

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  
« \_\_\_ » \_\_\_\_\_ 2024 г.

«Разработка модели виртуального тренажера для управления спецтехникой»

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА  
К ВЫПУСКНОЙ КВАЛИФИКАЦИОННОЙ РАБОТЕ  
ЮУрГУ-090401.2024.308/390 ПЗ ВКР

Руководитель работы,  
к.п.н., доцент каф. ЭВМ  
\_\_\_\_\_ Ю.Г. Плаксина  
« \_\_\_ » \_\_\_\_\_ 2024 г.

Автор работы,  
студент группы КЭ-405  
\_\_\_\_\_ Н.Л. Сахаров  
« \_\_\_ » \_\_\_\_\_ 2024 г.

Нормоконтролёр,  
ст. преподаватель каф. ЭВМ  
\_\_\_\_\_ С.В. Сяськов  
« \_\_\_ » \_\_\_\_\_ 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 «Электронные вычислительные машины»

УТВЕРЖДАЮ  
Заведующий кафедрой ЭВМ  
\_\_\_\_\_ Д.В. Топольский  
«\_\_» \_\_\_\_\_ 2024 г.

**ЗАДАНИЕ**  
**на выпускную квалификационную работу бакалавра**  
студенту группы КЭ-405  
Сахарову Никите Леонидовичу  
обучающемуся по направлению  
09.03.01 «Информатика и вычислительная техника»

**Тема работы:** «Разработка модели виртуального тренажера для управления спецтехникой» утверждена приказом по университету от «\_\_» \_\_\_\_ 2024 г.  
№

**Срок сдачи студентом законченной работы:** 01 июня 2024 г.

**Исходные данные к работе:**

Функциональные требования:

1. Реализовать функции выбора миссий.
2. Интерфейс должен быть масштабируемым и адаптивным под разное разрешение.
3. Использование визуальных элементов для указания пользователю направления и целей.

Нефункциональные требования:

1. Поддержка приложения на платформе Windows.

2. Масштабируемость системы, возможность добавления новых функциональных компонентов и миссий без значительных изменений в структуре приложения.

**Перечень подлежащих разработке вопросов:**

1. Аналитический обзор научно-технической, нормативной и методической литературы по тематике работы, анализ аналогичных решений.
2. Выбор и проектирование архитектуры тренажера.
3. Проектирование и разработка интерфейса тренажера.
4. Разработка и тестирование комплекса обучающих сценариев для обучения машиниста спецтехники.

**Дата выдачи задания:** \_\_\_\_ декабрь 2023 г.

Руководитель работы \_\_\_\_\_ / Ю.Г. Плаксина/

Студент \_\_\_\_\_ / Н.Л. Сахаров /

## КАЛЕНДАРНЫЙ ПЛАН

| Этап   | Срок сдачи | Подпись<br>руководителя |
|--|------------|-------------------------|
| Аналитический обзор научно-технической, нормативной и методической литературы              | 01.03.2024 |                         |
| Выбор и проектирование архитектуры тренажера   | 15.03.2024 |                         |
| Разработка интерфейса пользователя   | 04.04.2024 |                         |
| Реализация главного меню и вспомогательных элементов обучения                              | 15.04.2024 |                         |
| Разработка и тестирование комплекса обучающих сценариев для обучения машиниста спецтехники | 01.05.2024 |                         |
| Тестирование обучающих сценариев   | 15.05.2024 |                         |
| Компоновка текста работы и сдача на нормоконтроль  | 27.05.2024 |                         |
| Подготовка презентации и доклада.  | 30.05.2024 |                         |

Руководитель работы \_\_\_\_\_ / Ю.Г. Плаксина /

Студент \_\_\_\_\_ / Н.Л. Сахаров /

## ОГЛАВЛЕНИЕ

|  |    |
|--|----|
| АННОТАЦИЯ.....   | 7  |
| ВВЕДЕНИЕ.....  | 8  |
| 1. АНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ .....                                       | 10 |
| 1.1. Обзор аналогов .....  | 10 |
| Вывод по разделу 1 .....   | 14 |
| 2. ВЫБОР И ПРОЕКТИРОВАНИЕ АРХИТЕКТУРЫ ТРЕНАЖЕРА .....                    | 15 |
| 2.1. Функциональные требования.....                                      | 16 |
| 2.2. Нефункциональные требования .....                                   | 16 |
| 2.4. Инструменты реализации .....  | 16 |
| 2.4.1. Инструменты проектирования интерфейса.....                        | 17 |
| 2.4.2. Инструменты реализации интерфейса .....                           | 18 |
| 2.5. Шаблоны проектирования.....   | 22 |
| 2.5.1. Singleton (Одиночка).....   | 23 |
| 2.5.2. Observer (Наблюдатель).....                                       | 23 |
| 2.6. Выбор архитектурного подхода .....                                  | 24 |
| 2.6.1. MVC (Model-View-Controller).....                                  | 24 |
| 2.6.2. MVVM (Model-View-ViewModel) .....                                 | 26 |
| Вывод по разделу 2 .....   | 27 |
| 3. ПРОЕКТИРОВАНИЕ.....   | 28 |
| 3.1. Диаграмма вариантов использования системы управления миссиями ..... | 29 |
| 3.2. Проектирование интерфейса.....                                      | 30 |
| 3.2.1. Проектирование главного меню .....                                | 30 |
| 3.2.2. Проектирование всплывающей экранной формы паузы .....             | 31 |

|   |    |
|---|----|
| 3.2.3. Проектирование всплывающей экранной формы завершения задачи .. | 32 |
| 3.2.4. Элементы подсказок .....                                       | 33 |
| 3.3. Реализация главного меню .....                                   | 33 |
| 3.3.1. Главный файл UXML.....   | 33 |
| 3.3.2. Стили USS «StylesCss» .....                                    | 34 |
| 3.3.3. Реализация функционала главного меню .....                     | 35 |
| 3.4. Реализация панели состояний в интерфейсе .....                   | 37 |
| 3.4. Реализация всплывающих окон .....                                | 44 |
| 3.4.1. Экранная форма завершения миссии .....                         | 44 |
| 3.4.2. Экранная форма паузы .....                                     | 48 |
| 3.4.3. Реализация элементов задания .....                             | 50 |
| Вывод по разделу 3 .....  | 53 |
| 4. РАЗРАБОТКА И ТЕСТИРОВАНИЕ КОМПЛЕКСА ОБУЧАЮЩИХ<br>СЦЕНАРИЕВ .....   | 54 |
| 4.1. Реализация сценария на движение молотом .....                    | 55 |
| 4.1.1. Тестирование сценария на движение молотом .....                | 56 |
| 4.2. Реализация сценария на передвижение .....                        | 57 |
| 4.2.1. Тестирование сценария на передвижение .....                    | 58 |
| 4.3. Нефункциональное тестирование.....                               | 59 |
| Вывод по разделу 4 .....  | 60 |
| ЗАКЛЮЧЕНИЕ .....  | 61 |
| БИБЛИОГРАФИЧЕСКИЙ СПИСОК .....  | 62 |

## АННОТАЦИЯ

Н.Л. Сахаров «Разработка модели виртуального тренажера для управления спецтехникой» – Челябинск: ЮУрГУ, ВШ ЭКН; 2024, 63 с., 28 ил., библиогр. список – 14 наим.

Выпускная квалификационная работа посвящена разработке виртуального тренажера для управления спецтехникой с использованием кроссплатформенной среды разработки Unity.

В первой главе проведен анализ предметной области, рассмотрены существующие аналоги разрабатываемого симулятора.

Во второй главе обоснованы функциональные и нефункциональные требования к симуляции, а также представлены критерии выбора основного программного обеспечения, в частности, Unity. Осуществлен выбор архитектуры, с учетом масштабируемости и возможности добавления новых функциональных компонентов.

Третья глава посвящена разработке интерфейса тренажера, включая использование визуальных элементов вспомогательных элементов на сцене. Реализовано главное меню тренажера с выбором миссий.

В четвертой главе показаны этапы создания комплекса сценариев для обучающегося специалиста. Рассмотрены и проведены методы тестированию обучающих сценариев.

## **ВВЕДЕНИЕ**

Производство российской дорожно-строительной техники (ДСТ) по итогам 2022 года достигло 75,6 млрд рублей, что на 35% больше, чем в 2021 году. Такие данные в январе 2023 года привели в ассоциации «Росспецмаш» [1], которая объединяет компании, выпускающие порядка 80% ДСТ в РФ.

Отгрузки ДСТ в 2022 году потребителям выросли на 33% относительно показателя 2021 года и составили 73,4 млрд рублей. В количественном выражении отгрузки экскаваторов за 2022 год по сравнению с 2021 годом увеличились в 2,1 раза.

Скорость развития технологий постоянно растет, а требования к квалификации персонала становятся все выше, необходимость в эффективных методах обучения становится все более актуальной.

Теоретические занятия не дают возможности отработать навыки в реальных условиях. А практические упражнения на реальном оборудовании могут быть опасны и дорогостоящими из-за возникновения аварийных ситуаций, травм обучаемых, поломок техники, а также трата моторесурса и горючего. Разработка виртуального тренажера для работы со спецтехникой является актуальной задачей, решение которой позволит повысить эффективность обучения операторов. Реализация проекта позволит создать тренажер, отвечающий современным требованиям и обеспечивающий эффективное обучение операторов.

Целью представленного проекта является разработка виртуального тренажера для работы со спецтехникой, обеспечивающего эффективное обучение операторов с возможностью обучения в виртуальной реальности.

Виртуальная реальность (VR) – это технология, которая позволяет пользователю погрузиться в искусственный мир. VR-технология создает иллюзию присутствия в другом месте, времени или ситуации. Это достигается с помощью специальных сенсорных устройств, которые связывают действия пользователя с аудиовизуальными эффектами. Зрительные, слуховые, осязательные и моторные ощущения пользователя заменяются их имитацией, генерируемой компьютером.

### Преимущества VR:

- повышение безопасности. VR-технология позволяет безопасно обучаться и практиковаться в выполнении опасных или сложных задач.
- повышение эффективности. VR-технология позволяет сократить время обучения и повысить качество работы за счет сокращения ошибок и простоев.

### Недостатки VR:

- может вызывать головокружение.

VR-технология позволяет повысить безопасность, эффективность и производительность в различных областях и обучении.

Для достижения поставленной цели необходимо разработать компоненты виртуального тренажера, позволяющие отрабатывать различные операции и сценарии по управлению спецтехникой. Также необходимо разработать систему управления сценариями и отслеживания прогресса, что позволит корректировать обучение специалиста.

Поскольку над созданием виртуального тренажера мы работаем целой командой, часть моей работы заключается в создании сценариев и анализе различных методов обучения, что позволит создать виртуальный тренажер с единой масштабируемой системой обучения.

# 1. АНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ

## 1.1. Обзор аналогов

Симулятор представляет собой устройство или программный комплекс, разработанный с целью имитации управления определенным процессом, техническим устройством или транспортным средством.

Аналоги разрабатываемого симулятора представлены различными решениями от разработчиков из разных стран, таких как Канада, США, Словения, и России. Эти аналоги могут различаться по набору функций и возможностей. Например, некоторые симуляторы позволяют тренироваться в управлении разнообразной спецтехникой, в то время как другие поддерживают виртуальную реальность, предоставляют обратную связь через сервоприводы и подключение реальных органов управления.

Важным аспектом разработки аналогов симуляторов являются их интерфейсы. Интерфейсы включают в себя элементы управления, что способствует максимальной подлинности обучающего процесса.

Система управления миссиями в симуляторе является ключевым элементом, определяющим эффективность тренировки. Эта система должна предоставлять разнообразные сценарии и задачи, позволяя операторам развивать навыки управления в различных условиях. Кроме того, она может включать функции оценки производительности и обратной связи.

Таким образом, разработка модели виртуального тренажера для управления спецтехникой должна учесть аналоги, их интерфейсы, и систему управления миссиями, обеспечивая комплексное и эффективное обучение будущих операторов.

Рассмотрим несколько симуляторов дорожно-строительной техники. «CYBERMINE» от компании «Thoroughtec simulation» [2], представленный на рисунке 1, у данного продукта присутствует несколько достоинств: созданы различные сценарии, перемещение грунта, создание насыпей, планирование. Реализовано простое изменение ландшафта. Из недостатков можно отметить

отсутствие поддержки очков виртуальной реальности, отсутствие поддержки подвижной платформы.



Рисунок 1 – Учебный симулятор экскаватора «CYBERMINE» от компании «Thoroughtec simulation»

Еще один аналог – «Dozer Simulator Training Pack» [3], данный симулятор представлен на рисунке 2, присутствует несколько достоинств: созданы различные сценарии – перемещение грунта, создание насыпей, планирование, выравнивание площадки, погрузка техники, движение по маршруту. Здесь лучше реализовано перемещение грунта в сравнении с прошлым аналогом.

После выполнения сценариев показывается прогресс и точность оператора. Присутствуют аналогичные недостатки – это отсутствие поддержки очков виртуальной реальности, отсутствие поддержки подвижной платформы на электроприводе. А также могут возникнуть проблемы с покупкой симуляций, потому что разработчик не из России.



Рисунок 2 – «Dozer Simulator Training Pack», разработчик CM Labs Simulations

Отечественный симулятор «Четра» [4], представлен на рисунке 3, имеет ряд преимуществ. В нем реализованы разнообразные сценарии, включая подготовку

машины к работе и перемещение по маршруту, также компания предоставляет управление машиной в различных условиях – на ровной поверхности, под уклоном до 30 градусов, на склоне до 20 градусов, а также при заезде на трал на различных типах грунта, в разные времена года и суток, при различных условиях освещенности и погодных условиях. Тренажер включает в себя перемещение грунта. В процессе обучения фиксируются и анализируются ошибки. Среди недостатков аналогичных продуктов стоит отметить отсутствие поддержки очков виртуальной реальности, наличие в симуляции только одной модели бульдозера марки «Четра». Модель симуляции предназначена только для бульдозеров марки «Четра» и имеет высокий уровень энергопотребления.



