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 системного программирования

ДОПУСТИТЬ К ЗАЩИТЕ

Заведующий кафедрой, д.ф.-м.н.,
профессор

_____ Л.Б. Соколинский

«___»_____ 2024 г.

Разработка программных компонентов для симулятора демонтажного робота на платформе Unity

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА
ЮУрГУ – 02.03.02.2024.308-311.ВКР

Научный руководитель,
ст. преподаватель кафедры СП
_____ П.Г. Верман

Автор работы,
студент группы КЭ-401
_____ А.В. Севанькаев

Ученый секретарь
(нормоконтролер)
_____ И.Д. Володченко
«___»_____ 2024 г.

Челябинск, 2024 г.

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 системного программирования

УТВЕРЖДАЮ
Зав. кафедрой СП
_____ Л.Б. Соколинский
29.01.2024 г.

ЗАДАНИЕ
на выполнение выпускной квалификационной работы бакалавра
студенту группы КЭ-401
Севанькаеву Андрею Вячеславовичу,
обучающемуся по направлению
02.03.02 «Фундаментальная информатика и информационные технологии»

1. **Тема работы** (утверждена приказом ректора от 22.04.2024 г. № 764-13/12)
Разработка программных компонентов для симулятора демонтажного робота на платформе Unity.
2. **Срок сдачи студентом законченной работы:** 03.06.2024 г.
3. **Исходные данные к работе**

3.1. Хокинг Д. Unity – в действии. Мультиплатформенная разработка на C#. – СПб: Питер, 2016 г. – 336 с.

3.2. Найстром Р. Шаблоны игрового программирования. – Издательство Бомбора, 2022 – 456 с.

3.3. Рихтер Д. CLR via C#. Программирование на платформе Microsoft .NET Framework 4.5 на языке C#. 4-е изд. – Питер, 2019 – 896 с.

3.4. Флетчер Д. 3D Math Primer for Graphics and Game Development, second edition. //Taylor & Francis, 2011. – 896 с.

4. **Перечень подлежащих разработке вопросов**

4.1. Выполнить анализ предметной области.

4.2. Спроектировать программные компоненты.

4.3. Реализовать программные компоненты.

4.4. Провести тестирование.

5. Дата выдачи задания: 29.01.2024 г.

Научный руководитель,
ст. преподаватель кафедры СП П.Г. Верман

Задание принял к исполнению А.В. Севанькаев

ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ

Актуальность

Цифровая индустрия 4.0 была представлена на промышленной выставке в Ганновере в 2011 году и с тех пор олицетворяет переход к автоматизации, снижению участия человека и оптимизации операций через аналитику данных. Она представляет четвертую промышленную революцию, которая отличается интеграцией технологий Интернета вещей, искусственного интеллекта и больших данных в производственные процессы [1]. Эта интеграция не только усиливает производственные возможности, но и открывает новые пути для инноваций и управления ресурсами.

Симуляторы, как элемент индустрии 4.0, обеспечивают виртуальное моделирование реальных процессов, что позволяет компаниям тестировать различные стратегии и операции без фактических рисков. Симуляторы становятся инструментами для обучения персонала, оптимизации процессов и предсказания потенциальных недостатков еще до начала производства. Это позволяет не только снижать затраты, но и значительно увеличивать эффективность производства и бизнес-процессов благодаря лучшему планированию и управлению [2].

В настоящее время симуляторы становятся все более востребованными в области обучения и тренировки специалистов. Стоит отметить, что они также находят применение в оценке профессиональных навыков и тестировании кандидатов [3]. Их актуальность подтверждается повышенным спросом от компаний, занимающихся обучением и подготовкой специалистов [4].

Постановка задачи

Целью выпускной квалификационной работы является разработка программных компонентов для симулятора демонтажного робота на платформе Unity. Для достижения поставленной цели необходимо решить следующие задачи:

1) выполнить анализ предметной области;

2) спроектировать программные компоненты;
3) реализовать программные компоненты;
4) провести тестирование.

Структура и содержание работы

Работа состоит из введения, четырех глав, заключения и списка литературы. Объем работы составляет 48 страниц, объем списка литературы – 22 источника.

В первой главе описывается анализ области симулятора и обзор существующих аналогичных проектов, на основе которых были созданы необходимые требования к создаваемым компонентам.

Вторая глава посвящена описанию анализа требований к программным компонентам, приведены функциональные и нефункциональные требования к системе, а также определены и описаны варианты использования.

В третьей главе приведена реализация компонентов системы, представлены средства реализации, описаны компоненты симулятора и показана демонстрация работы реализованных компонентов симулятора демонтажного робота.

В четвертой главе представлены результаты тестирования программных компонентов.

1. АНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ

1.1. Предметная область проекта

Симуляторы строительной техники представляют собой программы, позволяющие пользователям пережить реалистичный опыт управления различными видами строительной машиностроения в виртуальной среде. Такие симуляторы позволяют имитировать различные условия работы, а также обучать навыкам управления техникой в различных сценариях, при помощи как аппаратного, так и программного оборудования [5].

Симуляторы не только предлагают виртуальную реализацию техники, но также воссоздают реалистичные 3D модели машин, имитируют реалистичное поведение физики и строительных площадок, что помогает приобрести навыки управления без фактической опасности или риска. Также они предоставляют возможность управлять виртуальной реализацией техники с помощью специфичного для нее устройства ввода, тренажеров, для подготовки людей к работе с реальным оборудованием и техникой [6].

1.2. Обзор существующих решений

Существует множество симуляторов техники, использующихся в разных сферах. Были изучены современные представители строительных симуляторов и тренажеров: «Construction Simulator», «Sim Excavator 2022» и тренажер-симулятор «ЧЕТРА». Выбор основывался на следующих критериях:

1) наличие возможности управлять строительной техникой;
2) реалистичное поведение техники и окружения;
3) предоставляет различные сценарии работ;
4) возможность работать с разными устройствами ввода.

Construction Simulator

Строительный симулятор «Construction Simulator» предлагает опыт взаимодействия с большим парком специальной техники [7]. На рисунке 1 представлен скриншот из проекта.

Рисунок 1 – Скриншот из симулятора «Construction Simulator»

В данном симуляторе пользователь в роли оператора специальной техники выполняет различные задания, связанные со строительством или перевозкой. Выполнять эти задания помогает большой спектр различных грузовых машин и строительной техники. Это позволяет пользователям выполнять в разнообразные задачи, от земляных работ до укладки асфальта. Кроме того, симулятор предлагает детально проработанные локации и реалистичные условия строительства.

В симуляторе присутствует деформация ландшафта при работе со специальными видами техники на строительных площадках. При помощи бульдозера можно уровнять землю, экскаватором вырыть яму и вырытую землю поместить в грузовую машину, чтобы отвезти в другое место, где с помощью нее заполнить яму или повысить уровень земли т.д. Техника в симуляторе ведет себя реалистично, при поворотах на большой скорости крениться, груженная машина тормозит дольше чем пустая и т.д. Показатели машин отображается с помощью удобного интерфейса, пользователь должен следить за ними для корректной работы техники.

Симулятор обладает высоким уровнем графики, что видно в высокой детализации техники и окружения. Construction Simulator позволяет использовать различные виды джойстиков и клавиатуру, но устройства ввода симулирующие реальные пульты управления различных типов техники не поддерживает.

Sim Excavator 2022

На рисунке 2 представлен скриншот из симулятора экскаватора «Sim Excavator 2022». Он является тренажером по работе с экскаватором [8].

Рисунок 2 – Скриншот из симулятора «Sim Excavator 2022»

В данном симуляторе необходимо управлять экскаватором и выполнять поставленные задания, которые по мере выполнения усложняются. Также пользователю предоставляется специальный пульт управления имитирующий реальный аналог.

Экскаватор симулирует все датчики и поведение реальных аналогов, поэтому пользователю необходимо следить за всеми датчиками машины при выполнении заданий: температура двигателя, давления масла и т.д. Виртуальная машина обладает проработанной физикой передвижения и взаимодействия с ландшафтом, при езде учитывается тип грунта и наклон,

что влияет на скорость машины и расход топлива. Также Sim Excavator 2022 обладает высокой детализацией техники.

Данный симулятор предлагает большое количество маневров, которые должен знать оператор экскаватора, программа фиксирует ошибки, задает норматив по времени, а также смотрит правильность выполнения заданий. На основе собранных данных выдает рекомендации по окончании упражнения. Выполнять упражнения оператор может с помощью клавиатуры и мыши, специального пульта управления, или учебного тренажера «Экскаватор-студент», имитирующего нахождение внутри кабины машины.

Учебный симулятор бульдозера «ЧЕТРА»

На рисунке 3 представлен скриншот из тренажера-симулятора «ЧЕТРА», который используется для обучения операторов машин [9].

Рисунок 3 – Скриншот из тренажера-симулятора «ЧЕТРА»

Тренажер-симулятор бульдозера предназначен для обучения и повышения квалификации операторов бульдозеров без фактического исполь-

зования реальной техники. Он создает условия работы в различных дорожных и погодных условиях, имитирует визуальную и звуковую обстановку, а также оснащен управляющими органами и индикацией, симулирующими реальный бульдозер. Таким образом, операторы могут практиковаться в безопасной и контролируемой среде, что способствует улучшению их навыков.

Оператор должен следить за параметрами виртуальной техники с помощью интерфейса, который симулирует показатели настоящей техники, и учитывать их, чтобы техника работала корректно. В симуляторе реализована взаимодействие грунта с техникой, работа техники отличается на разных типах грунта. Бульдозер может изменять ландшафт на рабочих площадках.

В симуляторе «ЧЕТРА» кроме модуля оператора присутствует модуль инструктора, представляющий специально оборудованный компьютер, который следит за выполнением работы оператора и может общаться с ним по специальным устройствам связи.

1.3. Анализ представленных решений

Все представленные проекты обладают системой заданий, они предоставляют оператору различные цели для выполнения, ознакомления с техникой или обучения работы с ней, при помощи маркеров в интерактивной среде симулятора (виртуальном или игровом мире) и пользовательского интерфейса.

Все решения обладают реалистичным поведением техники. Техника в представленных проектов симулирует реальные аналоги их передвижение, взаимодействие с окружающей средой.

Во всех решениях присутствует возможность работы с различными устройствами ввода, клавиатура с мышью, джойстики, специальные пульты тренажеры или модули, имитирующие нахождение внутри техники.

В каждом решении присутствует пользовательские интерфейс, информирующий пользователя о текущем состоянии техники, задании и т.д. В каждом решении интерфейс отличается в зависимости от типа используемой техники.

