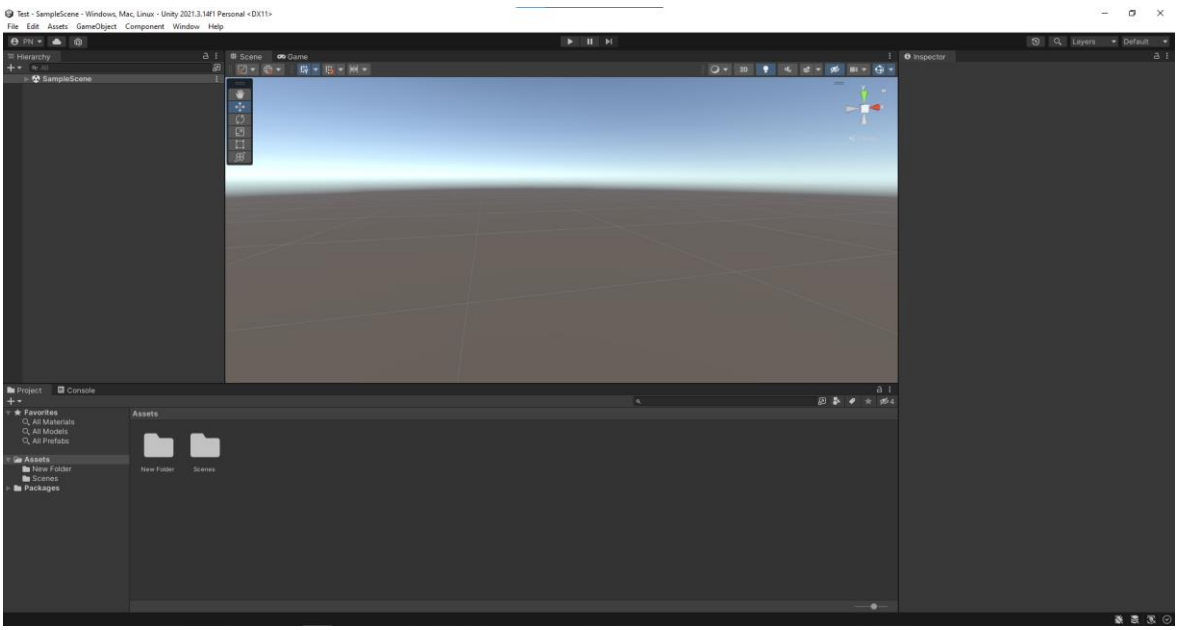

Рисунок 2.2 – Интерфейс Unity 2021

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 для разработки [7];
– большой выбор коммерческих лицензий с фиксированной оплатой [8].

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 распространённая программа CryEngine.

CryEngine – программа, созданная немецкой компанией Crytek в 2002 году и первоначально используемая в игре от первого лица Far Cry [9].

Первоначально она была полностью платной, а бесплатная версия была доступна только обучающим учреждениям для обучения их учеников и студентов. Но в 2016-м году стратегия распространения программы поменялась. Теперь она и средства разработки игр от Crytek доступны абсолютно всем бесплатно до достижения доходов разработчиков с программного продукта свыше 5000$ [10]. На рисунке 2.3 показан интерфейс CryEngine 5.

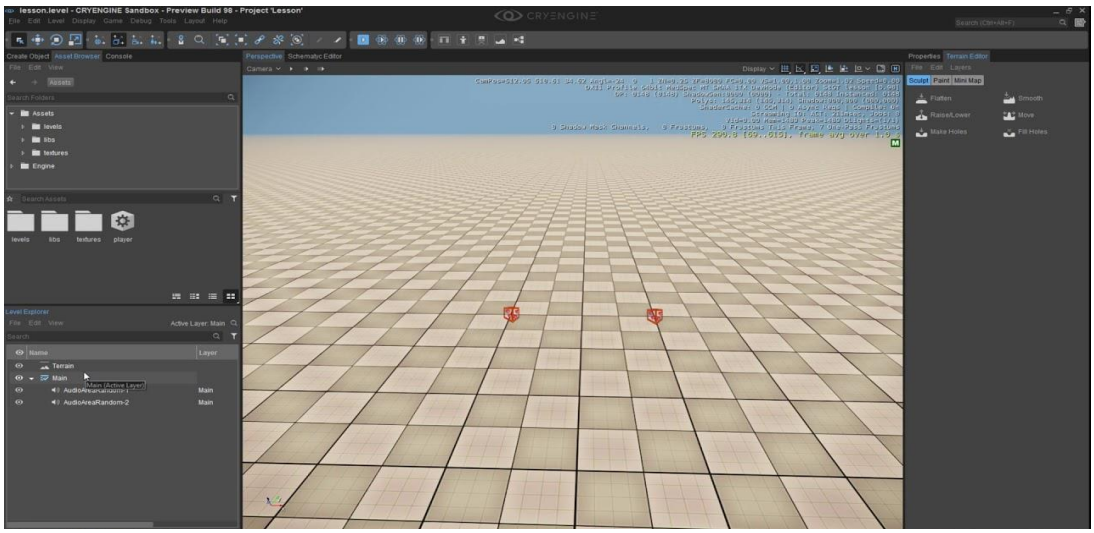

Рисунок 2.3 – Интерфейс CryEngine 5

Достоинства:

– многие инструменты доступны сразу после установки софта;
– программирование на языке C#;
– низкие системные требования [11];
– 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 игровой среда разработки Unity.

3 РАЗРАБОТКА ПОЛЬЗОВАТЕЛЬСКОГО ИНТЕРФЕЙСА СИМУЛЯТОРА

Для того, чтобы выполнить установленным требованиям, необходимо было создать изображения модулей и компонентов стенда с последующим отображением их в двумерном пространстве, а также реализовать систему взаимодействия пользователя с симулятором посредством устройств ввода.

3.1 Создание моделей

Для создания моделей были сделаны подробные фотографии лабораторного стенда. С каждого компонента стенда были взяты размеры. Фотография стенда представлена на рисунке 3.1.

Рисунок 3.1 – Стенд «Электротехника-Электромеханика»

На основе полученных фотографий в редакторе векторной графики были созданы цифровые изображения модулей стенда. Также был создан дополнительный модуль, включающий себя стрелочные амперметры и вольтметры. Помимо этого, элементы стенда, с которыми взаимодействует пользователь, были отдельно экспортированы для использования их в отдельных объектах внутри Unity.

Перечень созданных модулей стенда:

1. Модуль питания (рисунок 3.2);

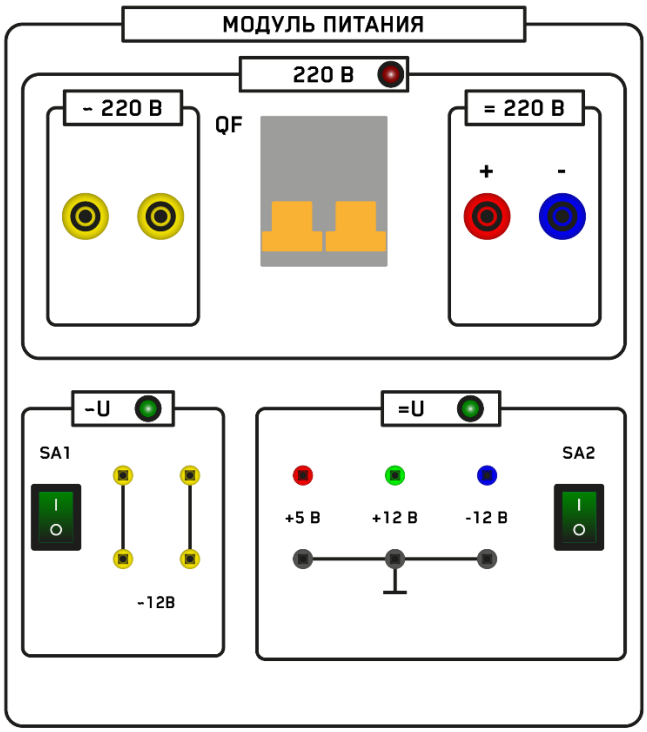

Рисунок 3.2 – Модуль питания

Содержит в себе общий выключатель, генераторы переменного и постоянного тока разного напряжения, клеммы для подключения проводов.