Рисунок 3 – Кабина симулятора «Четра»

Некоторые из рассмотренных аналогов имеют специфические ограничения, например, отсутствие поддержки подвижной платформы или фокус на разработке симулятора только для определенного типа бульдозеров. Разработка нового симулятора будет с учетом выявленных недостатков.

Потенциальные заказчики:

- компании, занимающиеся обучением персонала для использования строительной техники и специализированной техники;
- производители спецтехники, желающие предоставить своим клиентам инновационные средства обучения для более эффективного использования и обслуживания их техники;

- образовательные учреждения, предоставляющие программы обучения по эксплуатации и управлению строительной и промышленной техникой;
- компании, специализирующиеся на обучении персонала для обслуживания и управления техникой в сельском хозяйстве и грузоперевозках;
- подрядные строительные компании, стремящиеся улучшить профессиональное обучение своих машинистов с целью повышения безопасности и эффективности строительных работ;
- военные учреждения, где обучение машинистов спецтехники имеет критическое значение для успешного выполнения заданий;
- крупные организации, использующие технику в различных отраслях, такие как горнодобывающая, энергетическая, и другие, и стремящиеся оптимизировать процессы обучения своего персонала.

Из перечисленных потенциальных заказчиков, представленных на рисунке 4, можно отметить:

- крупнейший российский производитель дорожно-строительной и специальной техники «UMG» или «Azarrus»;
- производитель строительного оборудования «JCB»;
- объединение производственных и инжиниринговых предприятий «Интехрос»;
- учебный центр «Лидер»;
- челябинский механический завод;
- учебный центр «Спецтехника»;
- «Стройдормаш».



Рисунок 4 – Потенциальные заказчики

## **Вывод по разделу 1**

В первой главе проведен анализ предметной области, что помогло определить требования к виртуальному тренажёру для управления спецтехникой. Были изучены различные аналоги, представленные на рынке, что позволило выявить их сильные и слабые стороны. Главный недостаток существующих аналогов в том, что нет возможности их приобрести и использовать на отечественном производстве.

## 2. ВЫБОР И ПРОЕКТИРОВАНИЕ АРХИТЕКТУРЫ ТРЕНАЖЕРА

Создание системы сценариев обучения специалистов спецтехники требует тщательного исследования различных методов и видов обучения при помощи симуляции и VR оборудования. Для эффективного обучения нужно продумать множество нюансов, влияющих на восприятие пользователем информации в симуляции, для этого рассмотрим основные требования к системе.

Требования к внешним интерфейсам пользователя:

Система навигации:

- интуитивность: интерфейс навигации должен быть интуитивно понятным для пользователей всех уровней подготовки;
- визуальная навигация: использование маркеров, стрелок и других визуальных элементов для указания пользователю направления и целей.

Система подсказок:

- адаптивность: система подсказок должна быть адаптивной, предоставляя информацию в соответствии с текущим контекстом;
- интерактивность: возможность взаимодействия с подсказками для получения дополнительной информации.

Главное меню:

- интерактивность: возможность легкого перехода между различными разделами, такими как выбор миссии, настройки, обучение и др.;
- визуализация: использование графических элементов для представления различных разделов и функций.

Панель управления:

- эргономика: разработка удобной и интуитивно понятной панели управления миссиями.

Система оценки точности:

- обратная связь: предоставление обратной связи пользователю о его производительности с учетом точности выполнения задач.

## **2.1. Функциональные требования**

Интерфейс системы управления миссиями должен работать под управлением подключенных устройств, таких как клавиатура, мышь. В интерфейсе реализованы функции выбора миссий при помощи библиотеки реактивного программирования. Интерфейс должен быть масштабируемым и адаптивным под разное разрешение экрана.

Функциональность:

- масштабируемость: система должна обеспечивать возможность добавления и модификации миссий без значительных изменений в интерфейсе;
- совместимость: система должна поддерживать различные типы учебных сценариев, таких как на передвижение техники и управление различными инструментами.

## **2.2. Нефункциональные требования**

- система должна использоваться на операционной системе Windows;
- оптимизация ресурсов: минимизация использования ресурсов, таких как центральный процессор (CPU) и видеокарта, для обеспечения плавной работы системы;
- система не должна уменьшать количество кадров в секунду ниже 30 для всей симуляции, интерфейс не должен нагружать систему;
- отзывчивость интерфейса: минимальное время отклика UI на взаимодействие пользователя.

## **2.4. Инструменты реализации**

Тренажеры, предназначенные для подготовки операторов спецтехники, должны не только точно воспроизводить рабочие условия, но и обеспечивать высокий уровень взаимодействия и погружения в процесс. В этом контексте проектирование пользовательского интерфейса (UI) и создание пользовательского опыта (UX) играют ключевую роль. Используются разнообразные инструменты для проектирования и реализации интерфейсов, каждый из которых обладает уникальными функциями и предназначен для определенных задач. Эти инструменты помогают не только в визуализации интерфейса, но и в его

интеграции с механиками тренажера, что важно для достижения максимальной эффективности обучения. Рассмотрим детально, какие инструменты и методы используются при создании интерфейсов в таких системах.

### **2.4.1. Инструменты проектирования интерфейса**

Существует множество инструментов для проектирования интерфейсов, которые популярны среди дизайнеров и разработчиков.

1. Figma – онлайн-инструмент для проектирования интерфейсов, который удобен для создания пользовательских интерфейсов (UI) и пользовательского опыта (UX). Он предоставляет инструменты для создания элементов интерфейса, прототипирования, коллаборации и экспорта.
2. Adobe XD – это комплексный инструмент от Adobe, предназначенный для дизайна UI/UX, который позволяет создавать дизайны, прототипы и даже делиться ими для получения обратной связи. Adobe XD поддерживает векторное редактирование, анимацию и создание интерактивных прототипов.
3. Sketch – это векторный редактор интерфейсов, который изначально разработан для macOS. Sketch широко используется для создания дизайнов интерфейсов благодаря его простоте и мощным функциям для работы с символами и стилями, что делает его идеальным для работы над комплексными проектами.
4. Axure RP – это инструмент для более сложного прототипирования, который позволяет дизайнерам создавать богатые интерактивные прототипы без написания кода. Axure может использоваться для тестирования сложных логических структур и взаимодействий.

#### **Вывод по инструментам проектирования:**

Для создания интерфейса виртуального тренажера был выбран инструмент Figma [5], который облегчит разработку управляющих панелей и элементов управления для симуляции спецтехники в Unity, так как он предоставляет возможности анимации, визуализации, доступ к шаблонам и облачному хранению проектов, не требует установки, т.к. доступна web-версия.

## 2.4.2. Инструменты реализации интерфейса

В разработке интерактивных приложений, таких как виртуальные тренажеры и симуляторы, выбор правильного инструментария играет ключевую роль. Эти инструменты не только облегчают процесс разработки, но и определяют качество взаимодействия пользователя с конечным продуктом. Движки, такие как Unity, представляют собой комплексные решения, которые предоставляют разработчикам наборы мощных инструментов для создания многоплатформенных приложений. Они позволяют реализовывать сложные интерактивные сценарии, что является неотъемлемой частью обучения и тренировок с использованием тренажеров. Основные инструменты, используемые для реализации интерфейсов в таких системах, и обсудим их ключевые характеристики и преимущества, на примере популярного движка Unity и его альтернатив.

1. Unreal Engine [6] – это один из самых мощных игровых движков на рынке, известный своей высококачественной графикой и реалистичными визуальными эффектами. Unreal Engine подходит для создания сложных и детализированных 3D-сцен, а также обладает мощными инструментами для реализации интерфейсов, включая поддержку пользовательских элементов UI и создание интерактивных элементов.
2. Godot [7] – это открытый игровой движок, который поддерживает как 2D, так и 3D форматы. Godot выделяется своей модульной структурой и легкостью в освоении. В Godot также имеются инструменты для реализации интерфейсов, включая создание пользовательских элементов UI и управление пользовательским взаимодействием.
3. CryEngine [8] – известен своей передовой графикой и мощными инструментами для создания реалистичных окружений и текстур. CryEngine часто используется для создания игр и проектов, где важно детальное визуальное воспроизведение. В CryEngine также предоставляются инструменты для реализации интерфейсов с возможностью создания пользовательских элементов и управления взаимодействием пользователя.

4. Unity [9] – это кроссплатформенный движок, который используется для создания интерактивных 2D и 3D приложений, включая игры, симуляторы и виртуальную реальность. Основные преимущества Unity по сравнению с другими игровыми движками:

- 1) Кроссплатформенность: Unity поддерживает разработку приложений для различных платформ, таких как iOS, Android, Windows, Mac, Linux, WebGL и многих других.
- 2) Широкие возможности графики: Unity обладает мощным движком рендеринга, поддерживает шейдеры и обеспечивает высококачественную графику, что особенно важно для создания реалистичных симуляторов.
- 3) Обширное сообщество: Unity имеет огромное сообщество разработчиков, что облегчает обмен опытом, получение поддержки и доступ к множеству ресурсов и плагинов.
- 4) Простота использования: Unity предоставляет интуитивно понятный интерфейс, благодаря чему можно быстро освоить его и начать разрабатывать приложения.

Для реализации системы управления сюжетными миссиями, как и для всего виртуального тренажера, будем использовать кроссплатформенный движок Unity, такой выбор сделан в компании DST Урал.

Для создания симулятора спецтехники Unity предоставляет инструменты для разработки физики, взаимодействия и анимации. Он поддерживает создание реалистичных 3D моделей и обеспечивает удобное взаимодействие с внешним оборудованием, таким как джойстики, сенсоры и другие устройства ввода. С использованием Unity можно также легко интегрировать виртуальную реальность для более глубокого и интенсивного опыта взаимодействия с симулятором спецтехники.

В состав Unity входит редактор для создания сцен и объектов, и множество других инструментов. При обучении персонала очень важно уделить внимание созданию интерфейсов, которые обеспечивают пользовательский опыт и

визуальное восприятие. Чтобы достичь этой цели, Unity предоставляет различные инструменты и компоненты, которые упрощают процесс разработки интерфейсов.

Более подробно рассмотрим эти компоненты Unity, специально предназначенные для создания пользовательских интерфейсов.

#### **2.4.2.1 UI Toolkit**

UI Toolkit в Unity – это мощный инструмент для создания сложных пользовательских интерфейсов. Его главное преимущество заключается в гибкости и масштабируемости. В отличие от более простых решений, таких как Canvas, UI Toolkit предоставляет более высокий уровень контроля над стилизацией элементов. Система визуальных элементов позволяет создавать сложные макеты с учетом требований дизайна.

Одним из ключевых преимуществ UI Toolkit является его система событий. Она обеспечивает удобный механизм обработки пользовательских взаимодействий, таких как нажатия кнопок, перетаскивание и многое другое. Это существенно упрощает реализацию интерактивности в пользовательском интерфейсе.

Также следует отметить, что UI Toolkit предоставляет удобное управление ресурсами. Вы можете легко настраивать шрифты, изображения и стили.

HTML и CSS в UI Toolkit: UI Toolkit в Unity предоставляет функциональность, схожую с веб-разработкой, благодаря использованию USS (UI Style Sheets), которые вдохновлены CSS. Это позволяет разработчикам использовать знакомые подходы к стилизации:

- 1) USS (UI Style Sheets): USS в UI Toolkit обеспечивает синтаксис, схожий с CSS. Разработчики могут определять стили для элементов интерфейса, применять их селекторы, и создавать каскадные таблицы стилей.
- 2) Flexbox-подобный макет: система в UI Toolkit поддерживает гибкий макет, аналогичный Flexbox в CSS. Это упрощает распределение и выравнивание элементов интерфейса.

3) Темы: UI Toolkit поддерживает темы, что позволяет легко изменять визуальный стиль всего приложения аналогично изменению тем в веб-дизайне.

#### **2.4.2.2 Canvas**

Canvas, еще один инструмент для создания интерфейса в Unity, с другой стороны, предоставляет более простой и быстрый способ создания интерфейса. Этот инструмент, ориентированный в первую очередь на 2D пространство, предоставляет легкий способ размещения элементов относительно экрана.

Он также отлично интегрируется с 3D сценой. Это особенно полезно при создании симулятора спецтехники, где информацию можно отображать прямо на экране игрового мира.

Однако Canvas может оказаться ограниченным, если требуется более сложная структура интерфейса или если необходима более тщательная стилизация элементов.

#### **2.4.2.3 Zenject**

Zenject [10] – это фреймворк для управления зависимостями (dependency injection framework) в приложениях на языке программирования C#. Разработанный под платформу Unity, Zenject позволяет эффективно управлять зависимостями между компонентами игрового движка Unity, обеспечивая легкость внедрения зависимостей и повышая гибкость кода.

#### **2.4.2.4 Unity Reactive Extensions**

UniRx [11] (или Unity Reactive Extensions) – это библиотека реактивного программирования для языка C# и платформы Unity. Реактивное программирование основано на концепции потоков данных и событий, позволяя разработчикам управлять асинхронными операциями и реагировать на изменения данных с помощью функционального стиля программирования.

#### **Вывод по выбору инструментов:**

Для создания главного меню, где важны сложная структура, высокий уровень стилизации и обработка событий, будет использован UI Toolkit. Это обеспечит больший контроль над внешним видом и взаимодействием элементов.

Для элементов интерфейса на сцене, где ключевыми являются простота использования и интеграция с 3D сценой, Canvas будет предпочтительным инструментом. Он позволяет легко внедрять элементы интерфейса в игровой мир, обеспечивая при этом простоту использования.

Таким образом, комбинирование UI Toolkit для главного меню и Canvas для элементов на сцене предоставляет оптимальный баланс между гибкостью и простотой в разработке пользовательского интерфейса в Unity.

Создание системы миссий для симулятора может включать использование различных инструментов, принципов проектирования и шаблонов для эффективной и управляемой структуры.