Таким образом, разрабатываемый симулятор должен иметь приближенную к реальности интерактивную среду с возможностью подключения специальных устройств управления, в которой предоставляется информация о текущем состоянии технике в зависимости от используемых устройств управления и в которой есть система заданий для обучения специалистов.

Вывод по первому разделу

В данном разделе был проведен анализ предметной области симуляторов специальной техники, была определена предметная область проекта, а также были рассмотрены существующие решения в данной предметной области. Были выявлены сходства и различия в приемах и тонкостях, характерных для данной категории симуляторов. Сравнительный анализ выявил общие подходы, применяемые различными разработчиками симуляторов специальной техники. Рассмотрение аналогов помогло увидеть общие тенденции и особенности, присущие данному сегменту виртуальных симуляторов.

2. ПРОЕКТИРОВАНИЕ ПРОГРАММНЫХ КОМПОНЕНТОВ

2.1. Требования к системе

Приложение позволяет пользователю управлять демонтажным роботом. Для разделения логики главного меню и основного процесса в разных окружениях используются сцены. Сцена представляет собой пространство, которое содержит объекты виртуального окружения и собственную логику. Они могут использоваться для создания главного меню, отдельных уровней и для других целей. В приложении есть сцена главного меню и основная сцена.

Сцена главного меню предоставляет позволяет выбрать сцену с необходимым сценарием управления демонтажным роботом и ознакомиться с управлением демонтажного робота.

На всех основных сценах пользователю необходимо выполнять задания управляя демонтажным роботом, каждая основная сцена имеет свою последовательность заданий и виртуальное окружение. Основные сцены имеют одинаковый список программных компонентов, функционал и возможности. Отличается набор предоставляемых заданий и окружения, так как каждая сцена должна предоставлять различные ситуации работы с демонтажным роботом.

Функциональные требования к проектируемой системе:

1) система должна отображать состояние компонентов демонтажного робота;

2) система должна иметь возможность завершать сцену досрочно и выбирать новую из предложенных;

3) система должна предоставлять возможность менять устройство ввода во время нахождения на основной сцене.

Нефункциональные требования к проектируемой системе:

1) система должна работать на операционной системе Windows 10 и выше;

2) система должна быть разработана на платформе Unity, с использованием языка C#;

3) главное меню должно быть разработано при помощи инструмента разработки интерактивных интерфейсов UI Toolkit [10], предоставляемым Unity.

2.2. Диаграмма вариантов использования

На рисунке 4 представлена диаграмма вариантов использования симулятора (тренажера-симулятора) демонтажного робота.

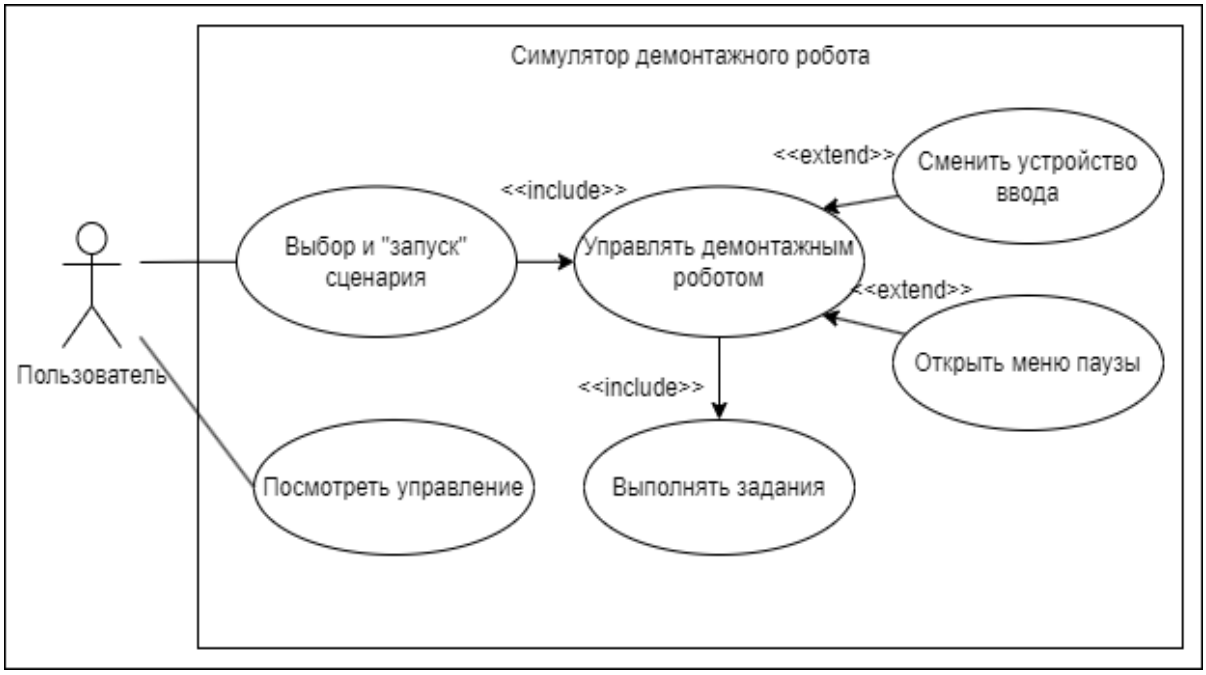

Рисунок 4 – Диаграмма вариантов использования

С системой взаимодействует только один основной актер – пользователь. Пользователь – это человек, использующий симулятор демонтажного робота и выполняющий определенные задания в симуляторе.

Пользователь может сделать выбор сценария и его запуск, приложение имеет набор сделанных сценариев, где каждый сценарий показывает определенную ситуацию работы демонтажного робота или обучает управлению техникой.

Пользователь может посмотреть управление, в меню пользователь может ознакомиться с назначением клавиш и их комбинаций на специальном контроллере, имитирующем устройство дистанционного управления демонтажным роботом.

Пользователь может управлять демонтажным роботом для ознакомления и обучения работы с демонтажным роботом, перемещения по интерактивной среде, взаимодействия с виртуальным окружением, выполнения задач.

Пользователь может выполнять задания, размещенные на сцене, задания служат для ознакомления пользователя с работой демонтажного робота. Задачи бывают разных видов, например, задание передвижения до выделенной зоны, задание разрушения определенных объектов.

Пользователь может менять устройство ввода для использования специального устройства имитирующего настоящий контроллер демонтажного робота или клавиатуры.

В ходе симуляции пользователь может открыть меню паузы, процесс управления и симуляции интерактивной среды приостанавливается, а пользователю предоставляется выбор о продолжении или досрочном завершении сессии.

2.3. Компоненты основной сцены симулятора

Основная сцена симулятора демонтажного робота, сцена, где мы управляем демонтажным роботом, выполняем поставленные задачи и взаимодействуем с интерактивной средой, состоит из нескольких компонентов, где каждый отвечает за определенную часть логики сцены. На рисунке 5 показана диаграмма компонентов основной сцены симулятора демонтажного робота.

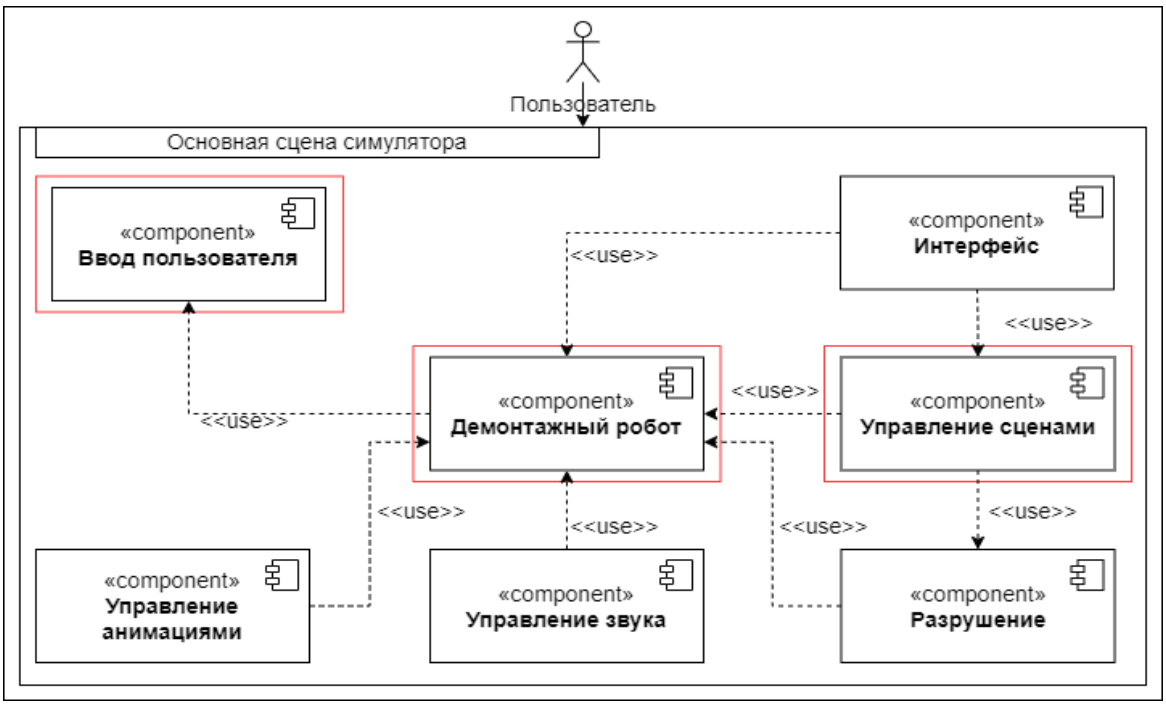

Рисунок 5 – Диаграмма компонентов основной сцены

Компонент «ввод пользователя» отвечает за обработку ввода. Он позволяет обрабатывать ввод с клавиатуры и специального контроллера имитирующего устройство дистанционного управления демонтажным роботом, изменять устройство ввода во время симуляции. Является посредником между программной реализацией интерфейса устройства ввода и интерфейсом управления сцены.

Компонент «управление сценами» отвечает за управление сценами и задачами на них. При загрузке сцены отвечает за последовательное предоставление задач пользователю. По завершению задач переключение в главное меню или повтор попытки.

Компонент «демонтажный робот» обеспечивает симуляцию работы демонтажного робота, его передвижение в интерактивной среде, взаимодействие с окружающим виртуальным миром, симуляцию работы башни, стрелы, опор (лап) и гусениц.

Компонент «интерфейс» предоставляет пользователю информацию о текущем состоянии демонтажного робота, задачах, степени разрушения поверхности.