2. Модуль цифровых амперметров (рисунок 3.3);

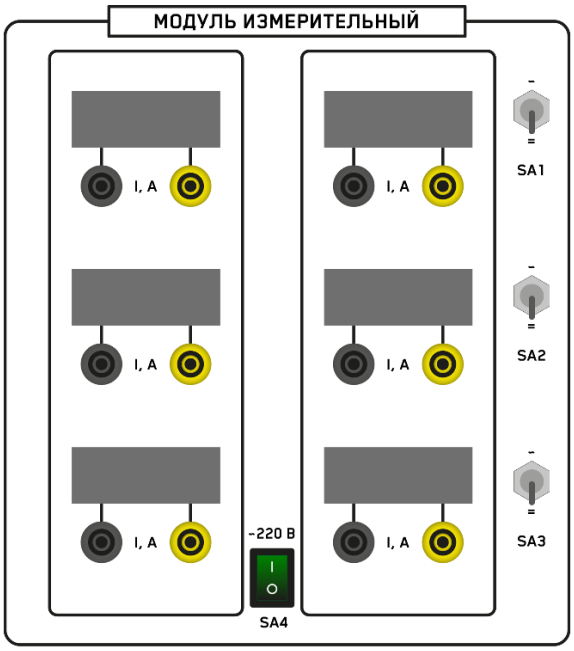

Рисунок 3.3 – Модель цифровых амперметров

Содержит шесть амперметров, каждый из которых имеет положительный и отрицательный порт и линейный цифровой дисплей.

3. Измеритель мощности (рисунок 3.4);

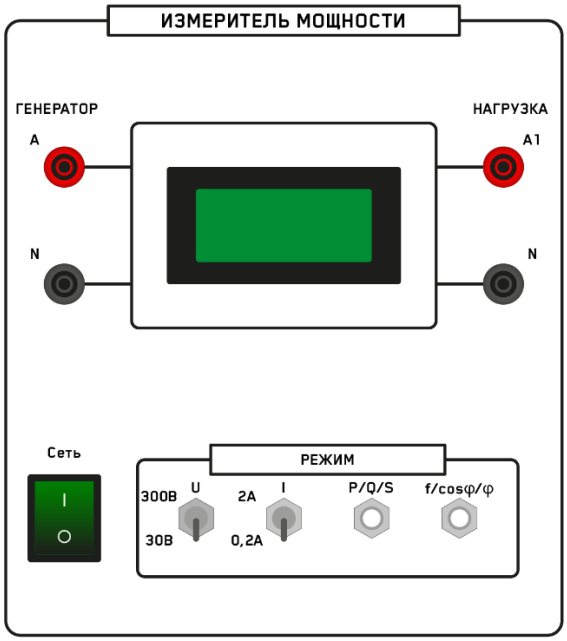

Рисунок 3.4 - Измеритель мощности

Включает в себя 4 разъёма Ваттметра, матричный дисплей, переключатели пределов измерения и параметров отображения.

4. Модуль резисторов (рисунок 3.5);

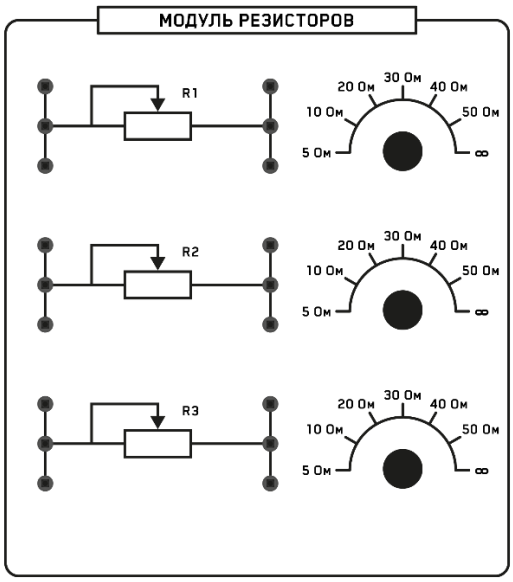

Рисунок 3.5 – Модуль резисторов

Имеет три переменных резистора, ручку для выбора номинала и клеммы.

5. Модуль аналоговых измерителей (рисунок 3.6);

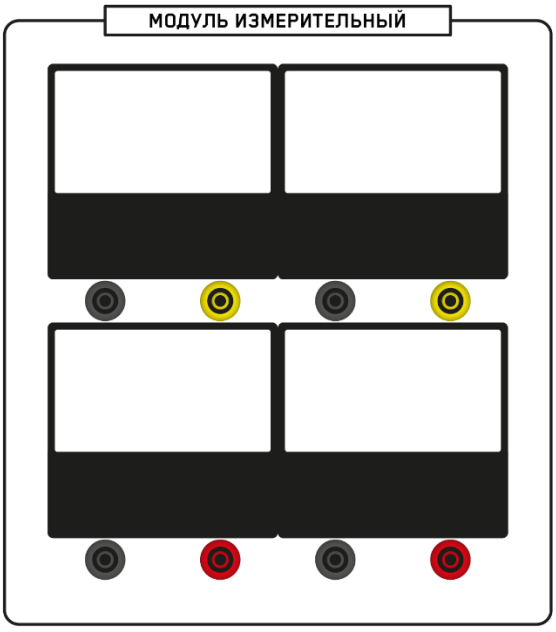

Рисунок 3.6 – Модуль аналоговых измерителей

Содержит два стрелочных амперметра и два стрелочных вольтметра.

6. Модуль мультиметров (рисунок 3.7).

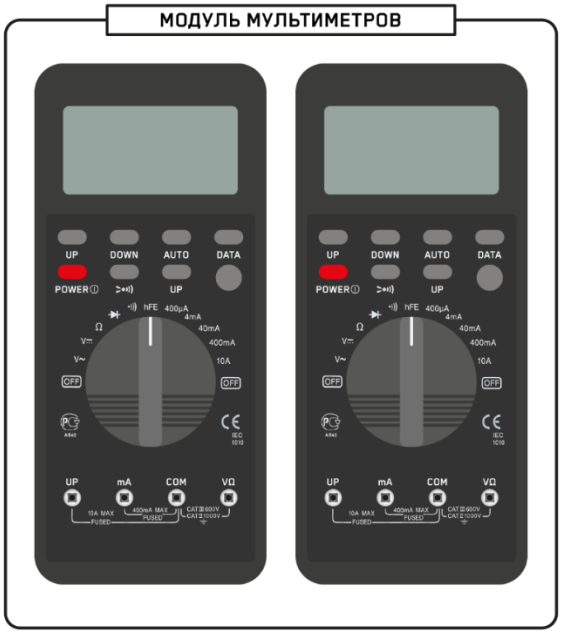

Рисунок 3.7 – Модуль мультиметров

Созданные изображения были загружены в Unity и выставлены в два ряда. Окончательный внешний вид симулятора представлен на рисунке 3.8.

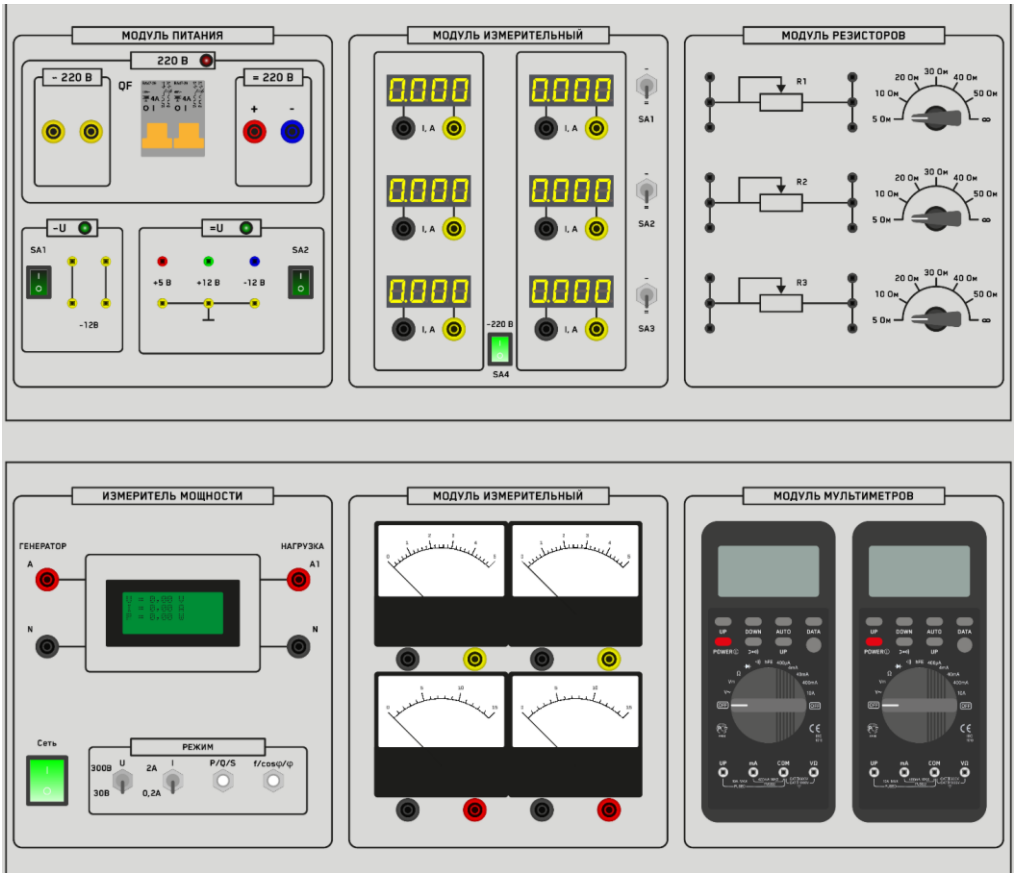

Рисунок 3.8 – Внешний вид симулятора

3.2 Система ввода

Система ввода действий пользователя реализована при помощи подключаемого компонента Unity.InputSystem [12]. Функционал это компонента позволяет гибко настроить управление симулятором при помощи разных устройств ввода. За работу системы ввода отвечает класс «StandInputSystem.cs».