## **2.5. Шаблоны проектирования**

Шаблоны проектирования представляют собой повторяющиеся архитектурные конструкции в области разработки программ. Они предлагают стандартные решения для типичных проблем, возникающих в ходе проектирования программных систем.

Использование шаблонов проектирования обладает несколькими преимуществами:

1. Снижение сложности разработки: за счет предварительно разработанных абстракций, шаблоны позволяют решать целый класс проблем, упрощая разработку и снижая риск возникновения ошибок.
2. Облегчение коммуникации: шаблоны являются общепризнанными конструкциями, что облегчает коммуникацию между разработчиками. Они предоставляют общий язык и позволяют ссылаться на известные решения, упрощая понимание кода и обмен опытом.
3. Унификация деталей решений: применение шаблонов способствует унификации деталей решений, таких как модулей и элементов проекта. Это помогает сократить количество ошибок, обеспечивая единообразие в структуре и архитектуре приложения.

Использование шаблонов проектирования способствует повышению эффективности и качества разработки. Рассмотрим необходимые в работе шаблоны.

### 2.5.1. Singleton (Одиночка)

Шаблон «Одиночка» [12], диаграмма которого представлена на рисунке 5, направлен на обеспечение того, чтобы у класса существовал единственный экземпляр, к которому можно было получить доступ глобально. Это решает проблему с общими ресурсами, которые должны быть доступны из разных частей приложения. В Unity, одиночки часто используются для создания глобальных менеджеров, например, менеджера ресурсов или контроллера игры. Это обеспечивает удобный способ централизованного управления приложением.

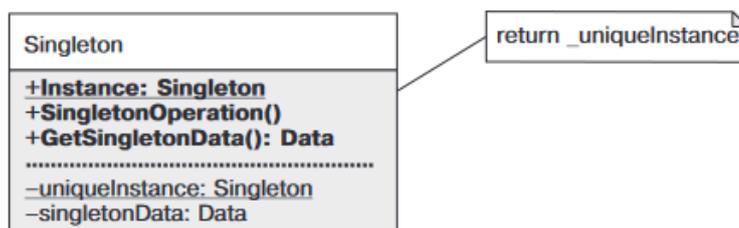


Рисунок 5 – Диаграмма классической реализации паттерна «Синглтон»

### 2.5.2. Observer (Наблюдатель)

Шаблон «Наблюдатель» [12], диаграмма реализации представлена на рисунке 6, позволяет объектам подписываться и получать уведомления об изменениях в других объектах. Это снижает степень зависимости между объектами, таким образом, изменения в одном объекте могут легко уведомлять все заинтересованные стороны. В Unity, шаблон «Наблюдатель» может быть реализован для создания гибких систем событий. Например, в случае изменения здоровья игрового персонажа, все объекты, подписанные на это событие, получат уведомление. Этот шаблон нам поможет реализовать Unity Reactive Extensions.

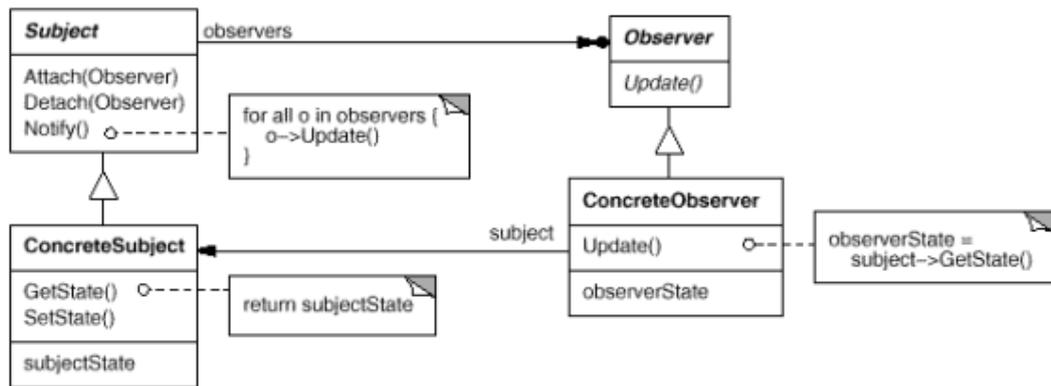


Рисунок 6 – Диаграмма классической реализации паттерна «Наблюдатель»

## 2.6. Выбор архитектурного подхода

Архитектурные подходы в разработке приложений необходимы по ряду следующих причин. Вот несколько ключевых причин:

1. Необходимость в структурировании приложения: архитектурные подходы позволяют разбить сложное приложение на более мелкие компоненты, что делает его структуру более понятной и управляемой.
2. Потребность в разделении обязанностей: они помогают четко определить, какие части приложения отвечают за какие функции, что упрощает понимание и поддержку кода.
3. Необходимость в масштабируемости: правильно спроектированная архитектура обеспечивает возможность масштабирования приложения, что позволяет эффективно добавлять новые функции или изменять существующие без значительных затрат на изменение всего приложения.
4. Потребность в поддержке и модификации: хорошо спроектированная архитектура облегчает внесение изменений и поддержку приложения в будущем. Это делает его более гибким и адаптируемым к изменяющимся требованиям.

### 2.6.1. MVC (Model-View-Controller)

Благодаря применению архитектурного шаблона MVC, который представлен на рисунке 7, можно разделить приложения на три компонента: модель, представление и контроллер.

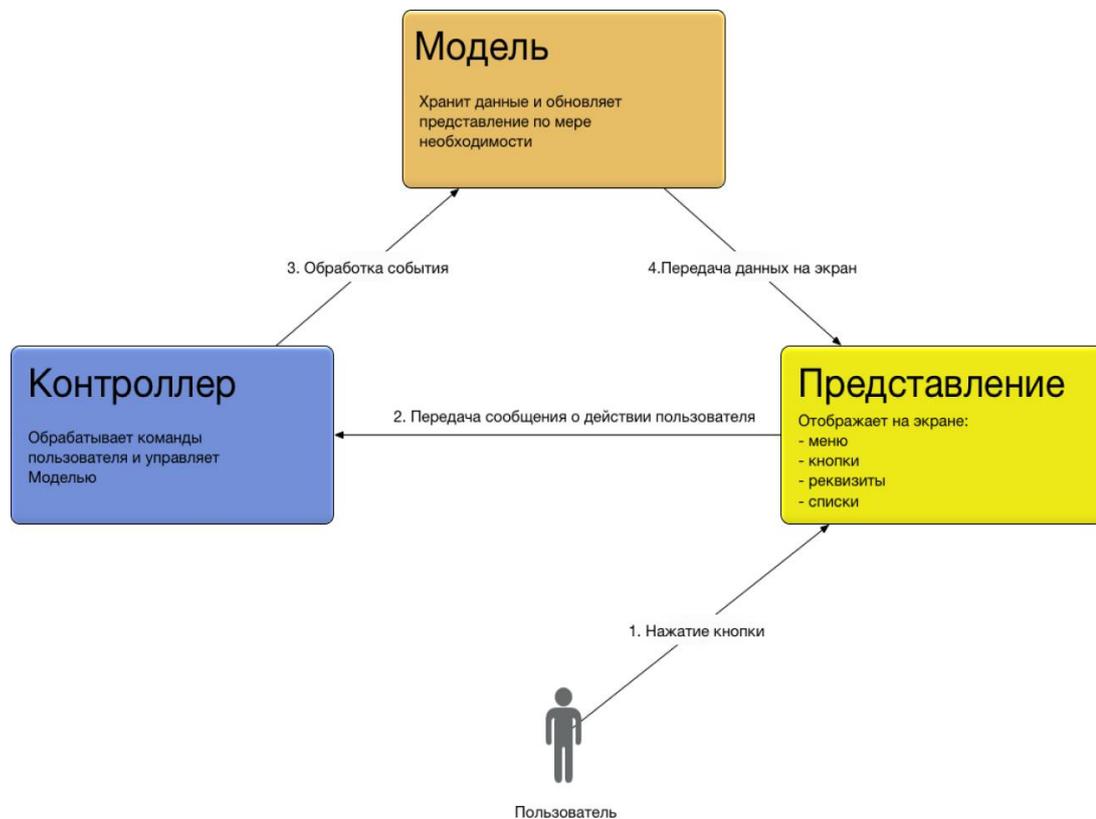


Рисунок 7 – Архитектурный шаблон MVC

- модель (Model): описывает данные приложения и связанную с ними логику, такую как валидация. Может быть представлена моделями представлений, хранящими данные и логику управления ими;
- представление (View): отображает визуальную часть или интерфейс приложения для пользователя. Может содержать логику отображения данных, но не должно обрабатывать запросы или управлять данными;
- контроллер (Controller): центральный компонент MVC, обеспечивающий связь между пользователем, представлением и данными. Обрабатывает запросы пользователя, получает данные и возвращает результат обработки, например, представление с данными модели.

Это разделение упрощает процесс разработки и тестирования, делает код более модульным и понятным. Благодаря четкому разделению обязанностей каждого компонента, процесс изменения и поддержки приложения в будущем становится более удобным и эффективным.

## 2.6.2. MVVM (Model-View-ViewModel)

Шаблон проектирования MVVM, структурная схема представлена на рисунке 8, используется для разделения приложения на три компонента.

- модель (Model): этот компонент приложения описывает данные и содержит логику, прямо связанную с этими данными, например, проверку их корректности. Модель не имеет никакого отношения к отображению данных или взаимодействию с пользовательским интерфейсом;
- представление (View): визуальный интерфейс, через который пользователь взаимодействует с приложением. Обычно представление определяется в виде XAML-кода, который содержит различные элементы управления, такие как кнопки и текстовые поля;
- модель представления (ViewModel): этот компонент связывает модель и представление, обеспечивая передачу данных между ними и управление логикой. ViewModel также обычно содержит логику для получения данных из модели и их обновления, а также отвечает за выполнение действий, инициируемых пользователем через представление.

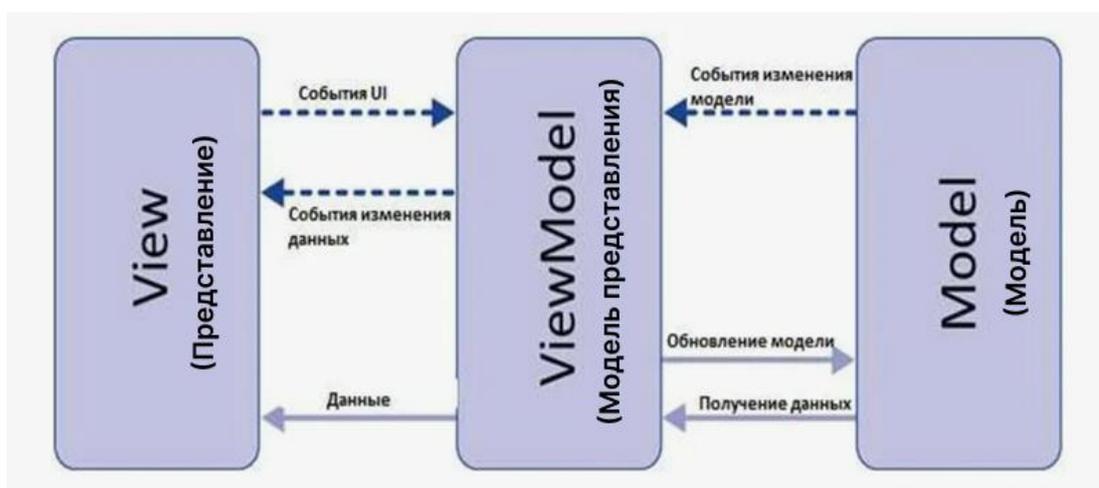


Рисунок 8 – Структурная схема MVVM

### Вывод по данным архитектурам:

Выбор шаблона проектирования MVC для сценариев симуляции спецтехники обоснован особенностями виртуального тренажера. MVC обеспечивает четкое разделение обязанностей между моделью, представлением и контроллером, что упрощает поддержку и модификацию кода. Контроллер

обеспечивает гибкое управление интерфейсом пользователя, способствует более быстрой разработке благодаря параллельной работе над различными аспектами системы.

### **Вывод по разделу 2**

Во второй главе дипломной работы были обоснованы функциональные и нефункциональные требования к разрабатываемому виртуальному тренажеру. Архитектура системы была тщательно продумана для обеспечения гибкости и возможности легкой интеграции новых функциональных компонентов в будущем.

Были рассмотрены вопросы выбора основных инструментов для реализации тренажера, таких как движки Unreal Engine, Godot, и CryEngine, Unity, каждый из которых предлагает свои уникальные возможности для реализации требуемых функций интерфейса и взаимодействия пользователя с системой.

### 3. ПРОЕКТИРОВАНИЕ

Общая модель виртуального тренажера спецтехники, которая реализована в рамках создания тренажера машинистов бульдозера D12 от компании DST. Общая схема отображена на рисунке 12.