Компонента «управление анимациями» отвечает за анимацию отдельных деталей демонтажного робота, определенных последствий взаимодействия демонтажного робота и интерактивной среды, например, эффектов при разрушении объектов.

Компонент «разрушения» позволяет разделить объект на множество кусочков и позволяет взаимодействовать с этим объектом, во время симуляции как с разрушаемым, он позволяет настраивать структуру разрушаемого объекта, можно устанавливать объекты, при разрушении которых смежные также будут разрушаться. Также компонент введет подсчет степени разрушения.

Компонент «управление звука» отвечает за управление звука во время симуляции работы демонтажного робота. Вызов звуков во время взаимодействия с интерактивной средой, перемещения и т.п.

Требуется разработка в рамках компонентов, выделенных красным цветом. Необходимо разработать компонент управления сценами, компонент ввода пользователя. Также необходимо реализовать в компоненте демонтажного робота поворот башни и стрелы, реализовать посредника между характеристиками демонтажного робота и интерфейсом.

2.4. Компонент управления сценами

Можно выделить следующий набор функциональных требований:

1) переход в главное меню по завершению всех задача на текущей сцене;
2) выбор любой сцены из главного меню;
3) возможность добавлять задачи на сцену в любом порядке;
4) возможность закончить сцену досрочно.

Компонент управления сценами отвечает за переключение сцен, отслеживание прогресса прохождения сценария, порядок выполнения заданий. На рисунке 6 представлена диаграмма компонентов программного компонента задач.

Рисунок 6 – Диаграмма компонентов компонента управления сценами

Компонент «менеджер сцен» отвечает за переключение сцен и отслеживания завершения сценария ни них. Обеспечивает автоматический переход в главное меню, по завершении сценария на текущей сцене. Позволяет выбрать конкретную сцену для загрузки.

Компонент «сценарий» отвечает за управление задачами, их порядком на текущей сцене и обеспечивает автоматическую смену задач, когда они выполнятся. Уведомляет компонент менеджера сценариев о завершении всех задач на текущей сцене.

Компонент «задача» отвечает за предоставление текущего задания в сценарии, которое должен выполнить пользователь. Обеспечивает оповещение компонента сценария о своем завершении.

2.5. Компонент ввода пользователя

Можно выделить следующий набор функциональных требований:

1) смену устройства можно производить на основной сцене;

2) смена устройства должна производиться при нажатии клавиши на клавиатуре;

3) управление мышкой должно быть доступно при любом устройстве ввода.

Компонент считывания ввода пользователя позволяет использовать различные устройства ввода и переключать их во время выполнения заданий на сцене. На рисунке 7 представлена диаграмма компонентов программного компонента считывания ввода.

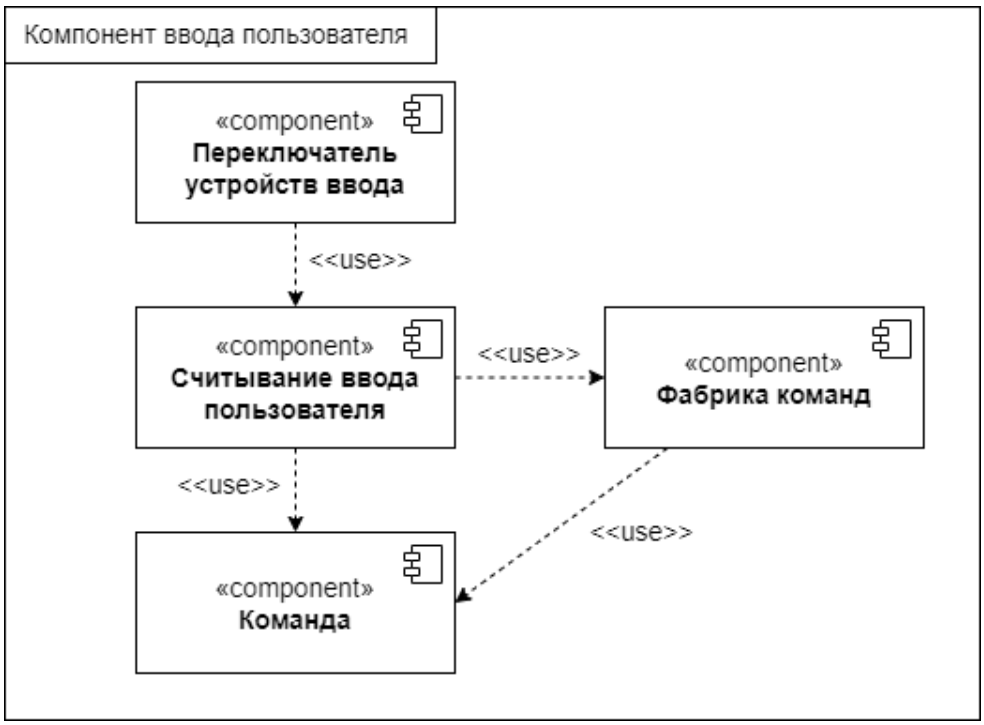

Рисунок 7 – Диаграмма компонентов компонента ввода пользователя

Компонент «переключатель устройств ввода» позволяет использовать различные устройства ввода, клавиатурой и контроллером, имитирующим работу устройство дистанционного управления демонтажным роботом, и переключаться в любой момент между устройствами, во время симуляции работы демонтажного робота.

Компонент «команда» отвечает за вызов действия в системе, не привязываясь к реализации устройства ввода, например, движение левой гусеницей вперед, движение первой секции стрелы вверх. Является реализацией паттерна проектирования «команда» [11]. Для команд, отвечающих за вызов действия демонтажного робота, требуются дополнительные параметры, вызывающий объект и в случае поворота элемента, например, башня, ось, вокруг которой должна вращаться башня.

Компонент «считывание устройства ввода» пользователя отвечает за получение ввода от пользователя и запуск команды, соответствующей вводу пользователя. Компонент считывания устройств ввода обращается к компоненту фабрики команд, для получения готовых конфигурация команд.

Компонент «фабрика команд» позволяет настраивать конфигурации команд и хранить их. Конфигурация команды происходит с помощью редактора Unity.

2.6. Макет интерфейса

Интерфейс основной сцены должен отображать на экране пользователя, поверх изображения виртуального мира, информацию о текущей задаче и о текущем состоянии характеристик робота. На рисунке 8 представлен макет интерфейса основной сцены.

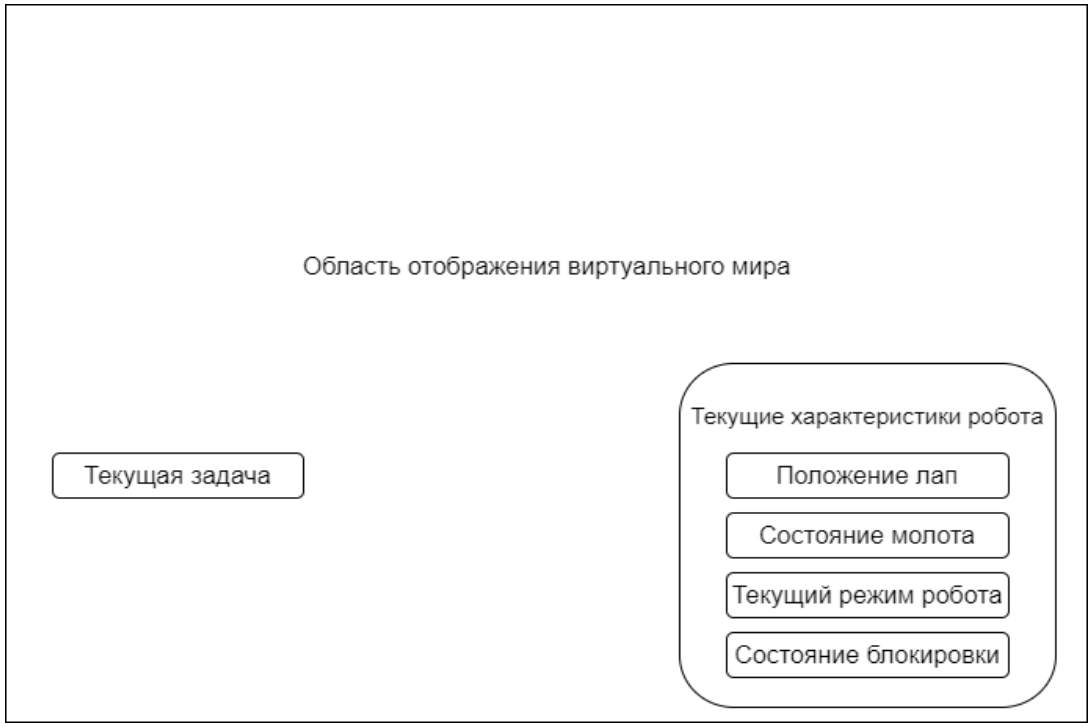

Рисунок 8 – Макет интерфейса основной сцены

Текущая задача отображает в текстовом виде задачу, которую пользователю необходимо сделать в текущий момент, также отображает прогресс выполнения задачи в текстовом виде, если такой имеется.

Текущие характеристики робота отображают значения характеристик робота в текущий момент времени.

Положение лап отображает текущее положение лап, они могут иметь следующие состояния: полностью опущены, промежуточное положение, полностью подняты. Состояние молота показывает работает молот в теку-

щие момент или нет. Текущий режим отображает в каком режиме сейчас находится робот: «Низ», «Вверх» или «Транспортировочный». Состояние блокировки показывает заблокирован ввод или нет.

Интерфейс окна завершения сценария должен предоставлять информацию о времени прохождения текущей сцены и предоставлять выбор пользователю, выйти в меню или начать текущую сцену заново. На рисунке 9 представлен макет интерфейса окна завершения сценария.