В проекте созданы две отдельных карты управления:

- управление камерой;
- управление стендом.

Разделение подобным образом позволяет избежать конфликтов между двумя картами управления в случае привязки разных действий к одинаковым клавишам.

На основе созданных карт автоматически генерируется класс, содержащий все действия ввода. Каждое нажатие клавиши вызывает событие, на которое можно подписать любую функцию. Это даёт возможность управлять поведением системы управления через код, предоставляя более точные настройки.

3.3 Коммутация приборных проводов

Для отображения проводов было решено использовать кривые Безье. Для реализации их в симуляторе был использован компонент UnityEngine.Splines [13]. Этот компонент имеет реализацию расчёта кривой по заданным точкам. Для отрисовки кривых используется компонент UnityEngine.LineRenderer [14]. При помощи данного компонента массив точек кривой преобразуется в линию на экране. Есть возможность выбора толщины, цвета, прозрачности и гладкости линии.

Для управления поведением провода был реализован скрипт «Cable.cs». Данный скрипт позволяет управлять величиной провисания провода, сглаживанием краёв линии и сглаживанием перемещения точек провода по времени. Для создания и уделения проводов в сцене написан класс «CableContainer.cs». Каждому созданному проводу присваивается уникальное имя. Для коммутации проводов устройством ввода, необходимо взаимодействовать с клеммами модуля.

Список параметров провода показан на рисунке 3.9.

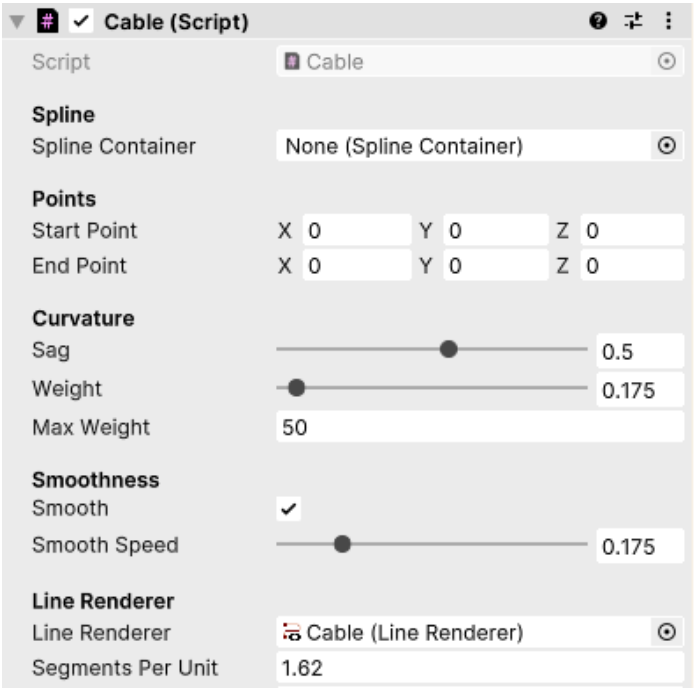

Рисунок 3.9 – Параметры скрипта Cable.cs

На рисунке 3.10 представлен внешний вид провода с указанными параметрами.

Рисунок 3.10 – Провод

3.4 Управление камерой

Поскольку стенд имеет множество мелких элементов, для удобства использования было реализовано управление камерой. Увеличение или уменьшение масштаба производится вращением колеса мыши, а изменение положения камеры производится перемещением мыши с зажатым колесом.

За поведение камеры отвечает класс «UICamera.cs». Перемещение камеры ограничено размерами стенда. Также при масштабировании учитывается положение курсора мыши, что значительно упрощает навигацию и делает интерфейс симулятора более удобным. Помимо этого, окно симулятора можно менять в любой время. При изменении размеров окна, камера автоматически масштабируется и подгоняется под новое соотношение сторон.

3.5 Взаимодействие со стендом

В классе «UICamera.cs» реализованы методы для определения находящихся под курсором приборов стенда при помощи функции Camera.ScreenPointToRay [15]. Для того, чтобы объект реагировал на действия мыши, ему нужно присвоить компонент BoxCollider2D [16]. Если под курсором находится интерактивный объект, то ему будет передана информация о совершённом действии. К интерактивным элементам стенда относятся:

- переключатели Вкл/Выкл (класс «Switch.cs»);
- галетный переключатель (класс «Knob.cs»);
- клеммы (класс «Port.cs»);

Каждый из вышеперечисленных классов наследуется от абстрактного класса «Control.cs». Таким образом, каждый интерактивный объект может реагировать на нажатие и отпускание обеих клавиш мыши, на прокрутку колеса и перемещение курсора с зажатой клавишей.

Переключатель Вкл/Выкл содержит свойство, обозначающие положение переключателя, где true – включено, false – выключено. Изменение положения производится щелчком левой кнопкой мыши.

Галетный переключатель имеет заранее заданный в среде разработки массив значений. Переключение производится зажатием левой клавиши мыши над ручкой и перетаскиванием курсора в сторону.

Клеммы предназначены для коммутации цепи проводами. Для того, чтобы собрать схему, необходимо нажать левой клавишей мыши на разъём необходимого модуля и не отпуская перевести курсор на другой разъём. В результате будет создан провод, соединяющий элементы цепи. Все созданные провода содержатся в объекте CableContainer.

4 РЕАЛИЗАЦИЯ РАСЧЁТА СИЛЫ ТОКА И НАПРЯЖЕНИЯ СКОММУТИРОВАННОЙ ЦЕПИ

4.1 Работа с библиотекой SpiceSharp

Для реализации симуляции был выбран фреймворк SpiceSharp [3]. Для создания цепей внутри библиотеки создаётся объект Circuit, в который добавляются необходимые для симуляции элементы. Для симуляции цепи постоянного тока потребуются классы SpiceSharp.Components.Resistor и SpiceSharp.Components.VoltageSource. Каждый компонент цепи должен иметь своё уникальное имя, а для соединения двух элементов между собой необходимо назначить уникальное строковое значение на отрицательный контакт первого элемента и положительный контакт другого.

Для симуляции цепи постоянного тока создаётся объект SpiceSharp.Simulations.DC, в параметрах которого указывается название симуляции, источник напряжения, начальное и конечное значение напряжения и шаг симуляции.

Для вывода данных симуляции необходимо создать объект экспорта RealPropertyExport из класса SpiceSharp.Simulations. Класс принимает экземпляр симуляции DC, название элемента цепи и название свойства, которое необходимо вывести. Для вывода силы тока используется свойство «i», а для напряжения «v».

Для вывода разности потенциалов между двумя точками цепи используется класс RealVoltageExport. В данном случае, в конструктор класса передаются названия точек соединений.

После создания экземпляров экспорта, необходимо подписать отвечающую за экспорт функцию к событию DC.ExportSimulationData. При запуске симуляции, все подписанные функции к этому событию будут выполнены. Присвоение переменным результатов симуляции производится в теле подписанной функции.

Таким образом можно осуществлять симуляцию простых цепей с последовательными и параллельными соединениями [17].

4.2 Преобразование скоммутированной пользователем цепи в объекты библиотеки

Для внедрения SpiceSharp в проект был создан класс «CircuitContainer.cs». Для преобразования собранной на экране схемы в вид, принимаемый библиотекой SpiceSharp, был разработан принцип создания и присвоения строковых названий для отрицательных и положительных контактов стенда. Уникальные названия каждого компонента указывается в среде Unity и хранится в имени объекта.

Соединение компонентов происходит через клеммы, которые реализованы в классе «Port.cs». Каждой клемме в поле m_component присваивается компонент модуля стенда. Поле bool isPositiveNode указывает, является ли клемма положительным контактом элемента. При запуске симулятора каждой клемма присваивается имя, состоящее из имени присвоенного компонента и строки “Pos”, если клемма подключена к положительному контакту, и “Neg”, если контакт отрицательный.

Класс каждого прибора стенда наследуется от абстрактного класса CircuitComponent. Абстрактный класс подразумевает, что каждый симулируемый элемент цепи должен хранить названия положительного и отрицательного контактов, количество соединений, а также иметь реализованные методы ProvideDC и ProvideCkt.