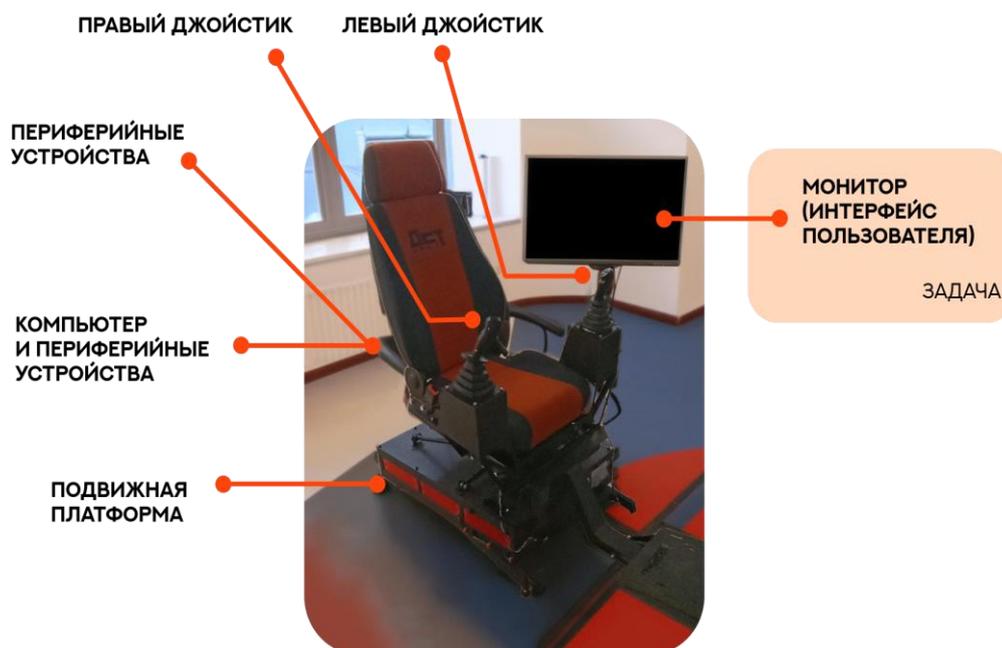


Рисунок 12 – Общая модель виртуального тренажера DST

Реализуем интерфейс симулятора (компонент представления) для тренажера, который будет отображаться на мониторе специалиста.

Модель тренажера для других видов техники может отличаться, поскольку могут отличаться элементы управления.

Для реализации всех компонентов рассмотрим схему архитектуры MVC для тренажера спецтехники, предоставленную на рисунке 13.

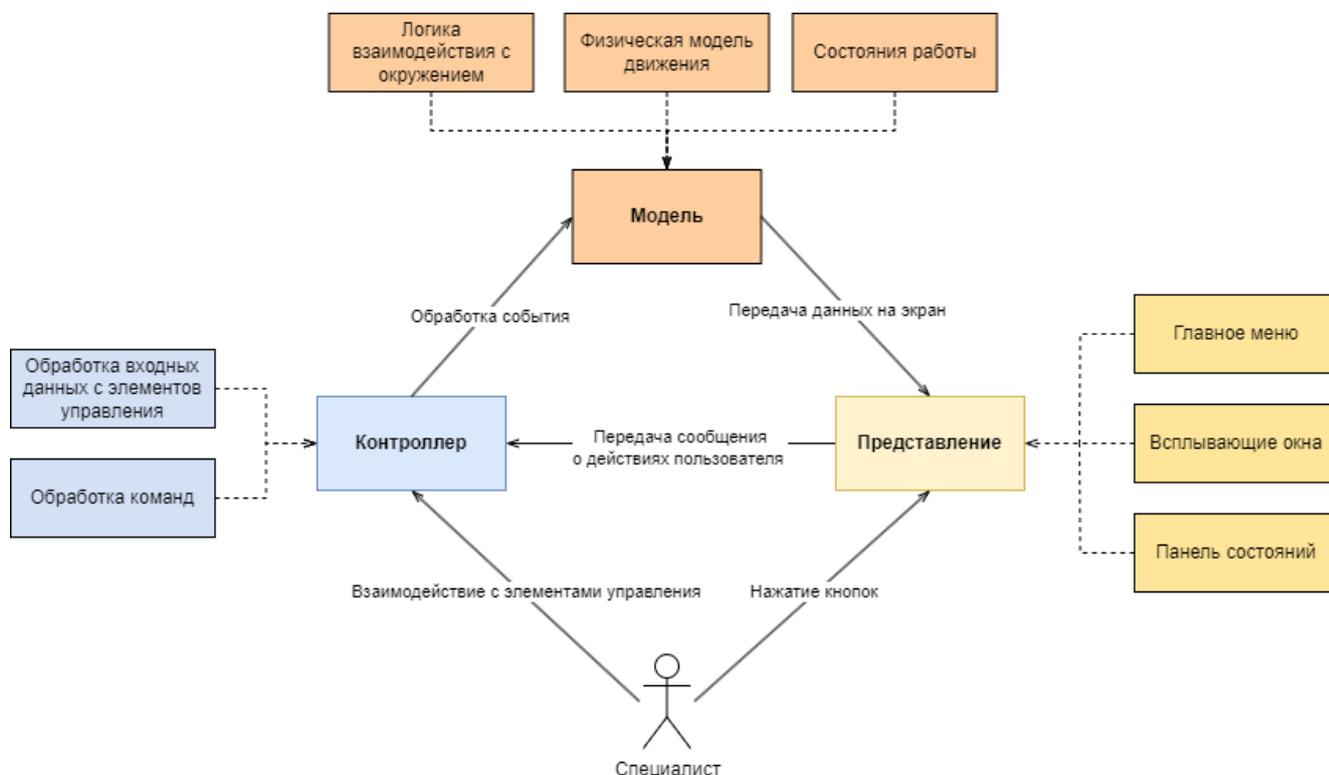


Рисунок 13 – Схема архитектуры MVC для виртуального тренажера

В данной работе будет реализован компонент «Представление» и часть компонента «Модель».

### 3.1. Диаграмма вариантов использования системы управления миссиями

Диаграмма вариантов использования для интерфейса машиниста спецтехники является инструментом анализа и проектирования пользовательского опыта. Опишем различные сценарии использования интерфейса, выявим основные возможности взаимодействия пользователя с системой. В рамках языка графического описания для объектного моделирования UML была построена модель взаимодействия актера «Пользователь» с программной системой.

Рассмотрим диаграмму вариантов использования, представленную на рисунке 14.

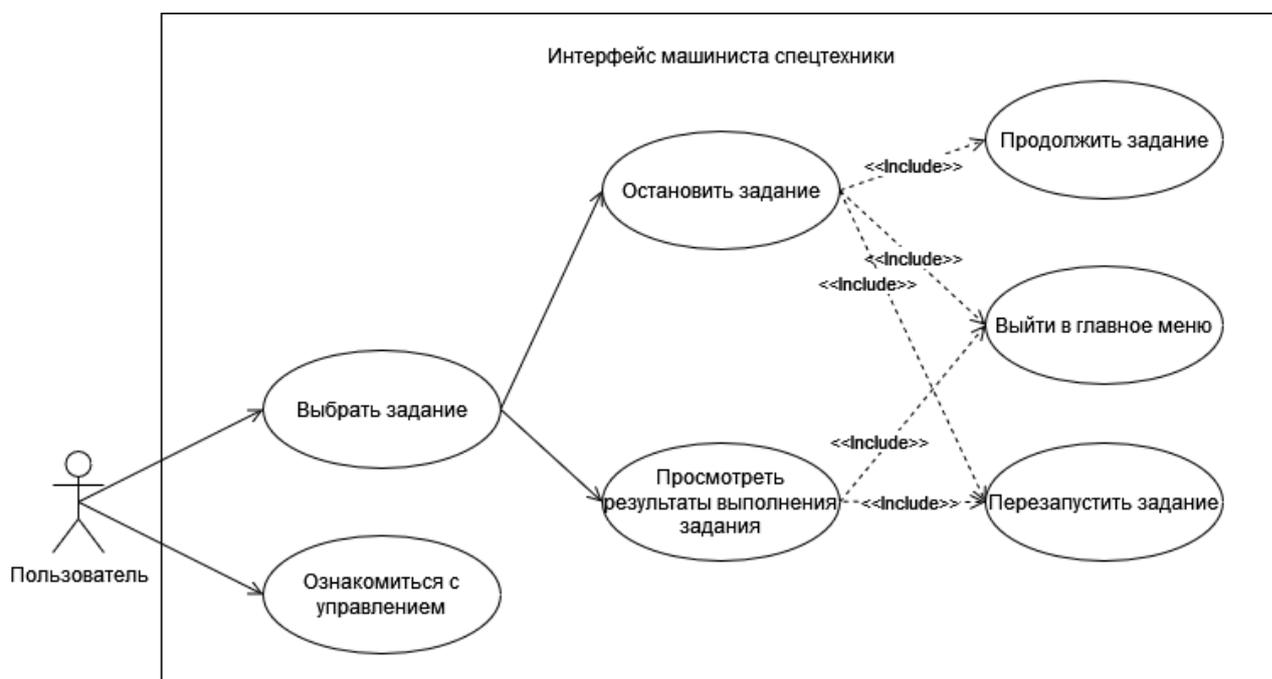


Рисунок 14 – Диаграмма вариантов использования интерфейса

Специалист начинает свое обучение с главного меню, представленное при помощи UI Toolkit [13], где у него есть возможность выбрать необходимое задание. Если пользователь не знаком с управлением, то есть возможность открыть отдельную экранную форму с перечисленными управляющими элементами. После того как пользователь выбирает миссию происходит загрузка сцены через специальный класс «UIController». На каждой сцене есть всплывающее меню, в котором отображаются элементы для взаимодействия, позволяющие в зависимости от статуса миссии продолжить выполнение, перезапустить задание, выйти в главное меню.

## 3.2. Проектирование интерфейса

При помощи инструмента Figma спроектируем визуальную часть интерфейса в соответствии с требованиями.

### 3.2.1. Проектирование главного меню

Меню, представленное на рисунке 15, должно включать:

- 1) Название компании.
- 2) Вид спецтехники.
- 3) Список сцен.

- 4) Возможность вызова всплывающего окна с инструкцией по управлению, при помощи кнопки «Управление».
- 5) Кнопка запуска сценария «Запустить сценарий».
- 6) Возможность покинуть симуляцию при помощи кнопки «Выйти».

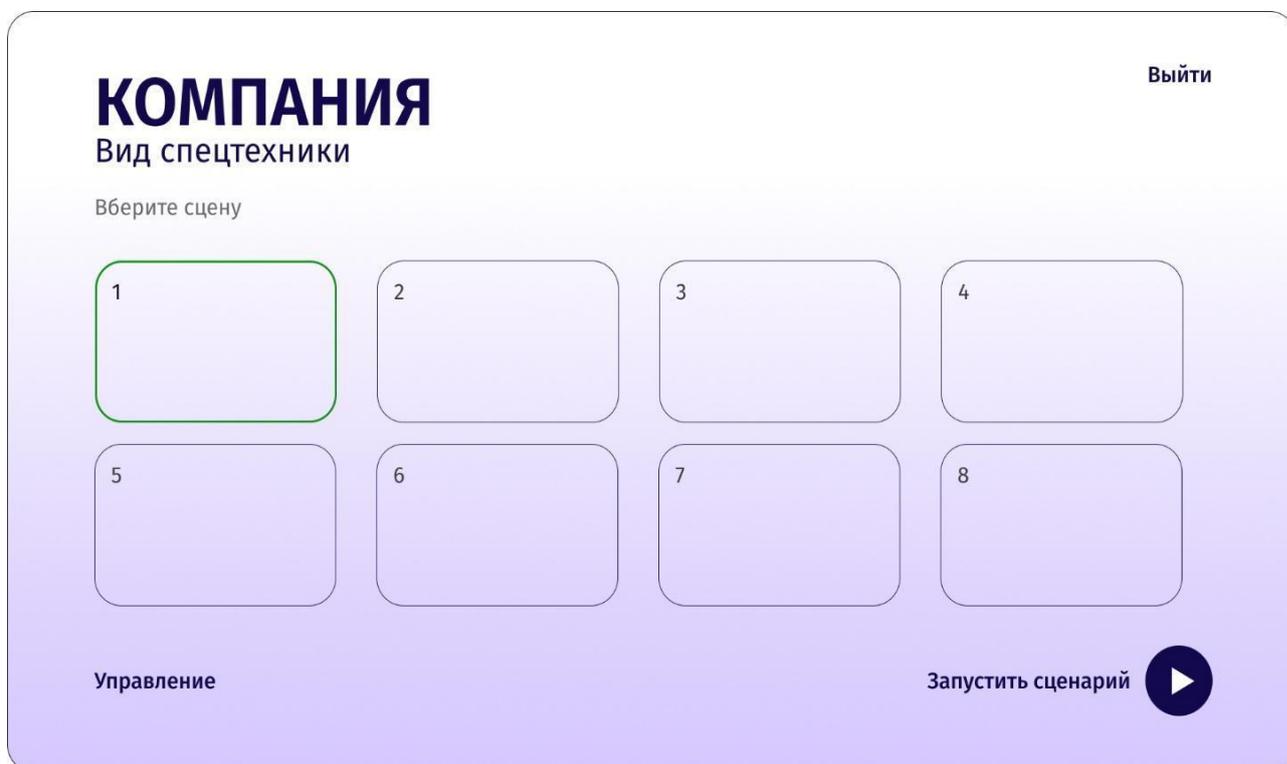


Рисунок 15 – Главное меню

### 3.2.2. Проектирование всплывающей экранной формы паузы

При создании всплывающего окна паузы, представленного на рисунке 16, в интерфейсе пользователя необходимо обязательно продумать все возможные варианты взаимодействия:

- 1) Возможность выйти в главное меню.
- 2) Продолжить задание.
- 3) Начать задание с начала.
- 4) Закрывать всплывающую экранную форму.