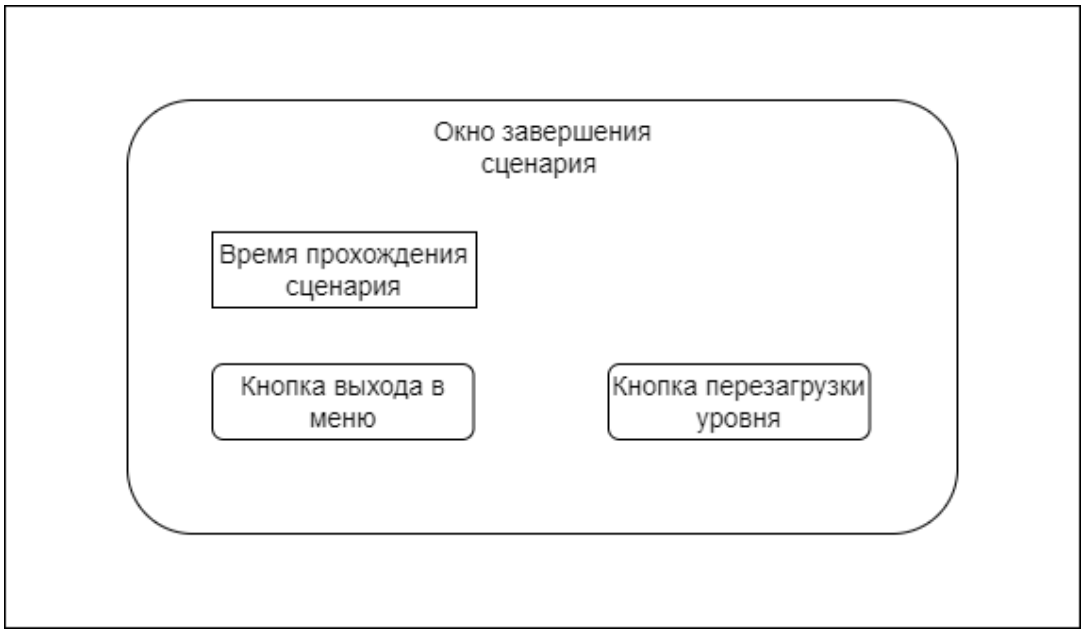

Рисунок 9 – Макет интерфейса окна завершения сценария

Когда пользователь выполняет последнюю задачу перед ним, поверх основной сцены, появляется окно завершения сценария. На нем отображается время прохождения выбранной сцены, кнопка выхода в меню, по нажатию которой пользователь переходит в меню выбора сцен, кнопка перезагрузки уровня, которая запускает текущую сцену заново.

Вывод по второму разделу

В данном разделе были представлены функциональные и нефункциональные требования симулятора демонтажного робота и архитектура основной сцены симулятора демонтажного робота. Показаны функциональные требования и архитектура компонентов управления сцены и ввода пользователя. Также продемонстрированы макеты интерфейсов.

3. РЕАЛИЗАЦИЯ КОМПОНЕНТОВ СИСТЕМЫ

3.1. Средства реализации

Для реализации приложения использован игровой движок Unity, для редактирования и создания кода использовался текстовый редактор Visual Studio Code и язык программирования C#.

Архитектура проекта Unity основана на шаблоне Entity Component System [12]. Согласно этому шаблону, приложение состоит из базовых сущностей (которые, например, хранят только свой уникальный id), функциональность которых расширяется с помощью специализированных компонентов.

Сцена (Scene) Unity состоит из игровых объектов (GameObject) с прикрепленными к ним компонентами (Component) [13]. Каждый игровой объект обязательно имеет собственный Transform, отвечающий за расположение объекта на сцене. Подобные объекты могут быть сохранены как «префабы» (Prefab).

Префабы – это особый тип файла, позволяющий хранить весь GameObject со всеми компонентами и значениями свойств. Префаб выступает в роли шаблона для создания экземпляров хранимого объекта в сцене [14]. Любые изменения в «префабе» немедленно отражаются и на всех его экземплярах.

Корутин (coroutine) – это блоки кода, которые работают асинхронно, то есть по очереди. В нужный момент исполнение такого блока приостанавливается с сохранением всех его свойств, чтобы запустился другой код. Когда управление возвращается к первому блоку, он продолжает работу [15].

Программирование в Unity заключается в первую очередь в разработке пользовательских классов, которые подключаются к игровым объектам как компоненты. Все такие классы должны наследоваться от класса MonoBehaviour [16]. Для обозначения классов-компонентов на диаграмме к их именам в конце добавлен указатель «Script».

В реализации компонента были использованы следующие фреймфорки.

1. Zenject – это фреймворк внедрения зависимостей с открытым исходным кодом для Unity, популярной платформы разработки игр. Он предоставляет способ управления зависимостями и реализации принципа инверсии управления в проектах Unity [17]. С помощью Zenject разработчики могут автоматизировать предоставления ссылок на игровые объекты, делая его более модульным, тестируемым и поддерживаемым.

2. UniRx – это библиотека реактивных расширений (RX) для Unity, предоставляющая способ работать с асинхронным и событийно-ориентированным программированием [18]. Она предоставляет удобный способ реализации отправки событий и их последующей обработки, позволяет обрабатывать сложные асинхронные операции, такие как ввод пользователя, сетевые запросы и события, в простом и композиционном стиле. С помощью UniRx разработчики могут использовать реактивное программирование [19] для создания отзывчивого и поддерживаемого кода.

3.2. Реализация компонента управления сценами

В компоненте задач представлена реализация классов `MissionManager`, `PlotScene`, которые являются менеджером сцен и сценарием на сценах, также представлена реализация классов задач сценария `MenuOfEnd`, `Area` и `TaskPlotDestruction`.

Реализация менеджера сцен

Класс `MissonManager` – менеджер сцен, он отвечает за переключение сцен в приложении. Он должен существовать в единственном экземпляре, и иметь возможность перемещаться между сценами, сохраняя свои данные. Zenject предоставляет объект `ProjectContext`, который является общим для всех сцен, перемещаться между сценами без уничтожения. В иерархии объектов `MissionManager` является дочерним элементом объекта `Pro-`

jectContext, что позволяет выполнить условия, представленные выше. На листинге 1 представлен фрагмент класса MissionManager.

Листинг 1 – Фрагмент реализации класса MissionManager

```
public  PlotScene currentPlot;
private CompositeDisposable _disposable = new CompositeDisposable();
public void Accept(AdapterPlotSceneWithMM adapter){
        adapter.Visit(this);
        currentPlot.PlotIsEnd
            .Subscribe(_ => {
                GoToMainMenu();
            })
            .AddTo(_disposable);
        Debug.Log(currentPlot.ID);
    }
```

В MissionManager был применен паттерн проектирования «Visitor» [20], при загрузке сцены, специальный класс «посетитель» устанавливает в MissionManager сценарий текущией сцены, затем менеджер сцен подписывается на предоставляемую UniRx «реактивную команду» PlotIsEnd, которая оповещает подписчиков о завершении сценария текущей сцены. В подписке MissionManager вызывает метод GoToMainMenu, который открывает сцену главного меню. Для загрузки сцен класс использует SceneManager предоставляемый Unity [21].

Реализация сценария

Класс PlotScene – является сценарием, расположен на каждой сцене и представляет собой набор упорядоченных заданий. Данный класс управляет последовательностью задач, выключая текущее задание, когда оно выполнено, и делает активным следующее в наборе. По завершению всех заданий в сценарии оповещает менеджер сцен. В листинге 2 предоставлен фрагмент реализации PlotScene.

Листинг 2 – Фрагмент реализации класса PlotScene

```
public ReactiveCommand PlotIsEnd = new ReactiveCommand();
private void Start()
    {
        foreach (var task in _TaskPlotArr){
            task.Deactivate();
        }
        _currentTask = CurrentTask;
        _currentTask.Activate();
        _currentTask.WhenTaskIsDone.Subscribe(_=>
                                TaskIsDone()).AddTo(_disposable);
```

```
    }
public void TaskIsDone()
    {
        _disposable.Clear();
        if (_IndexCurrTask >= _TaskPlotArr.Length - 1)
            PlotIsEnd.Execute();
        else
            NextTask();
    }
```

Во время старта сцены `PlotScene` включает первое задание и подписывается на «реактивную команду» `WhenTaskIsDone`, предоставляемую UniRx, оповещающую о завершении текущей задачи. Когда задача оповещает о своем завершении, вызывается метод `TaskIsDone`, где проверяется, является ли завершенная задача последней, если последняя, то класс `PlotScene` оповещает о завершении сценария, если задача не последняя, вызывается метод `NextTask`, который активирует следующую задачу.

Реализация задачи в сценарии

Все задачи наследуются от абстрактного класса `TaskPlot`. Он предоставляет абстрактный метод `Activate`, для написания логики, которую необходимо выполнить, когда данная задача становиться активной. Класс также предоставляет метод `Deactivate`, для написания логики, которую необходимо выполнить, когда данная задача становиться завершенной или неактивной. В листинге 3 предоставлен абстрактный класс `TaskPlot`.

Листинг 3 – Код абстрактного класса `TaskPlot`

```
public abstract class TaskPlot : MonoBehaviour
{
    public ReactiveCommand WhenTaskIsDone = new ReactiveCommand();
    public abstract void Activate();
    public abstract void Deactivate();
}
```

Класс содержит «реактивную команду», предоставляемую UniRX, оповещающую о завершении сценария текущей сцены.

Все задачи, добавляемые в сценарий, наследуются от `TaskPlot`. К данным задачам относятся: задача перемещения робота, задача завершения сценария, задачи по разрушению объектов.

Для обозначения в виртуальной среде задач, которые нужно выполнить, например, перемещение до указанной области используются специальные маркеры, в случае задачи перемещения это синяя зона с двигающейся над ней стрелкой, указывающей на эту зону.

Реализация задачи завершения сценария

Класс `MenuOfEnd` реализует последнюю задачу в сценарии, наследуется от класса `TaskPlot`. Данная задача должна присутствовать на каждой сцене последней для корректной работы компонента сценариев. Он представляет собой всплывающее окно, отображающее время прохождения сцены и кнопки возврата главного меню и перезагрузки уровня. В листинге 4 предоставлены код методов класса `MenuOfEnd`.

Листинг 4 –Кода методов класса `MenuOfEnd`

```
public void GoToMainMenu()
{
    base.WhenTaskIsDone.Execute();
}
public override void Activate()
{
    Cursor.visible = true;
    Cursor.lockState = CursorLockMode.None;
    this.gameObject.SetActive(true);
    camera_Rotation.Pause(true);
    Time.timeScale = 0;
    ActivateMenu.Execute();
}
```

Функция `GoToMainMenu` передается в кнопку, отвечающую за переход в главное меню. Данная функция оповещает систему сценариев о завершении, а так как задача последняя, система сценариев передаст сообщение менеджеру сцен о завершении сценария.

Функция `Activate` разблокирует и делает видимым курсор мыши, камера фиксируется в последнем положении, интерактивное пространство останавливается, а также делает видимым окно завершения сценария.

Реализация задачи перемещения

Для реализации тестовых сценариев использования симулятора, в которых проверяется управление роботом, требуется реализация задачи перемещения робота до указанной области.

Класс Area является задачей по перемещению демонтажного робота к выделенной зоне (точке) перемещения и является наследником класса TaskPlot. Она должна быть невидимой пока до нее не дойдет очередь. После того как наступит ее очередь она становиться видимой.