Метод ProvideDC необходим для компонентов, которые экспортируют данные о симуляции или являются источниками напряжения. Метод вызывается классом симуляции CircuitContainer, который передаёт в метод экземпляр объекта DC. В метод ProvideCkt передаётся контейнер всех элементов SpiceSharp.Circuit, после чего вызываемый объект добавляет в контейнер элемент цепи со всеми параметрами.

Когда пользователь соединяет проводом две клеммы стенда, все элементы стенда, участвующие в этом соединении, добавляются в общий список circuitComponents класса. При этом повторное добавление одного и того же компонента не допускается. Каждый компонент имеет свойство CountConnections. При соединении двух портов, значение свойства увеличивается на единицу. При

разрыве соединения, значение уменьшается на единицу. Если значение свойство становится равным нулю, компонент удаляется из контейнера и перестаёт участвовать в симуляции.

Поскольку в библиотеке SpiceSharp не реализованы объекты проводов, а соединение производится через строки, созданные пользователем провода добавляются в цепь как резисторы с нулевым сопротивлением. В отличие от компонентов модулей стенда, провода не имеют заранее определенных названий контактов. В момент коммутации, положительный контакт провода принимает название клеммы, из которой было создано соединение, а отрицательный контакт принимает название клеммы, в которую был проведен провод. Реализация таким способом позволяет упростить логику работы симулятора.

В симуляторе реализованы следующие классы компонентов:

– Resistor – класс резистора, имеет поле float res, содержащее значение сопротивления. При вызове метода ProvideCkt, добавляет в цепь объект SpiceSharp.Components.Resistor с присвоенным параметром сопротивления. Не участвует в экспорте данных;

– VoltageSource – класс источника напряжения, имеет поле float voltage, содержащее значение напряжения. При вызове метода ProvideCkt, добавляет в цепь объект SpiceSharp.Components.VoltageSource. Не участвует в экспорте данных;

– Ampermeter – класс амперметра. При вызове метода ProvideCkt, добавляет в цепь объект SpiceSharp.Components.Resistor с сопротивлением, соответствующим номиналу прибора. Экспортирует значение силы тока в данном участке цепи;

– Voltmeter – класс вольтметра. При вызове метода ProvideCkt, добавляет в цепь объект SpiceSharp.Components.Resistor с сопротивлением, соответствующим номиналу прибора. Экспортирует напряжение между двумя точками цепи;

– PowerMeter – класс измерителя мощности. При вызове метода ProvideCkt, добавляет в цепь один вольтметр и один амперметр. Вольтметр экспортирует напряжение подключенного генератора, а амперметр экспортирует силу тока в цепи. Мощность рассчитывается путём умножения напряжения на силу тока;

– Cable – класс провода, необходимый для связи элементов цепи. При вызове метода ProvideCkt, добавляет в цепь резистор нулевого сопротивления. Не участвует в экспорте;

Для отображения показаний к компонентам измерительных приборов должны быть привязаны соответствующие экземпляры индикаторов стенда. Все классы индикаторов наследуются от абстрактного класса Indicator.cs и должны реализовывать метод UpdateIndicatorValue. Данный метод вызывается в процессе экспорта симуляции внутри класса измерительного прибора.

Запуск процесса симуляции производится переключением главного переключателя на 220В, расположенного на модуле питания. При каждом изменении параметров или структуры цепи, симуляция перезапускается, а значения на измерительных приборах обновляются.

5 ОТОБРАЖЕНИЕ ПОЛУЧЕННЫХ РЕЗУЛЬТАТОВ НА ИЗМЕРИТЕЛЬНЫХ УСТРОЙСТВАХ

Для отображения результатов симуляции используется четыре типа индикаторов:

1. Цифровые линейные дисплеи на основе семисегментных индикаторов;
2. Аналоговые стрелочные индикаторы;
3. Матричный дисплей;
4. Мультиметр.

5.1 Линейный дисплей на основе семисегментных индикаторов

Для реализации дисплея было разработано два скрипта – скрипт «SSEG.cs» и скрипт «SSEGBuilder.cs».

«SSEG.cs» является объектом семисегментного индикатора и имеет два поля:

– Цвет активного сегмента индикатора;

– Массив из 7 сегментов индикатора и точки.

Для управления цветом индикатора используется свойство Color класса SSEG. Чтобы изменить цвет нужно присвоить данному свойству значение типа «UnityEngine.Color».

Для управления сегментами используется свойство Shape, принимающее массив из 8 элементов типа bool. Каждый элемент массива отвечает за включение или выключение сегмента, где значение true – включить сегмент, а false – выключить сегмент. На рисунке 5.1 показано соотношение индексов с управляемыми сегментами.

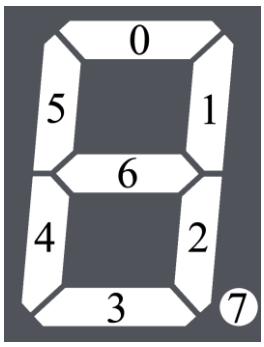

Рисунок 5.1 - Соотношение индексов с управляемыми сегментами

Класс «SSEGBuilder.cs» предназначен для построения дисплея на основе объектов класса SSEG заданной длины, изменения цвета индикаторов и вывода значения.

Имеет следующие свойства:

– Value – строковое значение, выводимое на дисплей;
– Color – цвет индикаторов;
– Length – целочисленное значение количества индикаторов;
– RightAlign – включение выравнивания по правому краю.

За переключение сегментов каждого индикатора отвечает метод DrawValues, принимающий на вход значение типа string. Полученное значение формализуется, после чего в цикле каждому индикатору дисплея присваивается свой символ. Для перевода из типа данных char в тип, принимаемый классом SSEG, используется метод CharToShape. Метод принимает один символ char и одно bool значение, отвечающее за точку. Возвращает массив из 8 элементов, соответствующий полученному символу при условии, что этот символ возможно вывести на индикатор. Если символ вывести нельзя, то метод вернёт выключенный дисплей.

Конечный вид дисплея показан на рисунке 5.2.

Рисунок 5.2 – Цифровой линейный дисплей

5.2 Аналоговые стрелочные индикаторы

Для имитации работы аналогового стрелочного измерителя был разработан скрипт «AnalogArrow.cs».

Скрипт имеет следующие параметры:

- Value – выводимое численное значение;
- Lower Limit – нижний предел измерения;
- Upper Limit – верхний предел измерения;
- Start Angle – положение начала шкалы в градусах;
- Arc Length – длина дуги шкалы в градусах;
- Arrow Speed – скорость вращения стрелки;
- Steps – количество делений шкалы;
- Scale Size – высота делений шкалы;
- Big Scale Size – высота крупных делений шкалы;
- Big Step Period – период крупных делений;
- Radius – радиус шкалы.

За создание шкалы отвечает метод DrawScale. Метод использует компонент LineRenderer [14] для создания линий между точками. Для расчёта координат точек на окружности используется метод Rad2Pos, принимающий значение радианы точки и радиус окружности и возвращающий координаты этой точки относительно нуля.

Для расчёта угла стрелки используется метод ValueToAngle. Угол определяется линейной интерполяцией между началом шкалы и её концом в соответствии с выводимым значением с учётом пределов измерения.

Внешний вид стрелочного измерителя представлен на рисунке 5.3.

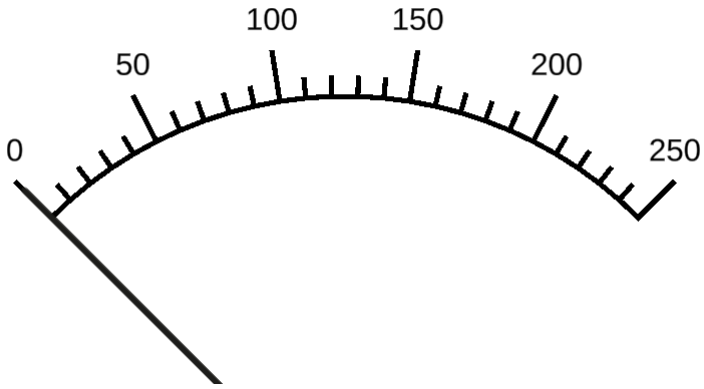

Рисунок 5.3 – Внешний вид стрелочного измерителя

5.3 Матричный дисплей

Для отображения матричного дисплея использован компонент TextMeshPro от Unity со специальным шрифтом, имитирующим отображение букв из отдельных пикселей на матричном дисплее [18].

Для реализации работы матричного дисплея был создан скрипт «MatrixDispaly.cs». Методом UpdateDisplay изменяется выводимый на дисплей текст. Метод принимает следующие параметры:

– double v – значение напряжения;

– double c – значение силы тока;

– double p – значение мощности.

Перед выводом, число округляется до двух знаков после запятой.

Финальный вид индикатора представлен на рисунке 5.4.