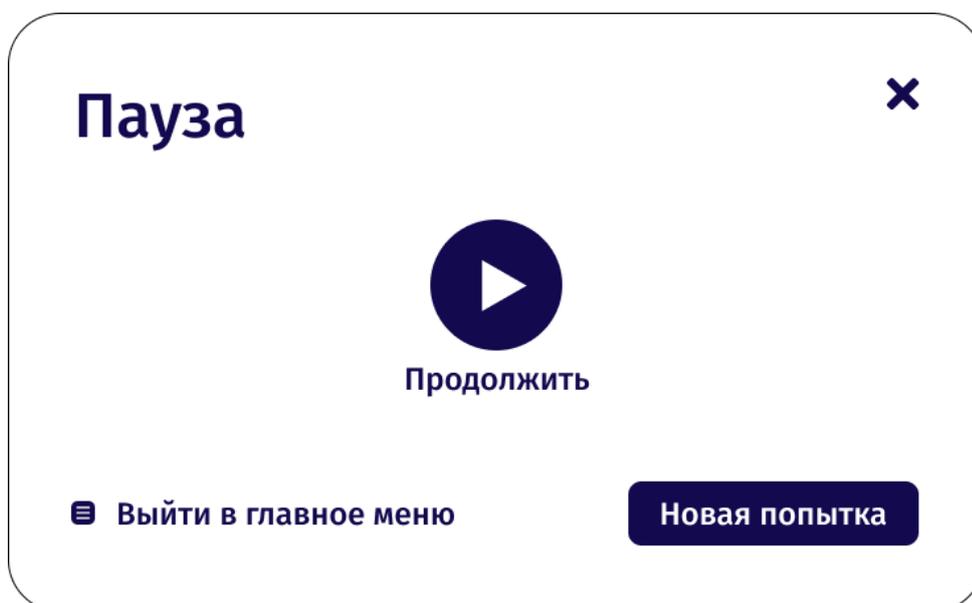


Рисунок 16 – Экранная форма паузы

### 3.2.3. Проектирование всплывающей экранной формы завершения задачи

Экранная форма завершения задачи, которое представлено на рисунке 17, должно включать:

- 1) Статус задачи – «Задача выполнена».
- 2) Время выполнения задачи.
- 3) Возможность выйти в главное меню.
- 4) Начать задание с начала.

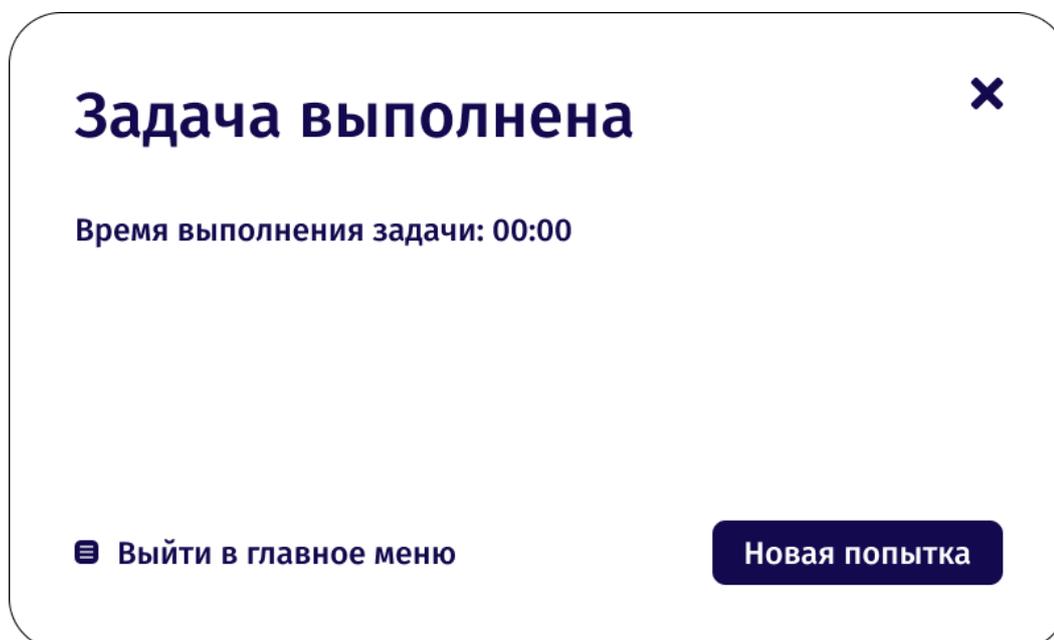


Рисунок 17 – Экранная форма завершения задачи

### 3.2.4. Элементы подсказок

Во время выполнения задания машинист спецтехники должен понимать в каком режиме работает техника, для это нужно разместить на экране, при помощи Canvas, элементы, информирующие пользователя.

Интерфейс должен содержать:

- 1) Короткое описание здания.
- 2) Состояние работы техники.
- 3) Текущая задача.
- 4) Этап выполнения задачи (в процентах).

Графическое отображение информационных элементов представлено на рисунке 18. Расположение элементов не должно перекрывать обзор для обучающегося.

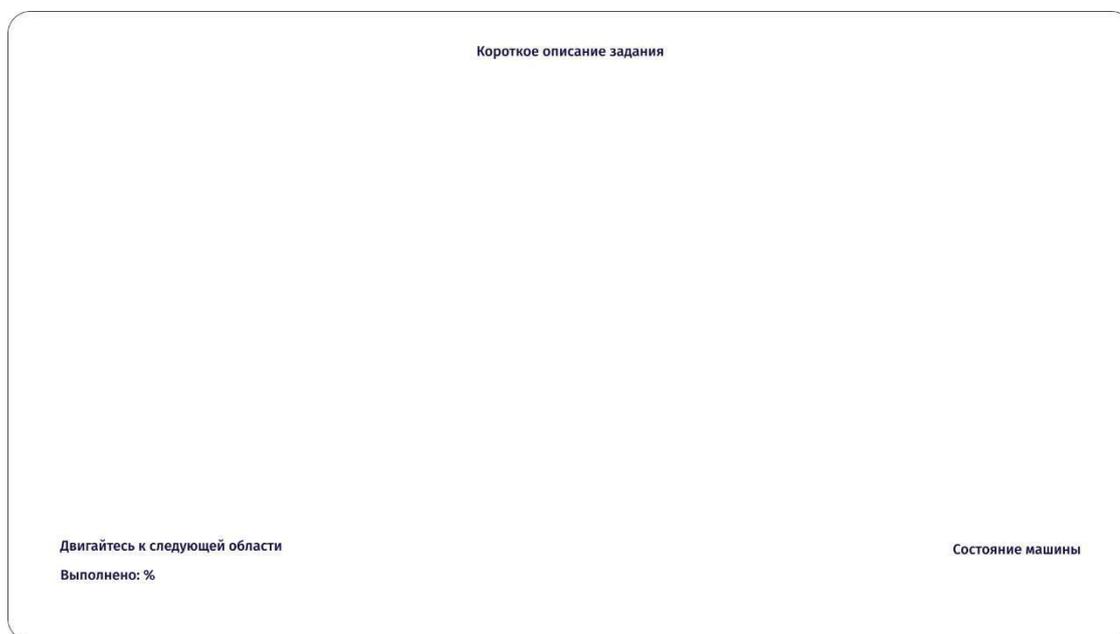


Рисунок 18 – Расположение панели состояний в интерфейсе

### 3.3. Реализация главного меню

Главное меню реализовано при помощи UI Toolkit в Unity.

#### 3.3.1. Главный файл UXML

Этот файл содержит разметку пользовательского интерфейса в формате XML. В нем определяются все элементы интерфейса, их расположение и взаимосвязь. Данный файл был реализован в среде разработки Unity при помощи

встроенного редактора, отображение интерфейса главного меню можно посмотреть на рисунке 19.

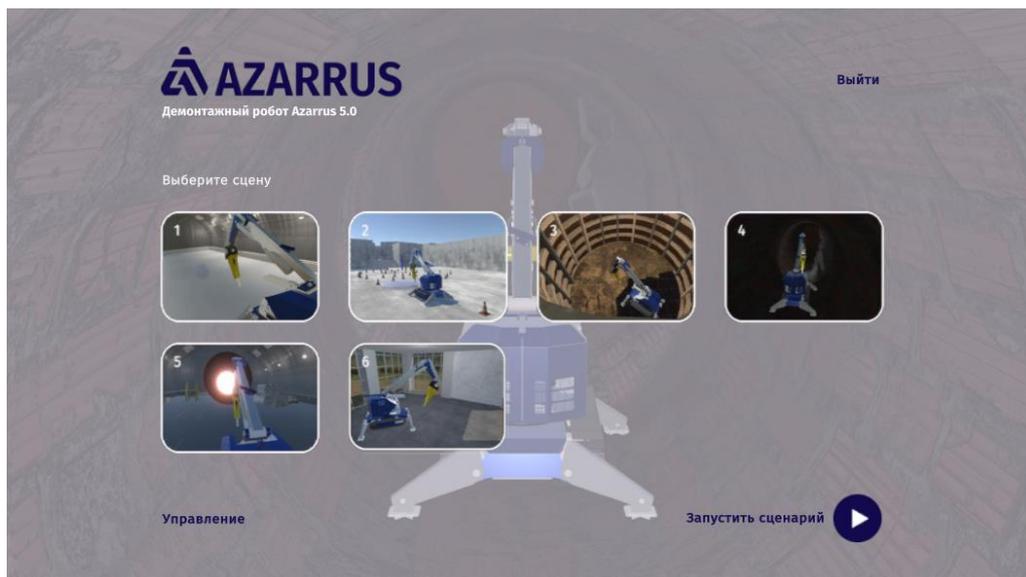


Рисунок 19 – Интерфейс главного меню

### 3.3.2. Стили USS «StylesCss»

Файлы стилей (USS) используются для описания внешнего вида элементов интерфейса. В файле «StylesCss» указываем все стили элементов меню, в листинг 1 представлена реализация базового стиля отображения кнопки «Play» и изменение стиля при наведении указателя мыши.

Листинг 1 – Фрагмент реализации параметров стиля для кнопки «Play» из главного меню

```
.buttonPlay {
    background-color: rgba(255, 255, 255, 0);
    background-image: resource('UIelements/Play');
    height: 100px;
    width: 100px;
    border-top-width: 0;
    border-right-width: 0;
    border-bottom-width: 0;
    border-left-width: 0;
    -unity-background-image-tint-color: rgb(255, 255, 255);
}
.buttonPlay:hover {
    -unity-background-image-tint-color: rgb(14, 148, 19);
}
```

### 3.3.3. Реализация функционала главного меню

Для обработки пользовательского взаимодействия и управления элементами меню потребуется скрипт «UIController». Этот скрипт содержит методы для обработки событий, нажатия кнопок, и для выполнения соответствующих действий, таких как загрузка уровней или отображение окон.

Рассмотрим диаграмму последовательности для «UIController», которая представлена на рисунке 20. Диаграмма последовательности показывает взаимодействие между пользователем, классами UIController, UIDocument и MissionManager, а также элементами пользовательского интерфейса (кнопки).

Описание всех связей:

- Start(): метод Start() класса UIController вызывается при запуске приложения, иницилируя последовательность действий;
- InitializeUIElements(): UIController инициализирует элементы пользовательского интерфейса, такие как корневой элемент и контейнер для инструкции;
- RegisterButtonClickEvents(): регистрирует обработчики событий для кнопок в пользовательском интерфейсе;
- LoadSceneButtons(): создает кнопки для выбора игровых сцен и регистрирует обработчики событий для каждой из них;
- NameSceneForButton(): метод определяет имя сцены для каждой кнопки с учетом ее номера;
- ClickScene(): вызывается при нажатии кнопки выбора сцены. Загружает выбранную сцену через MissionManager;
- LoadScene(): метод класса MissionManager, который загружает выбранную сцену.

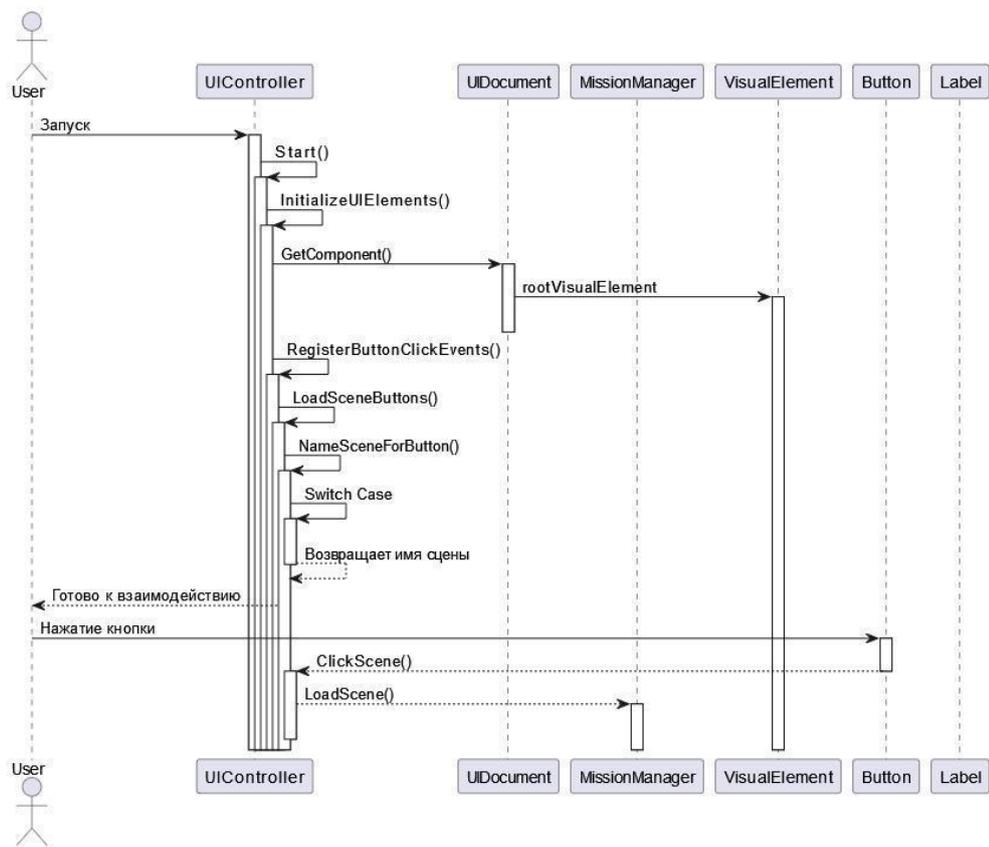


Рисунок 20 – Диаграмма последовательности «UIController»

Фрагмент реализации UIController представлен в листинге 2, описан основной метод «Start», где регистрируем для каждой кнопки обработчик событий. Каждый обработчик связан с определенным действием, которое должно произойти при нажатии на соответствующую кнопку.