Задача перемещения имеет в виртуальной среде представление в виде синей зоны с движущейся стрелкой над ней, которая указывает на эту зону. Движение стрелки реализовано при помощи «корутина», который включается при активации объекта и выключается при завершении задачи. В «корутин» передается метод, который каждый кадр изменяет положение стрелки между двумя точками. Объект задачи является «префабом», что позволяет многократно ее использовать.

Задача считается выполненной, когда все датчики, коллайдеры со специальным тэгом, на роботе находиться в зоне триггера одновременно. На листинге 5 предоставлена часть кода класс Area.

Листинг 5 – Часть кода класса Area

```
void Start()
{
_tagArray = new string[] {"LFDetected", "RFDetected", "LBDetected",
  "RBDetected"};

_colider = GetComponent<Collider>();
colider.OnTriggerEnterAsObservable()
       .Where(collider =>Array.Exists(_tagArray,(x)=> collider.tag == x))
       .Subscribe( _ => {
           _count++;
           if (_count == 4)
                {
                    WhenTaskIsDone.Execute();
                    _disposable.Clear();
                 }
       })
       .AddTo(_disposable);

_colider.OnTriggerExitAsObservable()
    .Where(collider => Array.Exists(_tagArray, (x) => collider.tag == x))
    .Subscribe(_ => {
        _count--;
    })
    .AddTo(this);
}
```

Функция OnTriggerEnterAsObservable добавляет к объекту, представляющему зону, дополнительный компонент, который обрабатывает

начало пересечения с другими объектами, считаются только объекты с необходимым тэгом. Функция `OnTriggerExitAsObservable` работает аналогично, только добавляемы компонент вызывается, когда пересечение с объектом, имеющим нужный тэг, оканчивается. Если компонент покидает зону счетчик уменьшается. Когда все объекты будут в зоне и счетчик станет равен количеству необходимых тэгов, задача завершается, и перестанет быть видимой.

Реализация задачи разрушения

Класс `TaskPlotDestruction` – представляет задачу по разрушению и является наследником класса `TaskPlot`. Данная задача отслеживает процент разрушения объекта и завершает задачу, когда целостность объекта падает, до необходимого уровня. На листинге 6 предоставлены часть кода класса `TaskPlotDestruction`.

Листинг 6 – Часть кода класса `TaskPlotDestruction`

```
public override void Activate()
{
    RootDestruct.Inizialization(NeedProcent, WhenTaskIsDone);
    GUI_Task.Activate();
}

public override void Deactivate()
{
    RootDestruct.Clear();
    GUI_Task.Deactivate();
}
```

В методе `Activate` разрушаемому объекту передается процент целостности, при достижении, которого необходимо активировать «реактивную команду» `WhenTaskIsDone`, для уведомления что задача завершена, также включает пользовательский интерфейс данной задачи.

В методе `Dectivate` разрушаемый объект очищается от «реактивной команды», выключает пользовательский интерфейс данной задачи.

3.3. Реализация компонента ввода пользователя

Реализация компонента считывания ввода представлена классами `KeyInputHandler`, `JoystickInputHandler`, классом переключения

устройств ввода `SwitchInputHandler`, классом фабрики комманд `FabricRotateCommand` и абстрактным классом команды `InputRotateCommand` с его наследниками.

Реализация компонента считывания устройств ввода

Считывание ввода пользователя контролируют классы, наследованные от абстрактного класса `InputHandler`. Каждый такой класс отвечает за считывание ввода пользователя на конкретном устройстве, и вызов команды, соответствующей запросу пользователя. Класс `KeyInputHandler` отвечает за считывание с клавиатуры, а `JoystickInputHandler` за считывание с устройства имитирующего настоящий контроллер демонтажного робота. Классы должны быть одинаковыми в возможностях управления роботом, но клавиатура должна иметь дополнительные кнопки для переключения устройства ввода, вызова меню, реализации инструментов разработчика. На рисунке 10 представлена диаграмма классов компонента считывания ввода.

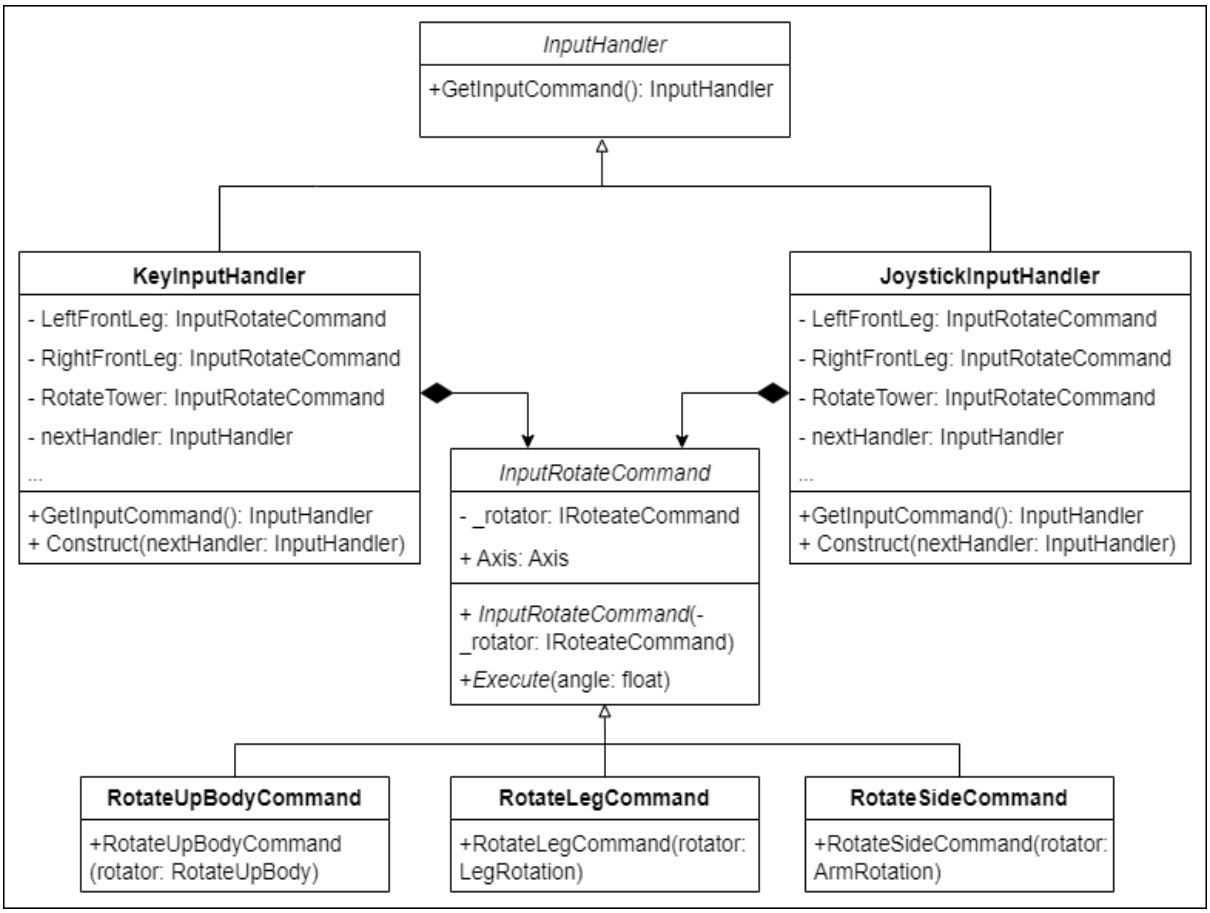

Рисунок 10 – Диаграмма классов компонента считывания ввода

`InputHandler` сделан абстрактными классом, а не интерфейсом, т.к. Unity не способен сериализовывать интерфейсы, из-за чего возникают проблемы с указанием ссылки на классы, реализовывающие интерфейсы, в редакторе.

Классы `KeyInputHandler` и `JoystickInputHandler` содержат команды, которые они вызывают в зависимости от ввода пользователя. Это могут быть команды движения секции стрелы, башни, гусеницы и т.п.

Класс `KeyInputHandler` считывает ввод пользователя с клавиатуры. В конструкторе класса происходит получение ссылки на другой класс считывания устройства ввода пользователя при помощи фреймворка Zenject. На листинге 7 представлен метод установки связей для фреймворка Zenject класса `KeyInputHandler`.

Листинг 7 – Код метода установки связей класса `KeyInputHandler`

```
[Inject]
public void Construct([Inject(Id = "KeyInputHandler")]InputHandler handler)
{
    _nextHandler=handler;
}
```

Атрибут Inject помечает данный конструктор для контейнера связей фреймворка Zenject, чтобы при старте сцены контейнер связей предоставил автоматически информацию о другом классе устройства считывания. `InputInstaller` является классом фреймворка Zenject и устанавливает, что необходимо передать в методы помеченные атрибутом `Inject`. На листинге 8 представлен фрагмент класса `InputInstaller`.

Листинг 8 – Фрагмент класса `InputInstaller`

```
public class InputInstaller : MonoInstaller
{
    [SerializeField] JoystickInputHandler JoystickHandler;
    ...
    public override void  InstallBindings()
    {
        Container.Bind<InputHandler>().WithId("KeyInputHandler")
            .To<JoystickInputHandler>().FromInstance(JoystickHandler)
            .AsSingle();
    }
    ...
}
```

При старте сцены Zenject автоматически предоставит экземпляр класса `JoystickInputInstaller`, в данном случае объект `JoystickHandler`, всем методам помеченным атрибутом Inject с идентификатором «`KeyInputHandler`».

После установки всех связей класс считывания ввода пользователя получает команды у объекта класса фабрики команд, которые будет запускать при получении необходимого ввода пользователя. На листинге 9 представлен фрагмент метода `Start`.

Листинг 9 – Фрагмент метода `Start`

```
void Start()
{
    RFLegRCommand = FabricRotate.CreateRFLegCommand();
    LFLegRCommand = FabricRotate.CreateLFLegCommand();
    RBLegRCommand = FabricRotate.CreateRBLegCommand();
    LBLegRCommand = FabricRotate.CreateLBLegCommand();

    _arrLegSkript = new LegRotationCommand[] { RFLegRCommand,
                    LFLegRCommand, RBLegRCommand, LBLegRCommand };
}
```

У класса `FabricRotate` запрашиваются команд с готовой конфигурацией, данные команды имеют информацию о исполнителе запроса, или метод исполняющий запрос, и дополнительная информация, если таковая имеется.

`GetInputCommand` класса `InputHandler`, считывает ввод пользователя и запускает соответствующую команду. На листинге 10 представлен фрагмент метода `GetInputCommand`.