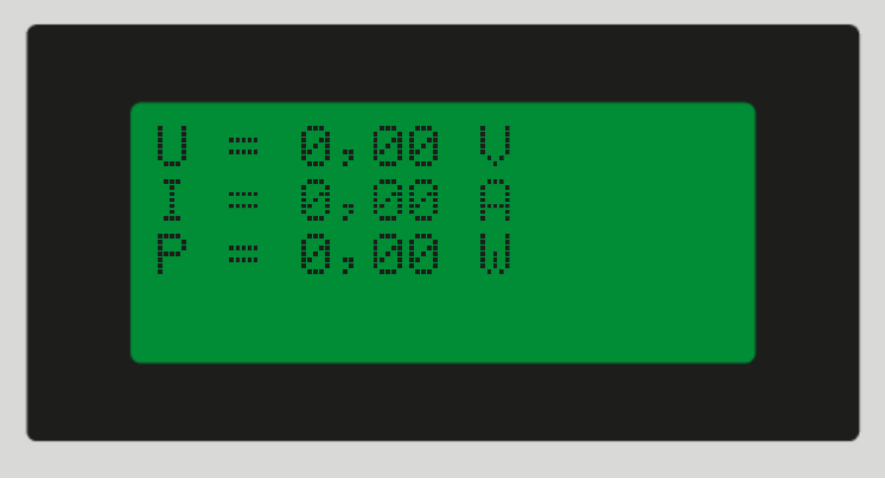

Рисунок 5.4 – Матричный дисплей

5.4 Модуль мультиметров

Помимо вышеперечисленных индикаторов, также был реализован модуль, состоящий из двух мультиметров. Выбор режима работы производится вращением ручки прибора. Внешний вид мультиметра представлен на рисунке 5.5.

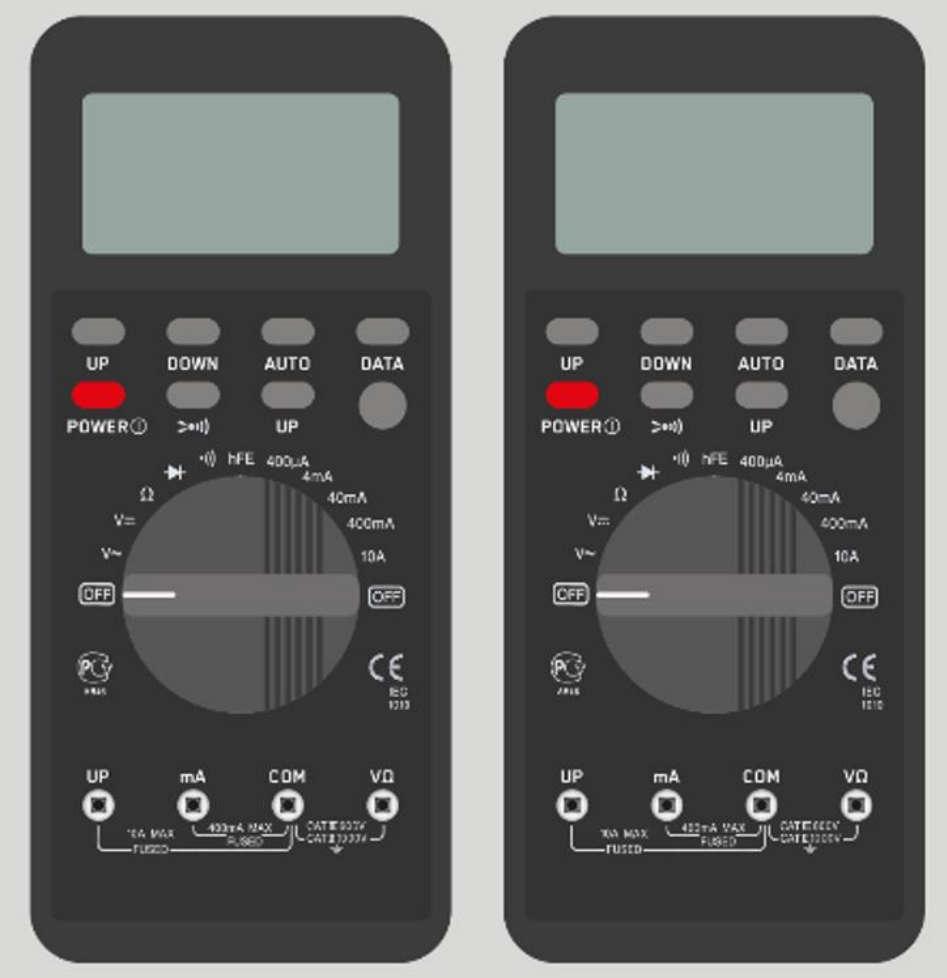

Рисунок 5.5 – Модуль мультиметров

5.5 Отображение всех параметров

Расположив разработанные индикаторы на модули стенда и привязав их к соответствующим компонентам цепи, можно продемонстрировать работу всех индикаторов стенда. Для этого была собрана простая цепь из источника напряжения, амперметра, трёх резисторов, вольтметра и измерителя мощности.

Монтажная схема данной цепи представлена на рисунке 5.6.

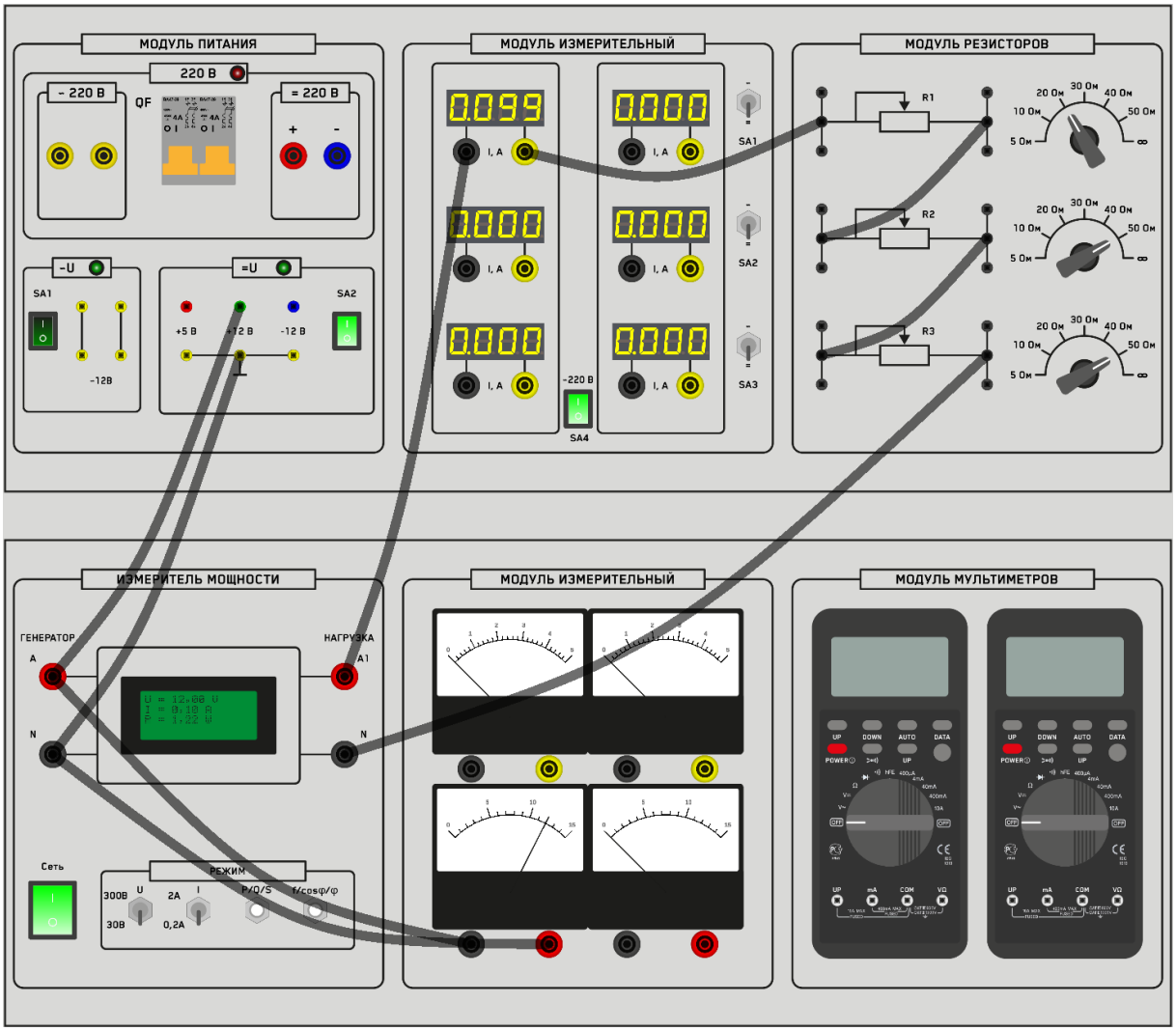

Рисунок 5.6 – монтажная схема цепи.