Листинг 2 – Фрагмент реализации UIController

```

void Start()
{
    // Получаем корневой элемент пользовательского интерфейса.
    root = GetComponent<UIDocument>().rootVisualElement;

    // Регистрируем обработчики событий для кнопок.
    _instruction = root.Q<Label>("Instruction");
    _instruction.RegisterCallback<ClickEvent>(ClickInstruction);

    _instruction = root.Q<Label>("Exit");
    _instruction.RegisterCallback<ClickEvent>(ClickExit);

    _back = root.Q<Label>("Back");
    _back.RegisterCallback<ClickEvent>(ClickBack);
    _Instruction_container
    root.Q<VisualElement>("Instruction_container");
}

```

## Продолжение листинга 2

```
_Instruction_container.RegisterCallback<ClickEvent>(ClickBack);

    _Instruction_container.style.display = DisplayStyle.None;

    _scenePlay = root.Q<Button>("Play");
    _scenePlay.RegisterCallback<ClickEvent>(ClickPlay);
    // Создаем кнопки для выбора сцены и регистрируем для них
    // обработчики событий.
    for (int i = 1; i <= 8; i++)
    {
        string buttonName = NameSceneForButton(i);
        var sceneButton = root.Q<Button>(buttonName);

        if (sceneButton != null)
        {
            _sceneButtons.Add(sceneButton);
            sceneButton.RegisterCallback<ClickEvent>(ev =>
ClickScene(sceneButton));
        }
        else
        {
            Debug.LogError($"Button {buttonName} not found!");
        }
    }
}
```

Рассмотрим реализацию загрузки сцен при нажатии на кнопку «Play», которая отображена на листинге 3. Для реализации системы загрузки мы обращаемся к отдельному компоненту обработки «MissionManager», который выполняет роль контроллера между графическим представлением и моделью.

Листинг 3 – Реализация функции «ClickPlay» для старта задания

```
void ClickPlay(ClickEvent e)
{
    // Загружаем выбранную сцену с помощью MissionManager.
    _missionManager.LoadScene(_currentSceneSelect);
}
```

### 3.4. Реализация панели состояний в интерфейсе

Для реализации вспомогательных элементов в интерфейсе была создана система управления состояниями и интерфейсом пользователя для спецтехники, она представлена на рисунке 21. Для этого были использованы Zenject для внедрения зависимостей и UniRx для реактивного программирования.

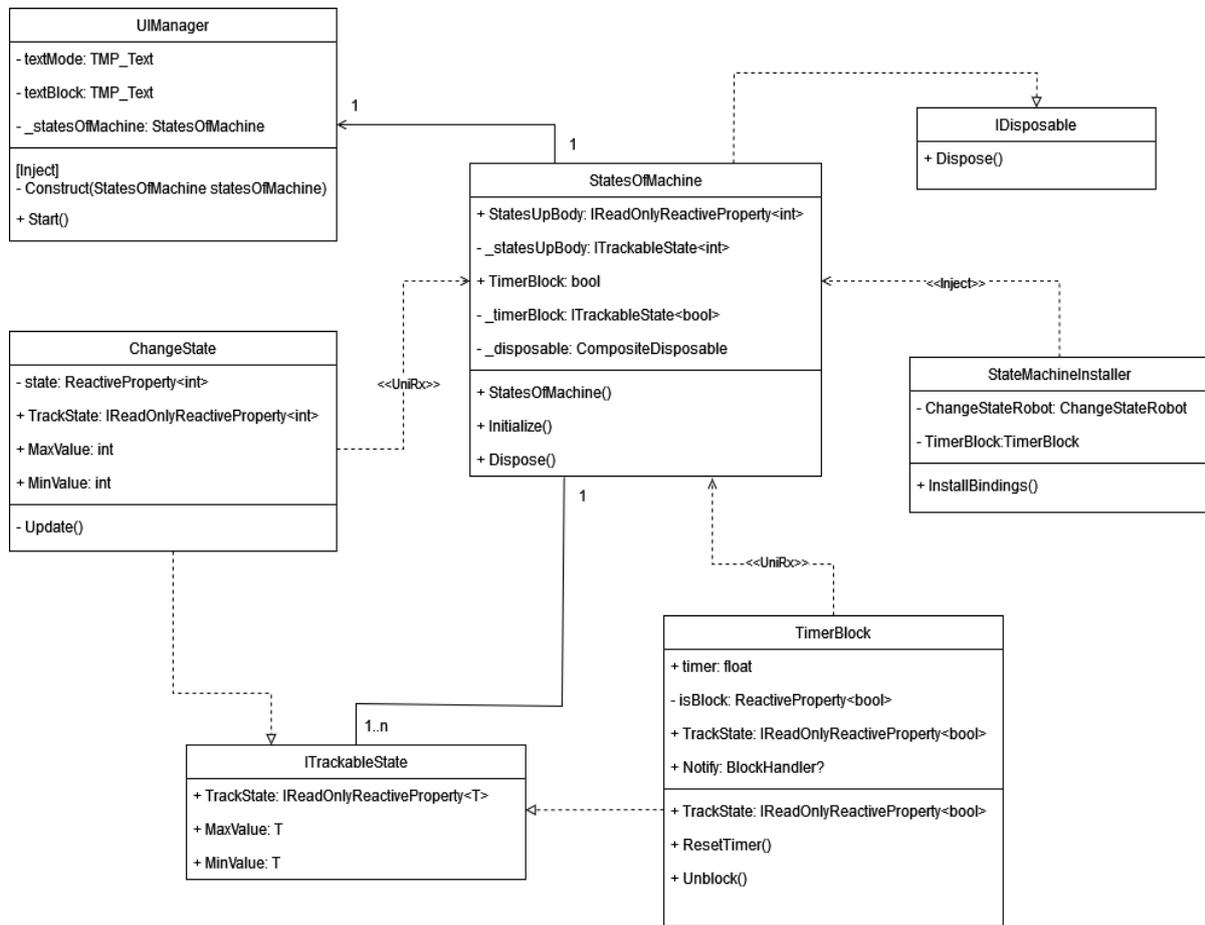


Рисунок 21 – UML диаграмма классов для работы панели состояний.

Диаграмма включает в себя следующие компоненты:

### 1. UIManager.

Скрипт, который отвечает за отображение информации о состояниях машины на пользовательском интерфейсе. Подписывается на изменения состояний машины и обновляет текстовые поля на основе этих изменений. На листинге 4 представлены поля класса, которые представляют текстовые элементы интерфейса, отображающие информацию о режиме и состоянии блокировки машины, а также `_statesOfMachine`, поле типа `StatesOfMachine`, которое представляет экземпляр класса, отвечающего за управление состояниями машины.

Метод `Construct` – это метод, используемый для внедрения зависимостей через `Zenject`. Атрибут `[Inject]` указывает на то, что этот метод будет вызван для внедрения зависимости `StatesOfMachine` в поле `_statesOfMachine`.

## Листинг 4 – Инициализация зависимостей в UIManager

```
public class UIManager : MonoBehaviour
{
    [SerializeField] TMP_Text textMode;
    [SerializeField] TMP_Text textBlock;

    private StatesOfMachine _statesOfMachine;
    [Inject]
    void Construct(StatesOfMachine statesOfMachine)
    {
        _statesOfMachine = statesOfMachine;
    }
}
```

В листинге 5 показана реализация режимов работы демонстрационного робота, вызывается метод Start при запуске объекта на сцене. Внутри метода происходит подписка на изменения состояний машины (StatesUpBody и TimerBlock).

Для каждой подписки используется метод Subscribe, который принимает лямбда-выражение для обработки полученного значения состояния. Для состояния StatesUpBody происходит изменение текста поля textMode в зависимости от текущего значения состояния.

Для состояния TimerBlock происходит изменение текста поля textBlock в зависимости от текущего значения состояния.

Метод AddTo(this) используется для добавления подписок к компоненту this (текущий объект UIManager), чтобы они автоматически отписывались при уничтожении объекта.

## Листинг 5 – Реализация основного функционала UIManager

```
public void Start()
{
    _statesOfMachine.StatesUpBody
        .Subscribe(state =>
        {
            switch (state)
            {
                case ((int)ModeRobot.Mode.Down) :
                    textMode.text = "Режим работы 1";
                    break;
                case ((int)ModeRobot.Mode.Up) :
                    textMode.text = "Режим работы 2";
                    break;

                case ((int)ModeRobot.Mode.Transport):
                    textMode.text =
```

## Продолжение листинга 5

```
"Режим работы 3";  
                                break;  
                                }  
        })  
        .AddTo(this);  
  
    _statesOfMachine.TimerBlock  
        .Subscribe(state =>  
        {  
            if (state)  
            {  
                textBlock.text = "Заблокировано";  
            }  
            else  
            {  
                textBlock.text = "Разблокировано";  
            }  
        })  
        .AddTo(this);  
    }
```

### 2. ChangeState.

Класс `ChangeState`, представлен в листинге 6, реализует интерфейс `ITrackableState<int>`, что позволяет отслеживать изменения состояния объекта. Внутри класса определено реактивное свойство `state`, которое хранит текущее состояние объекта. Это свойство доступно через `TrackState` в виде `IReadOnlyReactiveProperty<int>`, предоставляя другим компонентам доступ только для чтения к состоянию.

В методе `Update()` проверяются нажатия определенных клавиш (`ModeDown`, `ModeUp`, `ModeTransport`). При нажатии каждой клавиши значение `state` устанавливается в соответствующее значение перечисления `ModeRobot.Mode`.

Затем, при установке нового значения `state`, он автоматически отправляется (публикуется) всем подписчикам, которые подписались на это реактивное свойство.

## Листинг 6 – Реализация класса ChangeState

```
public class ChangeState : MonoBehaviour, ITrackableState<int>
{
    ReactiveProperty<int> state = new ReactiveProperty<int>();
    public IReadOnlyReactiveProperty<int> TrackState => state;
    public int MaxValue => throw new System.NotImplementedException();
    public int MinValue => throw new System.NotImplementedException();
    void Update()
    {
        if (Input.GetButtonDown("ModeDown"))
        {
            state.Value = (int)ModeRobot.Mode.Down;
        }
        if (Input.GetButtonDown("ModeUp"))
        {
            state.Value = (int)ModeRobot.Mode.Up;
        }
        if (Input.GetButtonDown("ModeTransport"))
        {
            state.Value = (int)ModeRobot.Mode.Transport; }
    }
}
```

### 3. TimerBlock.

Класс `TimerBlock` управляет таймером блокировки машины. Машина блокируется после определенного времени бездействия. Также реализует интерфейс `ITrackableState<bool>`, позволяющий `UIManager` отслеживать состояние блокировки.

Реализуем реактивное свойство типа `bool` в переменной `isBlock`, предоставлена в листинге 7, используется для отслеживания состояния блокировки, а свойство `TrackState` предоставляет доступ к `isBlock`. Другие части кода могут только читать значение `isBlock`, но не могут изменять его.

## Листинг 7 – Реализация реактивного свойства

```
private ReactiveProperty<bool> isBlock = new
ReactiveProperty<bool>();
public IReadOnlyReactiveProperty<bool> TrackState => isBlock;
```

В метод `Update` обновляет значение переменной `timer` каждый кадр:

- если значение `isBlock` равно `false`, увеличивает `timer` на время, прошедшее с прошлого кадра;
- если `timer` превышает 5 секунд, устанавливает `isBlock` в `true`, сбрасывает `timer` и вызывает событие `Notify` для оповещения о блокировке;
- проверяет нажатие клавиши "Пробел" или кнопки "Start" для разблокировки.

Методы `ResetTimer` и `Unblock`:

- `ResetTimer`: сбрасывает значение `timer` в ноль;
- `Unblock`: устанавливает `isBlock` в `false`, сбрасывает `timer` и вызывает событие `Notify` для оповещения об разблокировке.

#### 4. `StatesOfMachine`.

Класс `StatesOfMachine`, реализация которого представлена в листинге 8, является компонентом системы, отвечающим за отслеживание и управление состояниями объекта. Данный компонент является связующей точкой между разными классами с разными параметрами и интерфейсом.

Основные элементы класса:

- `StatesUpBody`: это свойство предоставляет доступ к состоянию тела объекта. Использует реактивное свойство `_statesUpBody.TrackState`, которое отслеживает текущее значение состояния;
- `TimerBlock`: это свойство предоставляет доступ к состоянию блокировки объекта. Использует реактивное свойство `_timerBlock.TrackState`, которое также отслеживает текущее значение состояния;
- `_statesUpBody` и `_timerBlock` приватные поля, хранящие ссылку на отслеживаемое состояние тела объекта и состояние блокировки;
- `_disposable` – объект `CompositeDisposable`, используемый для управления ресурсами, которые требуют освобождения при уничтожении объекта;
- `StatesOfDemRobot` – конструктор, который принимает в качестве параметров два отслеживаемых состояния: состояние тела объекта и состояние блокировки. Эти параметры внедряются с помощью `Zenject` с использованием атрибутов `[Inject(Id = "...")]`;
- `Initialize()` – функция, которая понадобится для расширения функциональности данного компонента.

Интерфейс `ITrackableState<T>` определяет общий интерфейс для отслеживаемых состояний.

## Листинг 8 – Реализация класса StatesOfMachine для управления состояниями

```
public class StatesOfMachine : IInitializable, IDisposable
{
    public IReadOnlyReactiveProperty<int> StatesUpBody =>
    _statesUpBody.TrackState;
    public IReadOnlyReactiveProperty<bool> TimerBlock =>
    _timerBlock.TrackState;

    private ITrackableState<int> _statesUpBody;
    private ITrackableState<bool> _timerBlock;

    private CompositeDisposable _disposable;

    public StatesOfDemRobot([Inject(Id = "ChangeStateRobot")]
        ITrackableState<int> statesUpBody,
        [Inject(Id = "TimerBlock")]
        ITrackableState<bool> timerBlock)
    {
        _statesUpBody = statesUpBody;
        _timerBlock = timerBlock;
    }
    public void Initialize()
    {
        _disposable = new CompositeDisposable();
    }
    public void Dispose()
    {
        _disposable.Dispose();
    }
}
```

### 5. StateMachineInstaller.

Класс StateMachineInstaller, как показано в листинге 9, устанавливает связи зависимостей в Zenject DI контейнере для интерфейсов ITrackableState<int> и ITrackableState<bool>, связывая их с соответствующими реализациями ChangeState и TimerBlock. Это обеспечивает доступ к состояниям и функциональности через DI контейнер.