Листинг 10 – Фрагмент метода `GetInputCommand`

```
public override InputHandler GetInputCommand()
{
    if (Input.GetKeyDown(KeyCode.RightBracket)){
        return _nextHandler;
    }
    if(Input.GetButton("Back") || Input.GetButton("L2"))
        BumperCommand.Execute(0);
    foreach (var legCommand in _commandsLeg){
        legCommand(Input.GetAxis);
    }
    ...
    return null;
}
```

Если пользователь запросил смену устройства ввода, то метод GetInputCommand вернет ссылку на следующий объект считывания устройства ввода, который был передан во время старта сцены. Если пользователь запросил движение демонтажного робота, будет выполнена конкретная команда.

Реализация компонента переключения устройств ввода

Класс SwitchInputHandler – компонент переключения устройств ввода, он отвечает за управление и переключения устройствами считывания ввода пользователя. Он вызывает каждый кадр метод GetInputCommand у текущего объекта считывания устройств ввода, чтобы переключить устройство ввода, когда это запрашивает пользователь. На листинге 11 представлен код вызова метода GetInputCommand.

Листинг 11 – Код вызова GetInputCommand

```
public void Update()
{
    InputHandler handler = _currentHandler.GetInputCommand();
    if(handler != null){
        _currentHandler = handler;
    }
}
```

Если пользователь запрашивает переключить устройство ввода, то метод GetInputCommand вернет ссылку на объект следующего устройства ввода и сделает данный объект текущим.

Реализация компонента команд

Абстрактный класс InputRotateCommand является командой поворота, он содержит ссылку на объект исполнитель команды, ось вокруг которой необходимо вращаться, поле Axis и абстрактный метод Execute, который вызывается классами считывания ввода, для исполнения команды. От класса InputRotateCommand наследуются классы RotateUpBodyCommand – команда поворота башни, RotateLegCommand – команда поворота ног и RotateSideCommand – команда поворота стрелы.

Реализация компонента фабрики команд

Класс `FabricRotateCommand` – представляет класс компонента фабрики команд позволяет настраивать конфигурации команд и хранить их. Он предоставляет возможность настроить все параметры команд в редакторе Unity, и затем получить сконфигурированные команды во время старта сцены или во время симуляции. На рисунке 11 представлен интерфейс фабрики команд в редакторе.

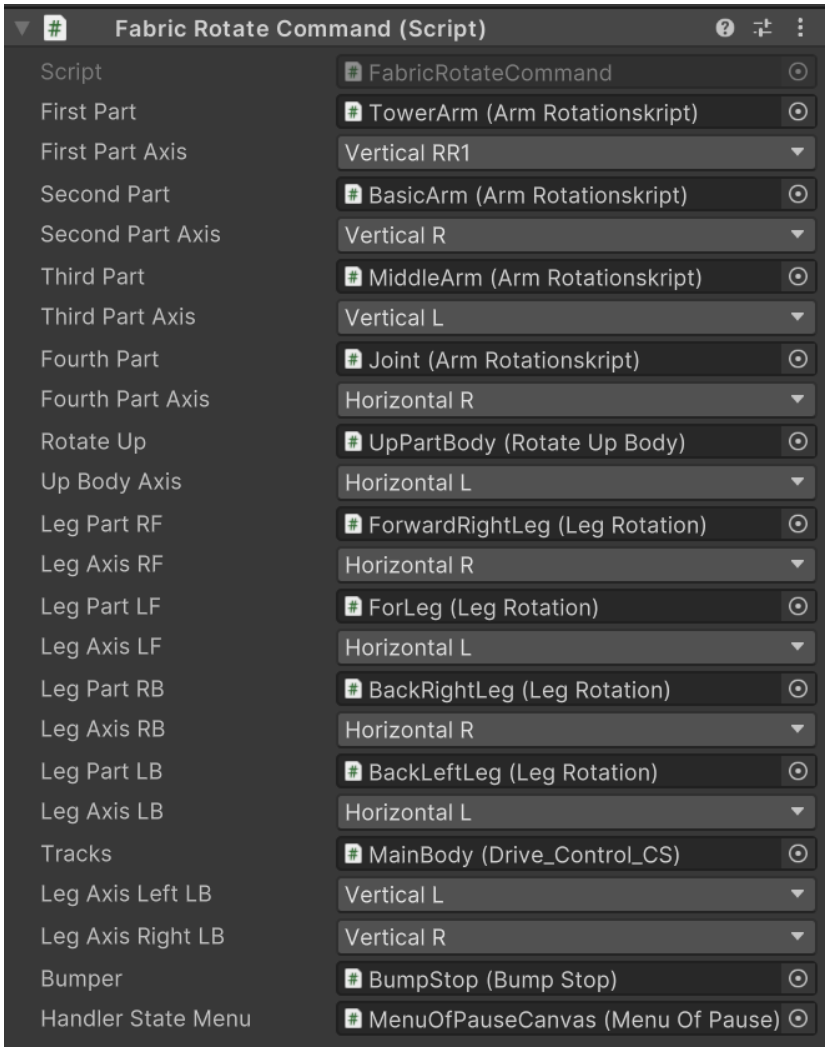

Рисунок 11 – Интерфейс фабрики команд в редакторе

Для каждой команды можно передать ссылку на исполнителя и дополнительные параметры, если имеются, например, ось вращения башни демонтажного робота.

3.4. Реализация компонентов демонтажного робота

Необходимо реализовать поворот компонента башни демонтажного робота, поворот стрелы (руки) демонтажного робота и посредника между текущими характеристиками робота и интерфейсом.

Компоненты башни и стрелы, обладают таймером блокировки. Он отвечает за блокировку управления, если демонтажный робот бездействует больше 5 секунд. Таймер сбрасывается, если не прошло больше 5 секунд и была получена команда о совершении действия демонтажным роботом. Если робот заблокировался, для разблокировки необходимо нажать соответствующую клавишу (кнопку) на текущем устройстве ввода.

Компоненты демонтажного робота, которые реализовывают поворот элементов робота и поворот может быть вызван из стороннего кода, реализовывают интерфейс `IRotateCommand`. Интерфейс содержит единственный метод Execute, который принимает значение, на которое нужно повернуть компонент.

Реализация поворота башни

Поворот башни реализуется в классе `RotateUpBody`, который реализовывает интерфейс `IRotateCommand`. В зависимости от режима работы демонтажного робота поворот башни будет осуществлен или нет. На листинге 12 представлен код метода Execute класса `RotateUpBody`.

Листинг 12 – Код метода `Execute` класса `RotateUpBody`

```
public void Execute(float axisValue = 0)
{
    if ((mode == (int)ModeRobot.Mode.Up
    || mode == (int)ModeRobot.Mode.Transport) && !isBlock){
     speed = speedRotation * axisValue;
    }
    if (axisValue != 0) _timer.ResetTimer();
    transform.localRotation *= Quaternion.Euler(0, 0,
        _scaleFactor * speed * Time.deltaTime);
    speed = 0f;
}
```

Поворот будет осуществлен, если демонтажный робот работает в режиме «Вверх» или «Транспортировочный». В режиме «Вверх» пользователь управляет башней, рукой демонтажного робота и отбойным молотком,

в данном режиме производиться работы по разрушению объектов. В режиме «Транспортировочный» робот вместо работы отбойным молотком, управляет передвижением робота вперед и назад, данный режим используется при спуске с транспортировочных платформ. Затем сбрасывается таймер блокировки, чтоб отсчет времени бездействий пошел сначала. После происходит преобразование поворота из градусов в кватернион [22] и умножается на кватернион, представляющий текущий поворот башни, для получения конечного поворота башни.

Реализация движения стрелы

Рука состоит из 3 секций, первая секция – секция, выходящая из башни, третья – оборудована отбойным молотом и вторая – соединяет первую и третью. Все секции имеют собственные максимальные углы наклона и движение каждой секции не зависит от других. Поворот секции стрелы реализовывает класс `ArmRotationSkript`, который реализовывает интерфейс `IRotateCommand`. В зависимости от режима работы демонтажного робота поворот секций включен или выключен. На листинге 13 представлен код метода Execute класса `ArmRotationSkript`.

Листинг 13 – Код метода Execute класса `ArmRotationSkript`

```
public void Execute(float axisValue = 0)
{
    angle = transform.localEulerAngles.y;
    angle = Mathf.Repeat(angle + 180, 360) - 180;
    if ((angle < maxAngle || angle > minAngle)
    && (mode == (int)ModeRobot.Mode.Up
    || mode == (int)ModeRobot.Mode.Transport)
    && !isBlock){
        speed = maxSpeed * axisValue * _scaleFactor;
        if (angle > maxAngle) speed = Mathf.Clamp(speed, -1000, 0);
        else if (angle < minAngle) speed = Mathf.Clamp(speed, 0, 1000);
    }
    if (axisValue != 0) _timer.ResetTimer();
    transform.localRotation *=
    Quaternion.Euler(0,speed * Time.deltaTime,0);
    speed = 0;
 }
```

Сначала происходит проверка, что текущий угол поворота секции не больше максимально допустимого угла. Поворот секции будет осуществлен если демонтажный робот работает в режиме «Вверх» или «Транспор-

тировочный». Затем сбрасывается таймер блокировки, чтоб отсчет времени бездействий пошел сначала. Поворот секции стрелы осуществляется аналогично повороту башни.

Реализация посредника между состояниями робота и интерфейсом

Пользователю необходимо выводить информацию о текущем состоянии характеристик робота на экран в текстовом виде при помощи интерфейса. Логика многих деталей робота запрограммирована при помощи математических терминов, например, информация о текущем повороте компонента опор (лап) содержится в виде кватерниона. Информацию о положении опор демонтажного робота нужно выводить в текстовом виде: «лапы опущены», «лапы подняты» или «лапы в промежуточном положении». Чтобы не привязывать интерфейс к каждому компоненту робота, создан промежуточный класс, где будет происходить отслеживание состояний робота, преобразование данных и предоставление информации при помощи поставляемых фреймворком UniRx «реактивных свойств». Они хранят значения и оповещают подписчиков о изменении своего значения.

Реализация отслеживаемого компонента

Классы, которые должны отслеживаться, реализуют шаблонный интерфейс `ITrackableState`. При его реализации класс должен создать «реактивное свойство», на которое промежуточный класс должен подписаться, чтобы отслеживать изменения. Также класс должен предоставлять максимальное и минимальное значение отслеживаемого параметра, например: чтобы знать минимальный и максимальный угол подъема опор робота. На листинге 14 представлен код интерфейса `ITrackableState`.