Источником является генератор на 12 Вольт, а общее сопротивление цепи равно сумме сопротивлений трёх резисторов и равно 120 Ом. Поделив 12 Вольт на 120 Ом, получим силу тока, равную 0.1А.

Соответственно, умножив напряжение и силу тока получим мощность, равную 1.2Ватт, что соответствует показаниям виртуального с учётом того, что все

измерительные приборы участвуют в симуляции цепи, а их номинал влияет на расчёты. На рисунках 5.6, 5.7 и 5.8 представлены показания индикаторов стенда.

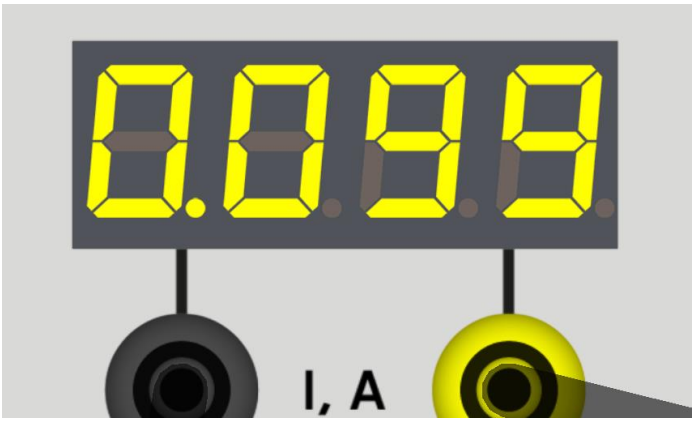

Рисунок 5.6 – Показание амперметра, А

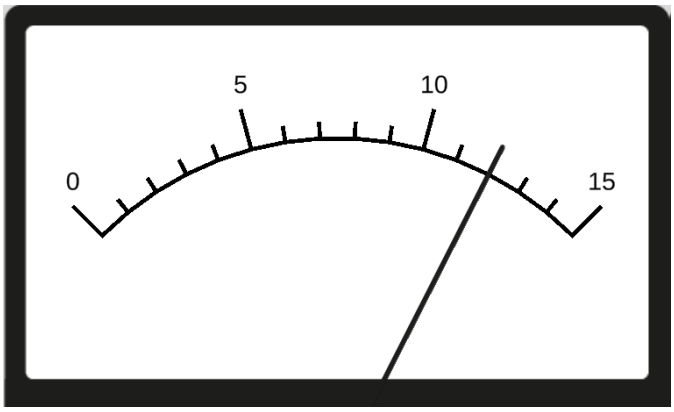

Рисунок 5.7 – Показание вольтметра, В

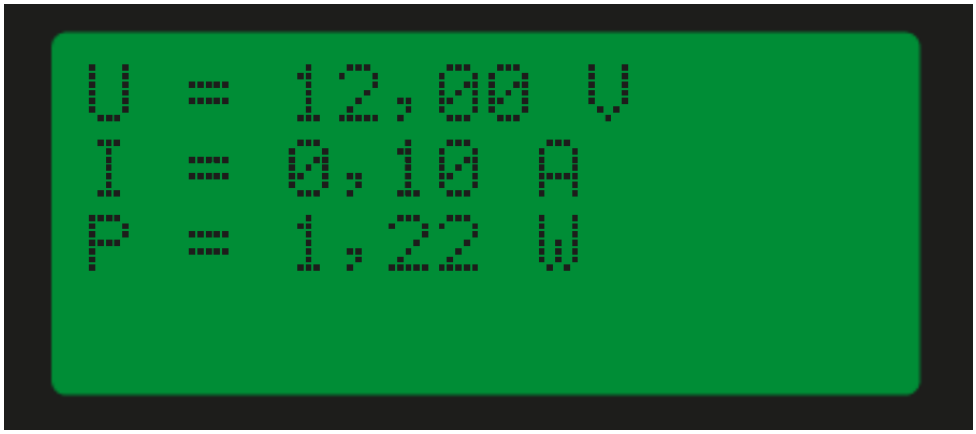

Рисунок 5.8 – Показание измерителя мощности

ЗАКЛЮЧЕНИЕ

В рамках данной выпускной квалификационной работы были выполнены следующие пункты:

1. Аналитический обзор аналогов.
2. Выбор средств разработки и определение требований.
3. Разработка пользовательского интерфейса симулятора.
4. Реализация расчёта силы тока и напряжения скоммутированной цепи.
5. Отображение полученных результатов на измерительных устройствах.

Разработанный симулятор позволяет выполнять лабораторные работы по электротехнике в разделе линейные электрические цепи постоянного тока. При этом, стенд способен симулировать любые построенные на нём цепи, что позволяет свободно расширять содержание лабораторных работ без необходимости в обновлении программы.

Значения, выводимые индикаторами симулятора, соответствуют расчётным, а внешний вид модулей соответствует реальному прототипу. Благодаря настраиваемой камере, все показания индикаторов стенда могут быть различимы на любом мониторе. Для управления стендом требуется только наличие подключённой компьютерной мыши.

Таким образом, были выполнены все установленные требования к симулятору.

БИБЛИОГРАФИЧЕСКИЙ СПИСОК

1. Курс: МООК_Теоретические основы электротехники (часть 1) [Электронный ресурс] // MOOC SUSU [сайт]. — URL: https://mooc.susu.ru/ (дата обращения: 15.02.23).
2. SPICE (симулятор электронных схем) [Электронный ресурс] // Википедия: свободная энциклопедия [сайт]. — URL: https://ru.wikipedia.org/wiki/SPICE_(симулятор электронных схем)/ (дата обращения: 16.02.23).
3. SpiceSharp [Электронный ресурс] // GitHub [сайт]. — URL: https://github.com/SpiceSharp/SpiceSharp/ (дата обращения: 16.02.23).
4. Unreal Engine [Электронный ресурс] // Skillfactory media [сайт]: — URL: https://blog.skillfactory.ru/glossary/unreal-engine/ (дата обращения: 22.02.2023).
5. Hardware and Software Specifications [Электронный ресурс] // Unreal Engine 5 Documentation [сайт]. — URL: https://docs.unrealengine.com/5.0/en-US/hardware-and-software-specifications-for-unreal-engine/ (дата обращения: 22.02.23).
6. Хокинг, Д.Р. Unity в действии. Мультиплатформенная разработка на C# / Пер. с англ. И. Рузмайкиной. — СПб.: Питер, 2016. — 20 с.
7. System requirements for Unity 2022. [Электронный ресурс] // Unity User Manual 2022.2 [сайт]. — URL: https://docs.unity3d.com/2022.2/Documentation/Manual/system-requirements.html/ (дата обращения: 23.02.23).
8. Unity Store. Plans and pricing [Электронный ресурс] // Unity [сайт]. — URL: https://store.unity.com/front-page?check_logged_in=1#plans-enterprise/ (дата обращения: 23.02.2023).
9. Серьёзные ошибки в коде CryEngine V [Электронный ресурс] // Habr [сайт] — URL: https://habr.com/ru/company/pvs-studio/blog/325600/ (дата обращения: 23.02.2023).

10. Игровой движок CryEngine от компании Crytek [Электронный ресурс] // Almaguz [сайт] — URL: https://almaguz.ru/igrovoy-dvizhok-cryengine-ot-kompanii-crytek/ (дата обращения: 23.02.2023).

11. CryEngine. System requirements [Электронный ресурс] // Cryengine [сайт] — URL: https://www.cryengine.com/support/view/system-requirements/ (дата обращения: 23.02.2023).

12. Input System [Электронный ресурс] // Unity Documentation [сайт]. — URL: https://docs.unity3d.com/Packages/com.unity.inputsystem@1.5/manual/index.html/ (дата обращения: 14.03.23).

13. Getting started with Splines [Электронный ресурс] // Unity Documentation [сайт]. — URL: https://docs.unity3d.com/Packages/com.unity.splines@2.2/manual/getting-started-with-splines.html/ (дата обращения: 18.03.23).

14. Line Renderer component [Электронный ресурс] // Unity Documentation [сайт]. — URL: https://docs.unity3d.com/2022.2/Documentation/Manual/class-LineRenderer.html/ (дата обращения: 20.03.23).

15. Camera.ScreenPointToRay [Электронный ресурс] // Unity Documentation [сайт]. — URL: https://docs.unity3d.com/2022.2/Documentation/ScriptReference/Camera.ScreenPointToRay.html/ (дата обращения: 15.03.23).

16. BoxCollider2D [Электронный ресурс] // Unity Documentation [сайт]. — URL: https://docs.unity3d.com/2022.2/Documentation/ScriptReference/BoxCollider2D.html/ (дата обращения: 16.03.23).

17. SpiceSharp [Электронный ресурс] // GitHub [сайт]. — URL: https://github.com/SpiceSharp/SpiceSharp/ (дата обращения: 05.04.23).