## Листинг 9 – Класс StateMachineInstaller

```
public class StateMachineInstaller: MonoInstaller
{
    [SerializeField] ChangeState ChangeState;
    [SerializeField] TimerBlock TimerBlock;

    public override void InstallBindings()
    {
        Container.Bind<ITrackableState<int>>().WithId("ChangeState").FromInstance(ChangeState).AsCached();
    }
}
```

## Продолжение листинга 9

```
Container.Bind<ITrackableState<bool>>().WithId("TimerBlock").FromInstance(TimerBlock).AsCached();}}
```

Взаимодействие между компонентами:

- `UIManager` подписывается на изменения состояний машины (`StatesOfMachine`) и обновляет пользовательский интерфейс на основе этих изменений;
- скрипты `ChangeState` и `TimerBlock` генерируют события при изменении своих состояний, что позволяет `UIManager` реагировать на эти изменения и обновлять интерфейс;
- скрипт `StateMachineInstaller` связывает зависимости между интерфейсами `ITrackableState<int>` и `ITrackableState<bool>` и соответствующими реализациями, обеспечивая внедрение зависимостей.

Функционал может быть расширен путем добавления новых состояний и функциональности в компоненты `ChangeState` и `TimerBlock`. Например, можно добавить новые режимы работы машины или новые условия для блокировки/разблокировки. Каждый компонент должен быть расширен соответствующим образом, добавляя новые методы или подписываясь на дополнительные события.

### 3.4. Реализация всплывающих окон

#### 3.4.1. Экранная форма завершения миссии

В реализации окна завершения миссии нужно учесть такой функционал как отображение времени выполнения задания, возможность сделать новую попытку, выйти в главное меню. Класс `MenuOfEnd` является компонентом интерфейса, реализованного при помощи `Canvas`, он наследуется от `TaskPlot`, который содержится в системе управления переключения задач.

Класс `PlotScene` отвечает за активацию и переключение между задачами сюжета. При завершении каждой задачи вызывается соответствующий метод, который активирует следующую задачу или, если это последняя задача, вызывает команду, указывающую на завершение сюжета. Класс `MenuOfEnd` представляет

меню, которое активируется по завершении сюжета и позволяет перейти в главное меню симуляции. Общая логика управления задачами реализована в абстрактном классе TaskPlot, от которого наследуются конкретные задачи сюжета.

Основные компоненты процесса вызова окна завершения задачи показаны на диаграмме классов на рисунке 22. Класс MenuOfEnd представляет экранную форму завершения миссии, которое активируется по завершении задачи. Оно содержит методы для перехода в главное меню и управления его активностью на сцене. Класс TimerMission, представленный в листинге 11, отвечает за отображение времени выполнения текущей миссии и связан с экранной формой завершения миссии для вызова соответствующих действий. Класс RestartButton представляет кнопку перезапуска миссии.

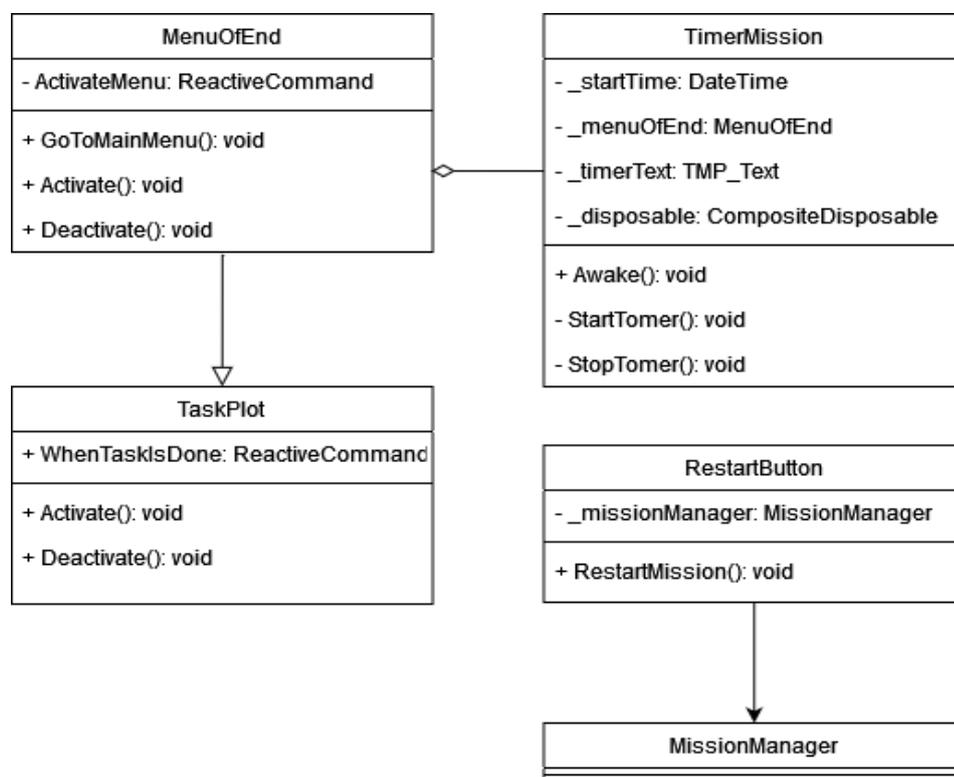


Рисунок 22 – Диаграмма классов для работы окна завершения задания

Рассмотрим класс MenuOfEnd, представленный в листинге 10.

- ReactiveCommand ActivateMenu: это команда, которая используется для активации меню завершения задачи.
- GoToMainMenu(): этот метод вызывается для перехода в главное меню. Он активирует событие WhenTaskIsDone, что указывает на завершение

текущей задачи. Затем метод проверяет текущую активную сцену и загружает сцену «MainMenuScene».

- **Activate():** метод активирует меню завершения задачи. Он делает указатель мыши видимым, устанавливает режим блокировки курсора в режим «None» и устанавливает активность объекта `gameObject` в `true`. Затем выполняется реактивная команда `ActivateMenu`, на которую подписан класс `TimerMission`, отвечающий за время выполнения задания.
- **Deactivate():** данный метод деактивирует меню завершения задачи. Он устанавливает активность объекта `gameObject` в `false`, скрывая меню завершения задачи.

#### Листинг 10 – реализация класса `MenuOfEnd`

```
public class MenuOfEnd : TaskPlot
{
    public ReactiveCommand ActivateMenu = new ReactiveCommand();
    public void GoToMainMenu()
    {
        WhenTaskIsDone.Execute();

        Scene currentScene = SceneManager.GetActiveScene();
    }
    public override void Activate()
    {
        Cursor.visible = true;
        Cursor.lockState = CursorLockMode.None;
        this.gameObject.SetActive(true);

        ActivateMenu.Execute();
    }

    public override void Deactivate()
    {
        this.gameObject.SetActive(false);
    }
}
```

#### Листинг 11 – реализация класса `TimerMission`

```
public class TimerMission : MonoBehaviour
{
    private DateTime _startTime;
    [SerializeField] private MenuOfEnd _menuOfEnd;
    [SerializeField] private TMP_Text _timerText;

    private CompositeDisposable _disposable = new
CompositeDisposable();
    void Awake()
```

## Продолжение листинга 11

```
        {
            StartTimer();

            _menuOfEnd.ActivateMenu
                .Subscribe(_ => {
                    StopTimer();
                })
                .AddTo(_disposable);
        }

        private void StartTimer()
        {
            _startTime = System.DateTime.Now;
        }

        private void StopTimer()
        {
            TimeSpan dateTime = System.DateTime.Now - _startTime;
            _timerText.text = $"Время выполнения задачи:
{string.Format("{0:00}:{1:00}",
                dateTime.Minutes,
                dateTime.Seconds)}";

            _disposable.Dispose();
        }
    }
```

- подписка, которая вызывает функцию в TimerMission из MenuOfEnd, происходит в методе Awake() класса TimerMission. Событие ActivateMenu из объекта \_menuOfEnd подписывается на метод StopTimer(). Когда ActivateMenu вызывается в MenuOfEnd, соответствующий метод StopTimer() в TimerMission будет вызван, завершая отсчет времени выполнения задачи;
- StopTimer(): Этот метод вычисляет разницу между текущим временем и временем начала выполнения задачи. Затем он форматирует это время в минуты и секунды и устанавливает полученное значение в текстовом поле \_timerText. После этого происходит отписка от всех событий.

Перезапуск задания происходит при нажатии на кнопку «Новая попытка» благодаря компоненту RestartButton, который представлен в листинге 12.

## Листинг 12 – Реализация класса RestartButton

```
public class RestartButton : MonoBehaviour
{
    [Inject] private MissionManager _missionManager;
```

## Продолжение листинга 12

```
public void RestartMission()

    {
        Cursor.visible = false;

        Cursor.lockState = CursorLockMode.Locked;
        Time.timeScale = 1.0f;
        _missionManager.LoadScene(SceneManager.GetActiveScene().name);
    }
}
```

При нажатии на кнопку вызывается метод `RestartMission()`, который скрывает курсор, разблокирует его, сбрасывает скорость времени и загружает текущую сцену миссии через `MissionManager`.

### 3.4.2. Экранная форма паузы

Класс `MenuOfPause` представляет собой меню паузы в симуляции, код класса представлен в листинге 13. Когда специалист вызывает меню паузы, он может выбрать опцию перехода в главное меню или закрыть само меню паузы. Меню паузы может отображаться и скрываться, в зависимости от текущего состояния задачи. Он также контролирует отображение и скрытие курсора, и управление временем через объект `Time.timeScale`. Класс использует `MissionManager` для загрузки главного меню.

- `GameObject _menuOfPause`: это ссылка на игровой объект, который содержит элементы меню паузы;
- `Camera_Rotation_CS camera_Rotation`: это ссылка на компонент, который управляет вращением камеры;
- метод `Construct`: это метод внедрения зависимости (`dependency injection`), используемый `Zenject` для внедрения `MissionManager`;
- метод `OnGoToMainMenu`: этот метод вызывается при выборе опции перехода в главное меню. Он устанавливает нормальную скорость времени (`Time.timeScale = 1.0f`) и загружает главное меню через `MissionManager`;
- метод `SwitchState`: этот метод переключает состояние меню паузы (отображено или скрыто). Если меню паузы не отображено, то метод отображает его, устанавливает курсор в видимое положение, останавливает

вращение камеры, замораживает время и активирует объект меню паузы. В противном случае метод скрывает меню паузы, скрывает курсор, возобновляет вращение камеры, восстанавливает время и деактивирует объект меню паузы;

- метод `CloseMenu`: этот метод вызывается для закрытия меню паузы. Он просто вызывает метод `SwitchState()` для переключения состояния меню;
- метод `Execute`: этот метод реализует интерфейс `ICommandUI` и вызывает метод `SwitchState()`. Он используется для обработки событий от пользовательского интерфейса.

### Листинг 13 – Реализация класса `MenuOfPause`

```
public class MenuOfPause : MonoBehaviour, ICommandUI
{
    private MissionManager _mm;
    public GameObject _menuOfPause;

    public Camera_Rotation_CS camera_Rotation;

    [Inject]
    void Construct(MissionManager mm)
    {
        _mm = mm;
    }
    public void OnGoToMainMenu()
    {
        Time.timeScale = 1.0f;

        _mm.LoadScene("MainMenuScene");
    }

    public void SwitchState()
    {
        if (!_menuOfPause.activeSelf)
        {
            Cursor.visible = true;
            Cursor.lockState = CursorLockMode.None;
            camera_Rotation.Pause(true);
            Time.timeScale = 0;
            _menuOfPause.SetActive(!_menuOfPause.activeSelf);
        } else
        {
            Cursor.visible = false;
            Cursor.lockState = CursorLockMode.Locked;
            camera_Rotation.Pause(false);
            Time.timeScale = 1.0f;
            _menuOfPause.SetActive(!_menuOfPause.activeSelf);
        }
    }
}
```

### Продолжение листинга 13

```
    }  
    }  
    public void CloseMenu()  
    {  
        SwitchState();  
    }  
  
    void ICommandUI.Execute()  
    {  
        SwitchState(); } }
```

#### 3.4.3. Реализация элементов задания

Для большинства видов спецтехники характерен общий тип задач на передвижение, для визуализации этого процесса в интерфейсе пользователя сделаем компоненты для отслеживания точности выполнения движений.

Для визуализации и отслеживания точности выполнения движений, мы создадим компонент MovingAccuracy, система в которой он участвует отображена на рисунке 23. Этот компонент будет отвечать за следующие функции:

- 1) Визуализация маршрута с помощью конусов: на протяжении маршрута будут размещены конусы для визуального представления пути.
- 2) Отслеживание задетых конусов: при соприкосновении с конусом, он будет подсвечен красным для обозначения того, что он был задет.
- 3) Расчет точности выполнения движений: по завершении маршрута будет выведена точность прохождения, которая будет отражать соотношение между количеством задетых конусов и общим числом конусов.

Компонент MovingAccuracy будет состоять из следующих элементов:

- список конусов: хранит информацию о всех конусах на маршруте;
- счетчик конусов: подсчитывает общее количество конусов на маршруте;
- счетчик задетых конусов: подсчитывает количество задетых конусов;
- методы для обработки столкновений с конусами: отслеживает столкновения с конусами и подсвечивает задетые конусы красным;
- метод для расчета точности выполнения движений: по завершении маршрута вычисляет процент задетых конусов от общего количества и выводит этот процент.