Листинг 14 – Код интерфейса `ITrackableState`

```
public interface ITrackableState<T>
{
    IReadOnlyReactiveProperty<T> TrackState { get; }
    T MaxValue { get; }
    T MinValue { get; }
}
```

Обертка `IReadOnlyReactiveProperty` для реактивного свойства нужна для предотвращения ситуаций, когда внешний код хочет изменить значение, обертка предоставляет только чтение свойства.

Реализация посредника между характеристиками робота и пользовательским интерфейсом

Класс `StateOfDemRobot` является посредником между характеристиками робота и пользовательским интерфейсом, предоставляет интерфейс для получения текущих характеристик робота. На листинге 15 представлена часть кода класса `StateOfDemRobot`.

Листинг 15 – Часть кода класса `StateOfDemRobot`

```
public IReadOnlyReactiveProperty<LegState> LeftSideLegUp =>_leftSideLegUp;
public IReadOnlyReactiveProperty<LegState> RightSideLegUp
        =>_rightSideLegUp;

public IReadOnlyReactiveProperty<bool> AllSideLegUp => _allSideLegUp;
public IReadOnlyReactiveProperty<bool> AllSideLegDown => _allSideLegDown;

private ITrackableState<float> _leftSideLeg;
private ITrackableState<float> _rightSideLeg;

private ReactiveProperty<LegState> _leftSideLegUp = new
        ReactiveProperty<LegState>(LegState.LegDownState);
private ReactiveProperty<LegState> _rightSideLegUp = new
        ReactiveProperty<LegState>(LegState.LegDownState);
```

Класс `StateOfDemRobot` позволяет получить данные о положении каждой опоры (лапы), общего положения ног. Получение ссылок на объекты, за параметрами которых необходимо следить, происходит при помощи инъекции зависимости Zenject, так как класс не является наследником `Monobehaviour`, инъекцию можно сделать при помощи конструктора. На листинге 16 представлена часть кода класса `StateOfDemRobot`.

Листинг 16 – Конструктор класса `StateOfDemRobot`

```
    Public StatesOfDemRobot(
    [Inject(Id = "LeftLegSide")] ITrackableState<float> leftSide,
    [Inject(Id = "rightLegSide")] ITrackableState<float> rightSide,
    ...,
    [Inject(Id = "Bumper")] ITrackableState<bool> bumper){
        _leftSideLeg = leftSide;
        _rightSideLeg = rightSide;
        ...
        _bumper = bumper;
    }
```

После конструктора вызывается метод инициализации, для подписки на отслеживаемые характеристики. На листинге 17 представлен код метода `Initialize` класса `StateOfDemRobot`.

Листинг 17 –Кода метода `Initialize`

```
public void Initialize(){
    _disposable = new CompositeDisposable();
    TrackLeftSideLegs.Subscribe(angle => {
        if (angle > _leftSideLeg.MaxValue - 1f){
           _leftSideLegUp.Value = LegState.LegUpState;
        }
        else if (angle < _leftSideLeg.MinValue + 1f){
            _leftSideLegUp.Value = LegState.LegDownState;
        }else{
            _leftSideLegUp.Value = LegState.IntermediateStateLeg;}
    }).AddTo(_disposable);}
```

3.5. Демонстрация работы реализованных компонентов

Демонстрация работы компонентов демонтажного робота будет происходить на сцене тестового полигона, на котором расположено задание по перемещению и разрушению. После загрузки сцены тестового полигона пользователь видит демонтажного робота с поднятыми лапами, снизу слева отображается в текстовом виде задание, которое необходимо выполнить, снизу справа показано в текстовом виде текущее состояние характеристик демонтажного робота. На рисунке 12 показана стартовая сцена с тестовым полигоном после загрузки.

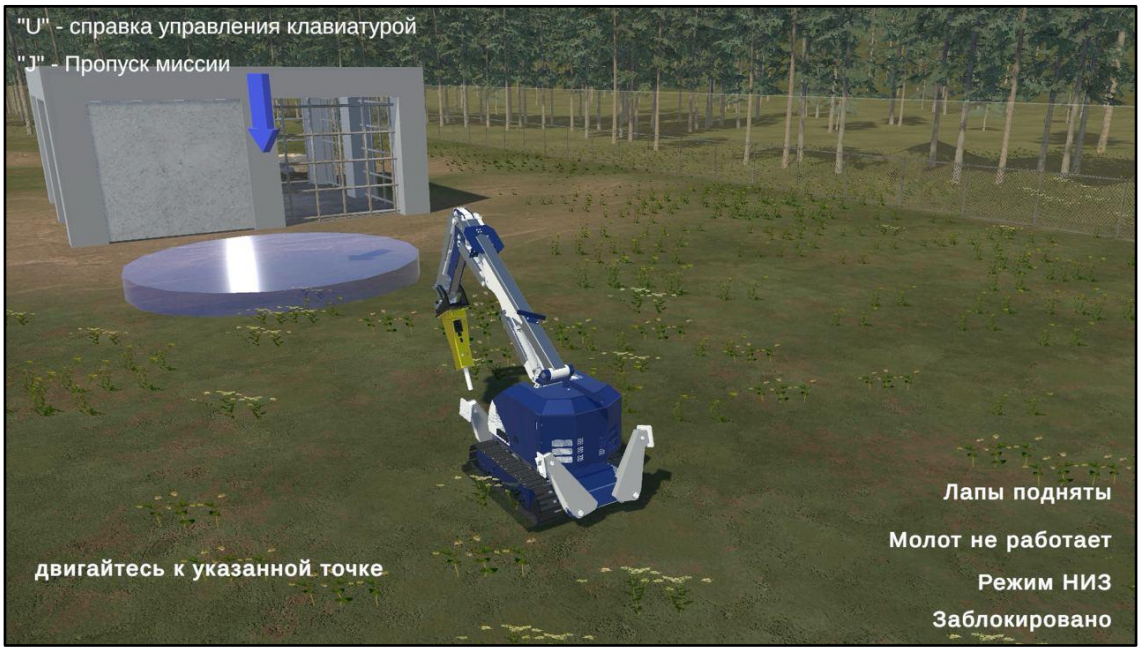

Рисунок 12 – Сцена с тестовым полигоном после загрузки

На рисунке 12 слева снизу изображено текущее задание, двигаться к указанной точке (зоне), зона изображена синей областью и двигающейся стрелкой над ней. Слева снизу изображено, что в текущий момент лапы подняты, молот (отбойный) не работает, включен режим «Низ», управление роботом заблокировано. Режим «Низ» используется для перемещения робота, в нем управляются только гусеницы.

Для передвижения к указанной зоне разблокируем управление и начнем перемещение к центру указанной зоны, пока задание не завершиться и не смениться другим. После достижения и въезда в отмеченную зону задание выполняется, и снизу слева текстовое задание меняется на «процент уничтожения». Для начала разрушения (уничтожения) опускаем лапы (опоры) робота вниз, пока не появиться надпись «лапы опущены», после переведем робота в режим «Вверх» для управления стрелой и башней демонтажного робота. На рисунке 13 изображена сцена с результатом разрушения стены по зданию.

Рисунок 13 – Сцена с результатом разрушения стены по заданию

На интерфейсе изображено, что лапы опущены, молот не работает, включен режим «Вверх», так как сейчас управляем башней и рукой демонтажного робота, управление заблокировано, процент разрушения равен 95.

После разрушения объекта задача завершается и, так как оно было последним, активируется задача завершения сценария. Появляется окно завершения сценария. На рисунке 14 представлено окно завершения сценария.

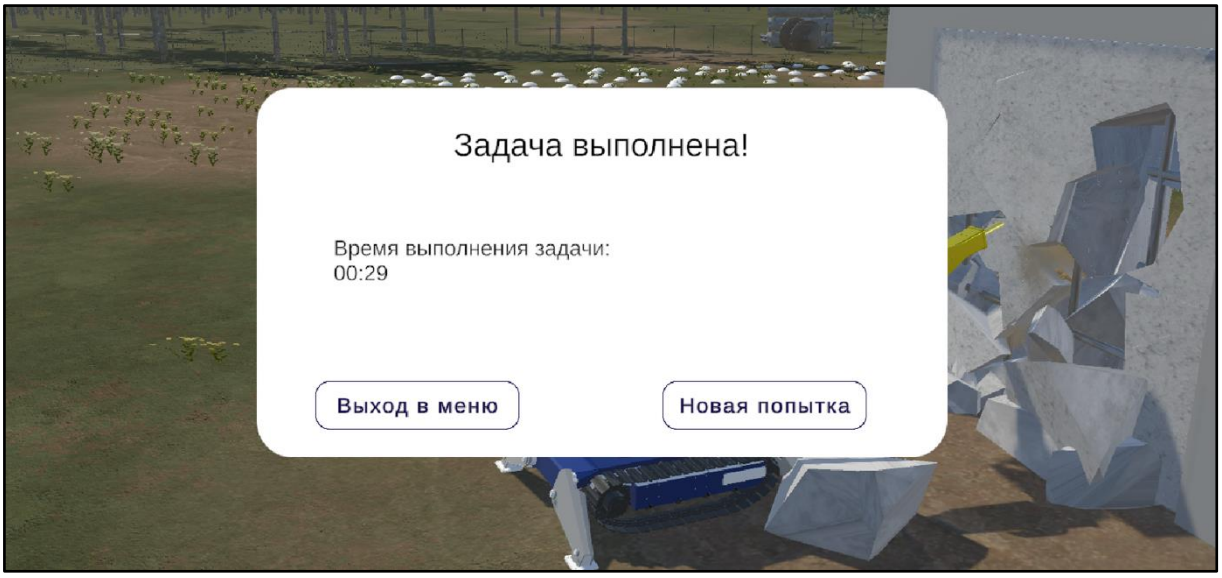

Рисунок 14 – Окно завершения сценария

Окно завершения сценария содержит отображения количества времени, потребовавшегося для прохождения сцены, кнопку выхода в меню выбора уровней, кнопку перезапуска текущего уровня. Камера не меняет положение при движении мышью, появляется курсор мышки.

Вывод по третьему разделу

В данном разделе были показаны средства реализации, показана реализация компонента управления сценами, компонента пользовательского ввода и реализация компонентов демонтажного робота. Была продемонстрирована работа компонентов выполнения заданий тестовом полигоне в симуляторе демонтажного робота.

4. ТЕСТИРОВАНИЕ

Для тестирования компонентов применялось функциональное тестирование, т. е. тестирование программного обеспечения в целях проверки реализованных компонентов.

Сцена полигона для тестирования

Для тестирования компонентов создана сцена с полигоном для тестирования. Она представляет прямоугольную область, где расставлены объекты для тестирования передвижения демонтажного робота, работы стрелы и отбойного молота разрушаемых объектов и компонентов ввода. Для испытания гусениц создана область с выступающими из земли белыми объектами. Для тестирования работы стрелы и разрушения имеется объект в виде белой постройки с одной разрушаемой стеной. Для проверки работы компонента управления сцен на сцене присутствует задача по перемещению демонтажного робота и задача завершения сцены.