18. TextMeshPro [Электронный ресурс] // Unity Documentation [сайт]. — URL: https://docs.unity3d.com/Manual/com.unity.textmeshpro.html/ (дата обращения: 28.04.23).

19. Корнилов, А. В. Unity. Полное руководство: [Текст] / А. В. Корнилов. — Санкт-Петербург: Изд-во Наука и Техника, 2020. — 432 с. — ISBN 978-5-94387-795-7.

20. Ларкович, С. Н. Справочник UNITY. Кратко, быстро, под рукой [Справочник]/ С. Н. Ларкович. — Санкт-Петербург: Изд-во Наука и Техника, 2020. — 288 с. — ISBN 978-5-94387-667-7.

ПРИЛОЖЕНИЯ

ПРИЛОЖЕНИЕ А

Класс «StandInputSystem.cs»

```
public class StandInputSystem : MonoBehaviour
{
    [SerializeField] UICamera uiCamera;
    private InputActions inputActions;
    private Control selectedControl;

    #region Input
    private void Awake()
    {
        inputActions = new InputActions();
    }

    private void OnEnable()
    {
        inputActions.Camera.Enable();
        inputActions.Camera.DisableStandControls.performed +=
DisableStandControls_performed;
        inputActions.Camera.DisableStandControls.canceled +=
DisableStandControls_canceled;
        inputActions.Camera.CameraMovement.performed += CameraMovement_performed;
        inputActions.Camera.CameraZoomIn.performed += CameraZoomIn_performed;
        inputActions.Camera.CameraZoomOut.performed += CameraZoomOut_performed;
        inputActions.Stand.Enable();
        inputActions.Stand.LeftClick.performed += LeftClick_performed;
        inputActions.Stand.LeftClick.canceled += LeftClick_canceled;
        inputActions.Stand.RightClick.canceled += RightClick_canceled;
        inputActions.Stand.RightClick.performed += RightClick_performed;
        inputActions.Stand.Drag.performed += Drag_performed;
        inputActions.Stand.Drag.canceled += Drag_canceled;
    }

    private void OnDisable()
    {
        inputActions.Camera.Disable();
        inputActions.Stand.Disable();
    }
#endregion
#region Camera
    private void CameraMovement_performed(InputAction.CallbackContext obj)
    {
        uiCamera.CameraMovement(obj.ReadValue<Vector2>());
    }

    private void CameraZoomIn_performed(InputAction.CallbackContext obj)
    {
        uiCamera.CameraZoomIn();
    }

    private void CameraZoomOut_performed(InputAction.CallbackContext obj)
    {
        uiCamera.CameraZoomOut();
    }
    #endregion
    #region UI
    private void LeftClick_performed(InputAction.CallbackContext obj)
    {
```

```
            inputActions.Stand.RightClick.Disable();
            if (selectedControl = uiCamera.GetControlAtPointer())
            {
                selectedControl.LeftClick_performed(obj);
            }
        }

        private void LeftClick_canceled(InputAction.CallbackContext obj)
        {
            if (selectedControl)
            {
                selectedControl.LeftClick_canceled(obj, uiCamera.GetControlAtPointer());
            }

            selectedControl = null;
            inputActions.Stand.RightClick.Enable();
        }

        private void RightClick_performed(InputAction.CallbackContext obj)
        {
            if (selectedControl = uiCamera.GetControlAtPointer())
            {
                selectedControl.RightClick_performed(obj);
            }
        }

        private void RightClick_canceled(InputAction.CallbackContext obj)
        {
            if (selectedControl)
            {
                selectedControl.RightClick_canceled(obj, uiCamera.GetControlAtPointer());
            }

            selectedControl = null;
        }

        private void Drag_performed(InputAction.CallbackContext obj)
        {
            if (selectedControl)
            {
                selectedControl.Drag_performed(obj);
            }
        }

        private void Drag_canceled(InputAction.CallbackContext obj)
        {
            return;
        }
        #endregion
    }
```

ПРИЛОЖЕНИЕ Б

Класс «UICamera.cs»

```
[RequireComponent(typeof(Camera))]
public class UICamera : MonoBehaviour
{
    [SerializeField] private Camera mainCamera;
    [SerializeField] private Vector3 cameraZone;
    private float zoneAspect;
    private float previousAspect;
    [SerializeField] private float scaleStep;
    [SerializeField] private float scaleMin;
    [SerializeField] private float scaleMax;

    private void Awake()
    {
        previousAspect = mainCamera.aspect;
        AspectScaler(mainCamera.aspect, zoneAspect);
        zoneAspect = cameraZone.x / cameraZone.y;
    }

    private void Update()
    {
        OnAspectChange();
    }

    private void OnAspectChange()
    {
        if (mainCamera.aspect != previousAspect)
        {
            CameraScaler(0, cameraZone/2);
            previousAspect = mainCamera.aspect;
        }
    }

    private Vector3 ClampCamera(Vector3 targetPosition)
    {
        float cameraWidth = mainCamera.orthographicSize * mainCamera.aspect;
        float targetX = cameraWidth >= cameraZone.x ? 0 : Mathf.Clamp(targetPosition.x,
        -cameraZone.x + cameraWidth,
                                                      cameraZone.x - cameraWidth);

        float targetY = mainCamera.orthographicSize >= cameraZone.y ? 0 :
Mathf.Clamp(targetPosition.y,
                                                      -cameraZone.y +
mainCamera.orthographicSize,
                                                      cameraZone.y -
mainCamera.orthographicSize);

        return new Vector3(targetX, targetY, targetPosition.z);
    }

    private void CameraScaler(float delta, Vector2 scaleOrigin)
    {
        Vector3 originalPos = mainCamera.ScreenToWorldPoint(scaleOrigin);

        if (mainCamera.aspect >= zoneAspect)
        {
            mainCamera.orthographicSize = Mathf.Clamp(mainCamera.orthographicSize +
delta, scaleMin, scaleMax);
        }
        else
        {
```

```
            mainCamera.orthographicSize = Mathf.Clamp(mainCamera.orthographicSize +
delta, scaleMin * zoneAspect / previousAspect, scaleMax * zoneAspect / previousAspect);
        }
        transform.position = ClampCamera(transform.position + originalPos -
mainCamera.ScreenToWorldPoint(scaleOrigin));
    }

    private void AspectScaler(float newAspect, float prevAspect)
    {
        if (newAspect < zoneAspect)
        {
            mainCamera.orthographicSize = Mathf.Clamp(mainCamera.orthographicSize *
prevAspect / newAspect, scaleMin, scaleMax);;
        }
    }

    public void CameraMovement(Vector2 delta)
    {
        Vector3 targetPosition = transform.position -
                    (mainCamera.ScreenToWorldPoint(delta) -
                    mainCamera.ScreenToWorldPoint(Vector2.zero));
        transform.position = ClampCamera(targetPosition);
    }

    public void CameraZoomIn()
    {
        CameraScaler(-scaleStep, Pointer.current.position.ReadValue());
    }

    public void CameraZoomOut()
    {
        CameraScaler(scaleStep, Pointer.current.position.ReadValue());
    }

    public GameObject GetObjectAtPointer()
    {
        Ray ray = mainCamera.ScreenPointToRay(Pointer.current.position.ReadValue());
        RaycastHit2D hit = Physics2D.GetRayIntersection(ray);
        if (hit.collider != null)
        {
            return hit.collider.gameObject;
        }
        else
        {
            return null;
        }
    }

    public Control GetControlAtPointer()
    {
        GameObject gameObject;
        if(!(gameObject = GetObjectAtPointer()))
        {
            return null;
        }
        Control control;
        if (!(control = gameObject.GetComponent<Control>()))
        {
            return null;
        }
        return control;
    }}
```