Тестирование компонента управления сценами

Тестирование управления сценами проводилось вручную на специальной сцене, являющейся полигоном для тестирования. Результаты представлены в таблице 1.

Таблица 1 – Тестирование работы компонента задач в симуляторе

№	Название теста	Последовательность действий	Полученный результат	Тест пройден?
1.	Переход в главное меню по завершению всех задач на текущей сцене.	Запустить сцену, выполнить все задания, нажать на кнопку перехода в главное меню.	Переход в главное меню.	Да
2.	Выбор любой сцены из главного меню.	В главном меню выбрать любую сцену нажав на ее картинку, нажать на кнопку запуска сцены.	Загрузка выбранной сцены.	Да
3.	Завершение всех задач в стандартном порядке сценария.	Выполнить все задачи, нажать на кнопку перехода в главное меню в окне завершения сценария.	Завершение сцены и переход в главное меню.	Да

Окончание таблицы 1

№	Название теста	Последовательность действий	Полученный результат	Тест пройден?
4.	Завершение всех задач в обратном порядке сценария.	Выполнить все задачи, нажать на кнопку перехода в главное меню в окне завершения сценария.	Завершение сцены и переход в главное меню.	Да
5.	Размещение заданий в нестандартном порядке в сценарии.	Выполнить все задачи, нажать на кнопку перехода в главное меню в окне завершения сценария.	Завершение сцены и переход в главное меню.	Да
6.	Проверка стабильности выполнения задачи перемещения.	Заехать в указанную зону не полностью, выехать из нее, повторить предыдущие шаги несколько раз, заехать полностью.	Задание будет выполнена, когда робот заедет полностью в указанную зону последний раз.	Да
7.	Проверка фиксирования степени разрушения.	Частично разрушить камень во время выполнения другой задачи, выполнить все задачи до задачи, камень которой разрушили.	Задача хранит степень разрушения камня, который частично разрушили вне очереди задач.	Да

Тестирование пользовательского ввода

Тестирование компонента пользовательского ввода проводилось вручную в меню и на специальной сцене, являющейся полигоном для тестирования. Результаты представлены в таблице 2.

Таблица 2 – Тестирование работы компонента пользовательского ввода в симуляторе

№	Название теста	Последовательность действий	Полученный результат	Тест пройден?
1.	Переключения устройства ввода.	Нажать на клавишу переключения устройства ввода.	Текущим считывающим устройством станет контроллер.	Да
2.	Работа мыши при любом устройстве ввода.	Осмотреться при помощи мыши, переключить устройство ввода, повторить первый шаг.	Мышь работает при любом устройстве ввода.	Да
3.	Открытие меню паузы, когда клавиатура текущее устройство ввода.	Выбрать клавиатуру как текущее устройство ввода, нажать на кнопку, вызывающую меню паузы.	Откроется меню паузы.	Да

Окончание таблицы 2

№	Название теста	Последовательность действий	Полученный результат	Тест пройден?
4.	Открытие меню паузы, когда контроллер текущее устройство ввода.	Выбрать контроллер как текущее устройство ввода, нажать на кнопку, вызывающую меню паузы на клавиатуре.	Откроется меню паузы.	Да
5.	Работа правого стика на контролере.	Наклонять правый в различные стороны.	Движение стрелы башни.	Да
6.	Отсутствие реакции робота на нажатие клавиш устройства ввода не являющимся текущим.	Переключить устройство ввода, на предыдущем устройстве ввода нажать на клавишу движения.	Робот не будет двигаться.	Да

Тестирование реализованных компонентов демонтажного робота

Для тестирования реализованных компонентов демонтажного робота применялось функциональное тестирование. Тестирование проводилось вручную на специальной сцене, являющейся полигоном для тестирования. Результаты представлены в таблице 3.

Таблица 3 – Тестирование работы представителя между состояниями робота и интерфейса

№	Название теста	Последовательность действий	Полученный результат	Тест пройден?
1.	Подъем всех секций стрелы демонтажного робота на максимальный угол.	Поднять первую секцию на максимальный угол, поднять вторую секцию на максимальный угол, поднять третью секцию на максимальный угол.	Рука демонтажного робота больше не будет подниматься.	Да
2.	Приподнять демонтажный робот при помощи стрелы.	Расположить руку перед демонтажным роботом так, чтобы отбойный молоток был перпендикулярен поверхности, опускать первую секцию стрелы.	Передняя часть демонтажного робота подниматься над землей.	Да
3.	Отображение блокировки.	Не осуществлять ввод на протяжении 5 секунд.	Отобразиться, что блокировка включена.	Да

Окончание таблицы 3

№	Название теста	Последовательность действий	Полученный результат	Тест пройден?
4.	Отображение работы молотка.	Нажать на клавишу, отвечающую за работу молотка.	Отобразиться, что молот включен.	Да
5.	Отображение поднятого положения лап.	Полностью поднять все опоры.	Отобразиться надпись, что лабы подняты.	Да
6.	Отображение промежуточного положения лап.	Поставить опоры в любые разные положения.	Отобразиться надпись, что лабы в промежуточном состоянии.	Да
7.	Отображение переключения режимов робота.	Переключить демонтажного робота в режим «Низ», затем в режим «Вверх».	Отобразиться надпись, что робот переключился в режим «Низ», затем в режим «Вверх».	Да

Вывод по четвертому разделу

В данном разделе были представлены результаты тестирования реализованных компонентов демонтажного робота, работа интерфейса с посредником между характеристиками робота и интерфейсом, компонента управления сценами и компонента считывания ввода пользователя. Было разработано 20 тестов.

ЗАКЛЮЧЕНИЕ

В данной работе были разработаны программные компоненты для симулятора демонтажного робота на платформе Unity.

Основные результаты

1. Выполнен анализ предметной области.
2. Спроектированы программные компоненты.
3. Реализованы программные компоненты.
4. Проведено тестирование.

Направления дальнейших исследований

Дальнейшим направлением работы будет создание сцен, предоставляющих новые ситуации работы с демонтажные роботом, добавление новых задач для компонента управления сценами, улучшение взаимодействия демонтажного робота с интерактивной средой, добавление звукового сопровождения и анимации.

ЛИТЕРАТУРА

1. Индустрия 4.0 – 4-я промышленная революция – Neftegaz. [Электронный ресурс] URL: https://neftegaz.ru/science/Oborudovanie-uslugi-materialy/331581-industriya-4-0-4-ya-promyshlennaya-revolyutsiya/ (дата обращения: 07.02.2024 г.).

2. Преимущества применения тренажёров-симуляторов для подготовки специалистов по опасным профессиям || Разработка компьютерных тренажеров, симуляторов, VR-тренажеров, электронных курсов – SIKE, САЙК. [Электронный ресурс] URL: https://sike.ru/advantages-of-using-simulators (дата обращения: 08.02.2024 г.).

3. Использование симуляторов в подготовке и оценке квалификации операторов печатного оборудования. [Электронный ресурс] URL: https://compuart.ru/article/25502 simulators (дата обращения: 08.02.2024 г.).

4. Operator Training Simulator Market Size – GMI. [Электронный ресурс] URL: https://www.gminsights.com/industry-analysis/operator-training-simulator-market (дата обращения: 08.02.2024 г.).

5. Профессиональные тренажерные симуляторы – Operex. [Электронный ресурс] URL: https://operex.ru/trenazhery-simulyatory/ (дата обращения: 08.02.2024 г.).

6. Симуляторы бульдозера – thoroughtec. [Электронный ресурс] URL: https://clck.ru/38k3C9 (дата обращения: 09.02.2024 г.).

7. Construction Simulator. [Электронный ресурс] URL: https://www.bau-simulator.de/en/ (дата обращения: 09.02.2024 г.).

8. Sim Excavator 2022 – Sim Craft. [Электронный ресурс] URL: https://www.tngsim.ru/shop/100/desc/soft-simulator-excavator (дата обращения: 09.02.2024 г.).

9. Учебный тренажер-симулятор «ЧЕТРА». [Электронный ресурс] URL: https://service-im.ru/training/oborudovanie-dlya-obucheniya.php (дата обращения: 09.02.2024 г.).

10. UI Toolkit – Unity. [Электронный ресурс] URL: https://unity.com/ru/features/ui-toolkit (дата обращения: 14.04.2024 г.).

11. Найстром Р. Шаблоны игрового программирования. // Издательство Бомбора, 2022. – № 2 – С. 26 – 38.

12. Entity Component System: An Introductory Guide – simplilearn. [Электронный ресурс] URL: https://www.simplilearn.com/entity-component-system-introductory-guide-article (дата обращения: 14.04.2024 г.).

13. Scenes – Unity Manual. [Электронный ресурс] URL: https://docs.unity3d.com/Manual/CreatingScenes.html (дата обращения: 14.04.2024 г.).

14. Prefabs – Unity Manual. [Электронный ресурс] URL: https://docs.unity3d.com/Manual/Prefabs.html (дата обращения: 14.04.2024 г.).

15. Coroutines – Unity Manual. [Электронный ресурс] URL: https://docs.unity3d.com/Manual/Coroutines.html (дата обращения: 14.04.2024 г.).

16. Unity – Scripting API: MonoBehaviour. [Электронный ресурс] URL: https://docs.unity3d.com/ScriptReference/MonoBehaviour.html (дата обращения: 14.04.2024 г.).

17. Zenject – Github. [Электронный ресурс] URL: https://github.com/modesttree/Zenject (дата обращения: 14.04.2024 г.).

18. UniRx – Github. [Электронный ресурс] URL: https://github.com/neuecc/UniRx (дата обращения: 14.04.2024 г.).

19. What Is Reactive Programming? – Baeldung. [Электронный ресурс] URL: https://www.baeldung.com/cs/reactive-programming (дата обращения: 14.04.2024 г.).

20. Тепляков С. Паттерны проектирования на платформе .NET // СПб.: Питер, 2015. - № 3 – С. 188 – 196.

21. Unity – Scripting API: SceneManager. [Электронный ресурс] URL: https://docs.unity3d.com/ScriptReference/SceneManagement.SceneManager.html (дата обращения: 02.05.2024 г.).

22. Математика Unity (Урок 17): Quaternion – poqxert. [Электронный ресурс] URL: https://poqxert.ru/blog/unity/tutorials/matematika/matematika-unity-urok-17-quaternion (дата обращения: 03.05.2024 г.).