ПРИЛОЖЕНИЕ В

Класс «Cable.cs»

```
public class Cable : CircuitComponent
{
    [Header("Spline")]
    [SerializeField] public SplineContainer splineContainer;
    private Spline spline;


    [Header("Points")]
    public Vector3 startPoint;
    public Vector3 endPoint;


    [Header("Curvature")]
    [Range(0,1)]
    [SerializeField] private float m_sag;
    [Range(0, 5)]
    [SerializeField] private float m_weight;
    [SerializeField] private float m_maxWeight;


    [Header("Smoothness")]
    [SerializeField] private bool m_smooth;
    [Range(0,1)]
    [SerializeField] private float m_smoothSpeed;


    [Header("Line Renderer")]
    [SerializeField] private LineRenderer lineRenderer;

    [SerializeField, Min(.0001f)]
    public float m_segmentsPerUnit;

    private Vector3 currentStartTangent = Vector3.zero;
    private Vector3 currentEndTangent = Vector3.zero;
    private Vector3 desiredStartTangent;
    private Vector3 desiredEndTangent;
    private BezierKnot startKnot;
    private BezierKnot endKnot;


    // Circuit
    private CircuitContainer circuitContainer;
    public Port connectedStartPort;
    public Port connectedEndPort;
    public string m_posNode;
    public string m_negNode;
    private int m_countConnections;


    public override string PosNode { get { return m_posNode; } set { m_posNode = value; }
}
    public override string NegNode { get { return m_negNode; } set { m_negNode = value; }
}
    public override int CountConnections
    {
        get { return m_countConnections; }
        set
        {
            m_countConnections = value;
            if (m_countConnections == 0)
```

```
            {
                circuitContainer.RemoveComponent(this);
            }
            else
            {
                circuitContainer.AddComponent(this);
            }
        }
    }


    public override void ProvideDC(DC dc)
    {

    }


    public override void ProvideCkt(Circuit ckt)
    {
        SpiceSharp.Components.Resistor resistor = new
SpiceSharp.Components.Resistor(name, PosNode, NegNode, 0);
        ckt.Add(resistor);
    }


    public void Start()
    {
        circuitContainer =
GameObject.FindGameObjectWithTag("CircuitContainer").GetComponent<CircuitContainer>();
        splineContainer =
GameObject.FindGameObjectWithTag("CableContainer").GetComponent<SplineContainer>();
        splineContainer.AddSpline();
        spline = splineContainer.Splines[splineContainer.Splines.Count - 1];
        spline.Add(new BezierKnot(startPoint));
        spline.Add(new BezierKnot(45ndpoint));
    }


    private void FixedUpdate()
    {
        UpdateKnots();
        DrawCable();
    }


    private void UpdateKnots()
    {
        float length = Vector3.Distance(startPoint, 45ndpoint);
        Vector3 endLocalPos = 45ndpoint - startPoint;

        startKnot = new BezierKnot(startPoint);
        45ndpoin = new BezierKnot(45ndpoint);

        float sag = endLocalPos.y >= 0 ? m_sag : 1 - m_sag;
        float weight = Mathf.Clamp(length * m_weight, 0, m_maxWeight);

        desiredStartTangent = new Vector3(Mathf.Lerp(0, endLocalPos.x, sag),
                                          Mathf.Lerp(endLocalPos.y, 0, sag) - weight,
                                          Mathf.Lerp(0, endLocalPos.z, sag));

        desiredEndTangent = new Vector3(Mathf.Lerp(-endLocalPos.x, 0, sag),
                                        Mathf.Lerp(0, -endLocalPos.y, sag) - weight,
                                        Mathf.Lerp(-endLocalPos.z, 0, sag));
```

```
        if (m_smooth)
        {
            desiredStartTangent = Vector3.Lerp(currentStartTangent, desiredStartTangent,
m_smoothSpeed);
            desiredEndTangent = Vector3.Lerp(currentEndTangent, desiredEndTangent,
m_smoothSpeed);
        }


        startKnot.TangentOut = desiredStartTangent;
        endKnot.TangentIn = desiredEndTangent;

        spline.SetKnot(0, startKnot);
        spline.SetKnot(1, endKnot);

        currentStartTangent = desiredStartTangent;
        currentEndTangent = desiredEndTangent;
    }


    private void DrawCable()
    {
        Vector3[] pos = GetSplinePositions(spline);
        lineRenderer.positionCount = pos.Length;
        lineRenderer.SetPositions(pos);
    }


    private Vector3[] GetSplinePositions(Spline spline)
    {
        List<Vector3> positions = new List<Vector3>();
        float _t = 0;
        positions.Add(spline.EvaluatePosition(_t));

        while (_t < 1)
        {
            spline.GetPointAtLinearDistance(_t, 1 / m_segmentsPerUnit, out _t);
            positions.Add(spline.EvaluatePosition(_t));
        }

        return positions.ToArray();
    }


    public void OnDestroy()
    {
        splineContainer.RemoveSpline(spline);
    }

    public override string ConnectionString()
    {
        return name + ": " + m_posNode + " " + m_negNode;
    }
      }
```

ПРИЛОЖЕНИЕ Г

Класс «Port.cs»

```
public class Port : Control
{
    [SerializeField] private CircuitComponent m_component;
    [SerializeField] public bool isPositiveNode;
    private Camera mainCamera;
    private CableContainer cableContainer;
    private CircuitContainer circuitContainer;
    public List<Cable> cables;
    private Cable currentCable;


    public CircuitComponent Component
    {
        get
        {
            return m_component;
        }
    }


    private void Start()
    {
        cableContainer =
GameObject.FindGameObjectWithTag("CableContainer").GetComponent<CableContainer>();
        mainCamera =
GameObject.FindGameObjectWithTag("MainCamera").GetComponent<Camera>();
        circuitContainer =
GameObject.FindGameObjectWithTag("CircuitContainer").GetComponent<CircuitContainer>();

        cables = new List<Cable>();

        if (m_component)
        {
            if (isPositiveNode)
            {
                if (m_component.GetType() == typeof(VoltageSource))
                {
                    name = "0";
                }
                else
                {
                    name = m_component.PosNode;

                }
            }
            else
            {
                name = m_component.NegNode;
            }
        }
    }


    public override void LeftClick_performed(InputAction.CallbackContext obj)
    {
        currentCable = cableContainer.CreateCable();
        currentCable.startPoint = currentCable.endPoint = transform.position;
    }
```

```
    public override void LeftClick_canceled(InputAction.CallbackContext obj, Control
canceledOn)
    {
        if (!canceledOn || canceledOn.GetComponent<Port>() == null || canceledOn == this)
        {
            cableContainer.DestroyCable(currentCable);
        }
        else
        {
            Port canceledOnPort = canceledOn.GetComponent<Port>();

            Component.CountConnections++;
            cables.Add(currentCable);
            currentCable.connectedStartPort = this;
            currentCable.PosNode = name;

            canceledOnPort.Component.CountConnections++;
            canceledOnPort.cables.Add(currentCable);
            currentCable.connectedEndPort = canceledOnPort;
            currentCable.NegNode = canceledOnPort.name;

            currentCable.CountConnections += 2;

            currentCable.endPoint = canceledOn.transform.position;
            currentCable = null;
        }
    }


    public override void RightClick_performed(InputAction.CallbackContext obj)
    {

    }


    public override void RightClick_canceled(InputAction.CallbackContext obj, Control
canceledOn)
    {
        if (cables.Count != 0)
        {
            cableContainer.DestroyCable(cables.Last());
        }
    }


    public override void Drag_performed(InputAction.CallbackContext obj)
    {
        currentCable.endPoint =
mainCamera.ScreenToWorldPoint(Pointer.current.position.ReadValue());
        currentCable.endPoint.z = 0;
    }


    public override void Scroll(InputAction.CallbackContext obj)
    {

    }
      }
```

ПРИЛОЖЕНИЕ Д

Класс «CircuitContainer.cs»

```
public class CircuitContainer : MonoBehaviour
{
    public List<CircuitComponent> circuitComponents = new List<CircuitComponent>();


    public void AddComponent(CircuitComponent component)
    {
        if (!circuitComponents.Contains(component))
        {
            circuitComponents.Add(component);
        }
    }


    public void RemoveComponent(CircuitComponent component)
    {
        if (circuitComponents.Contains(component))
        {
            circuitComponents.Remove(component);
        }
    }

    public void RunSimulation()
    {
        Circuit ckt = new Circuit();
        DC dc = new DC("DC");

        foreach(CircuitComponent circuitComponent in circuitComponents)
        {
            circuitComponent.ProvideCkt(ckt);
            circuitComponent.ProvideDC(dc);
        }

        try
        {
            dc.Run(ckt);
        }
        catch(ValidationFailedException e)
        {
            Debug.Log(e);
        }
    }
}
```

ПРИЛОЖЕНИЕ Е

Класс «Resistor.cs»

```
public class Resistor : CircuitComponent
{
    public CircuitContainer circuitContainer;
    public string m_posNode;
    public string m_negNode;
    public int m_countConnections = 0;
    public float res;

    public override string PosNode { get { return m_posNode; } set { m_posNode = value; }
}
    public override string NegNode { get { return m_negNode; } set { m_negNode = value; }
}
    public override int CountConnections
    {
        get { return m_countConnections; }
        set
        {
            m_countConnections = value;
            if (m_countConnections == 0)
            {
                circuitContainer.RemoveComponent(this);
            }
            else
            {
                circuitContainer.AddComponent(this);
            }
        }
    }


    public void Awake()
    {
        circuitContainer =
GameObject.FindGameObjectWithTag("CircuitContainer").GetComponent<CircuitContainer>();
        m_posNode = name + "Pos";
        m_negNode = name + "Neg";
    }

    public override void ProvideDC(DC dc)
    {

    }


    public override void ProvideCkt(Circuit ckt)
    {
        SpiceSharp.Components.Resistor resistor = new
SpiceSharp.Components.Resistor(name, PosNode, NegNode, res);
        ckt.Add(resistor);
    }

    public override string ConnectionString()
    {
        return name + ": " + m_posNode + " " + m_negNode;
    }
}
```