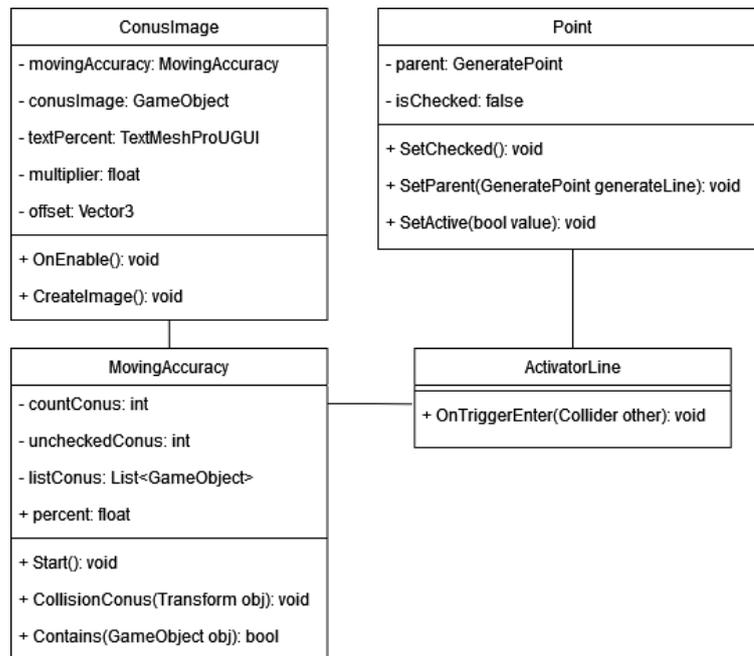


Рисунок 23 – Диаграмма классов для системы миссий с конусами

После интеграции компонента MovingAccuracy, код которого представлен в листинге 14, мы сможем получить информацию о точности выполнения движений и отобразить ее для дальнейшего анализа и улучшения навыков.

#### Листинг 14 – Реализация MovingAccuracy

```

public class MovingAccuracy : MonoBehaviour
{
    private int countConus = 0;
    private int uncheckedConus;

    private List<GameObject> listConus = new List<GameObject>();
    public float percent {
        get
        {
            if(countConus == 0) return 0;
            else return uncheckedConus / (float)countConus * 100;
        }
    }

    private void Start()
    {
        foreach (Transform child in transform)
        {
            child.OnTriggerEnterAsObservable().Where(x =>
x.gameObject.name == "ActivatorLine").FirstOrDefault().Subscribe(x =>
CollisionConus(child));
            countConus++;
        }
        uncheckedConus = countConus;
    }
}
  
```

## Продолжение листинга 14

```
private void CollisionConus(Transform obj)
{
    obj.gameObject.GetComponent<MeshRenderer>().material.color =
new Color(255, 0, 0);
    listConus.Add(obj.gameObject);

    uncheckedConus--;
}

public bool Contains(GameObject obj)
{
    return listConus.Contains(obj);
}
}
```

- **MovingAccuracy** содержит информацию о конусах, их количестве и текущем статусе;
- **ConusImage** используется для создания изображений конусов и отображения процента точности. **CreateImage** – функция отображения конусов в интерфейсе и расчета точности представлена в листинге 15.

## Листинг 15 – функция CreateImage

```
public void CreateImage()
{
    foreach (Transform child in movingAccuracy.transform)
    {
        GameObject image = Instantiate(conusImage, new Vector3(0, 0,
0), Quaternion.identity, transform);
        image.transform.localPosition = new
Vector3(child.transform.localPosition.x,
child.transform.localPosition.z, 0) * multiplier + offset;
        if (movingAccuracy.Contains(child.gameObject))
image.GetComponent<Image>().color = new Color(255, 0, 0);
    }
    textPercent.text = $"Процент точности:
{movingAccuracy.percent:#0.0}%";
}
```

- **ActivatorLine** и **Point** используются для отслеживания контакта между объектами в сцене, в данном случае для отслеживания контакта между конусами и активацией соответствующих методов в **MovingAccuracy**.

Таким образом, связь между ними состоит в том, что **ActivatorLine** и **Point** сообщают **MovingAccuracy** о контакте с конусами, а **ConusImage** отображает информацию о проценте точности на основе данных из **MovingAccuracy**.

### **Вывод по разделу 3**

Третья глава дипломной работы посвящена разработке интерфейса виртуального тренажера. Основное внимание уделено проектированию и реализации пользовательского интерфейса, что включает создание главного меню, различных всплывающих окон и интеграцию подсказок для пользователей.

Разработка главного меню обеспечивает быстрый доступ ко всем основным функциям тренажера, включая выбор миссий и настройки. Всплывающие окна, такие как экранная форма паузы и завершения задачи, организованы таким образом, чтобы предоставлять специалисту необходимую информацию во время использования тренажера.

Важной частью работы стало проектирование элементов интерфейса, которые напрямую влияют на удобство и функциональность использования тренажера. Реализация таких элементов как подсказки и индикаторы состояния помогает пользователю лучше ориентироваться в задачах.

#### **4. РАЗРАБОТКА И ТЕСТИРОВАНИЕ КОМПЛЕКСА ОБУЧАЮЩИХ СЦЕНАРИЕВ**

В рамках разработки комплекса сценариев реализуем различные задачи для одного из видов спецтехники – демонтажного робота. И проведем функциональное и нефункциональное тестирование.

Существует большое количество различных видов тестирования продуктов, но для тестирования обучающих сценариев мы выберем функциональное тестирование, так как оно нацелено на проверку функций и операций системы с точки зрения конечного пользователя. Функциональное тестирование помогает убедиться, что каждая функция программы выполняет заданные требования и корректно реагирует на разнообразные входные данные.

Преимущества функционального тестирования включают его простоту и эффективность в выявлении ошибок в поведении программы, что делает его идеальным выбором для проверки обучающих модулей. Это позволяет обеспечить, что обучающие модули будут работать так, как ожидается от них, предоставляя обучаемым корректную и последовательную обратную связь [14].

Нефункциональное тестирование – тестирование производительности, которое оценивает, как система работает под нагрузкой.

Демонтажный робот – это машина, предназначенная для разборки или сноса строений, старых автомобилей, и других объектов, где использование людей может быть опасным или неэффективным. Эти роботы обычно используются на стройках или в местах утилизации.

Основные части демонтажного робота:

1. Базовая платформа – это гусеничное шасси, которое обеспечивает мобильность робота.
2. Манипулятор – рука робота, которая может быть оснащена различными инструментами, такими как гидравлические клещи, дрели или камеры.
3. Лапы – это специальные устройства, которые позволяют роботу удерживаться на разных поверхностях.

4. Башня – это вращающаяся часть, которая обычно установлена на шасси робота. Она содержит основные рабочие механизмы, такие как манипуляторы и инструменты, и обеспечивает им поворот на 360 градусов для выполнения работы в различных направлениях без необходимости перемещения всего робота.

Перед реализацией нужно заложить базовые данные о состояниях работы техники, для робота это 3 режима:

- «НИЗ» – режим, в котором специалист управляет исключительно лапами и гусеницами;
- «ВЕРХ» – режим, в котором специалист башней и манипулятором;
- «ТРАНСПАТИРОВОЧНЫЙ» – режим, который используется для безопасной и удобной транспортировки робота с одного места на другое.

#### 4.1. Реализация сценария на движение молотом

При создании задания нужно ознакомить специалиста с возможностями работы манипулятора (стрелы), которая работает в режиме «ВЕРХ». Для этого можно реализовать задание, в котором нужно достигать стрелой определенной цели, например определенной точки в пространстве, при достижении которой будет появляться новая цель с другими координатами.

Разметим объект задачи, который представлен на рисунке 24, на сцене.



Рисунок 24 – Объект задачи для управления стрелой

После создания объектов на сцене их нужно объединить в группу и добавить в список заданий данного обучающего сценария. В интерфейсе пользователя, на

панели состояний будет отображено кол-во оставшихся целей, панель состояний показана на рисунке 25.

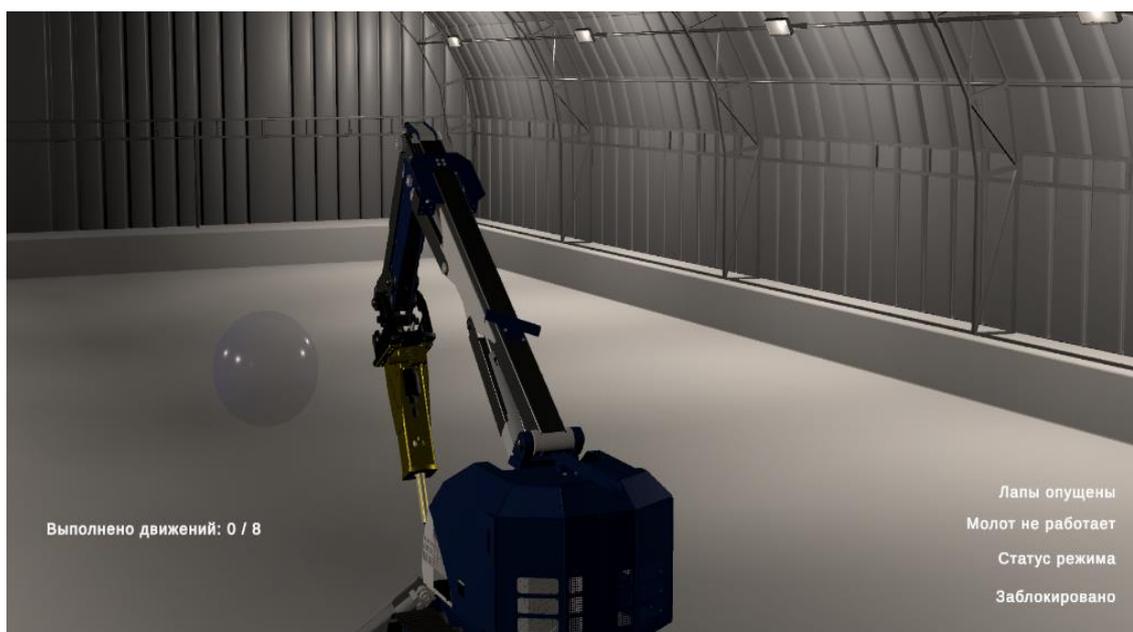


Рисунок 25 – Панель состояний сценария на управление стрелой

После настройки сцены и всех элементов интерфейса, добавим сценарий на сцену с главным меню и в UIController.

#### 4.1.1. Тестирование сценария на движение молотом

Функциональное тестирование сценария на движение молотом представлено в таблице 1.

Таблица 1 – Функциональное тестирование сценария на движение молотом

| Действие  | Результат   | Тест пройден? |
|---|---|---------------|
| Загрузка сцены из главного меню                               | Специалист успешно загрузил сценарий                        | Да            |
| Загрузка объектов, необходимых для выполнения задачи на сцене | Появляются необходимые объекты с заданными координатами     | Да            |
| Перемещение стрелы к начальной цели                           | Стрела достигает заданной точки в пространстве              | Да            |
| Отображение новой цели после достижения первой                | В пространстве появляется новая цель с другими координатами | Да            |
| Обновление информации на панели состояний                     | Панель состояний показывает оставшееся количество целей     | Да            |

Продолжение таблицы 1

|  |   |    |
|--|---|----|
| Завершение последовательности целей  | Все заданные цели достигнуты; открывается экранная форма с временем выполнения сценария | Да |
| Активация окна паузы через интерфейс   | При нажатии на заданную клавишу появляется экранная форма паузы                         | Да |
| Нажатие на кнопку «Начать заново» в окне паузы и окне завершения сценария        | Сценарий начинается заново с начальной точки  | Да |
| Нажатие на кнопку «Выйти в главное меню» в окне паузы и окне завершения сценария | Пользователь возвращается в главное меню  | Да |
| Нажатие на кнопку «Продолжить» в окне паузы                                      | Сценарий продолжается, манипулятор возобновляет движение к цели                         | Да |

#### 4.2. Реализация сценария на передвижение

Для обучения специалистов по управлению роботом-манипулятором в режиме «НИЗ», предлагается сценарий, в котором робот должен передвигаться по заданному пути, проходя через расставленные конусы и достигая определенных точек на маршруте. После успешного прохождения всего маршрута, на экране появляется экранная форма завершения, отображающее 2D карту пройденного пути и расположения конусов, время выполнения сценария, а также точность выполнения задания.

Для визуализации и более эффективного обучения, объекты задачи (конусы и точки) будут отмечены на сцене с точными координатами. Эти объекты затем объединяются в группу и добавляются в список заданий данного обучающего сценария, объекты представлены на рисунке 26.

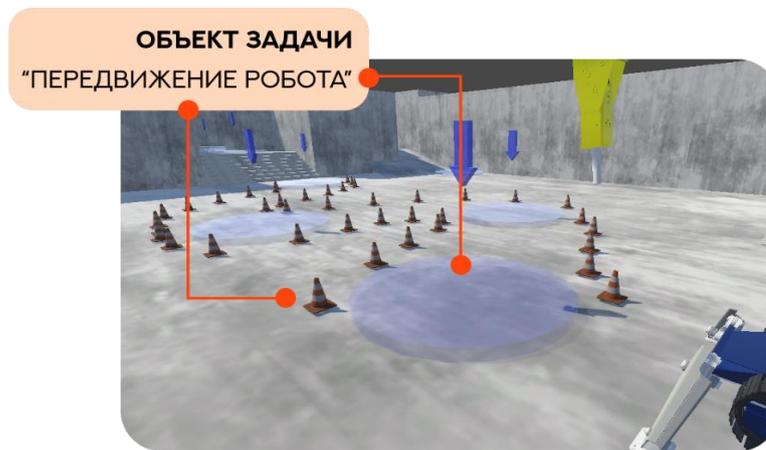


Рисунок 26 – Объекты сценария на передвижение техники

В пользовательском интерфейсе, на панели состояний, будет отображаться текущая задача, реализация панели состояний представлена на рисунке 27.

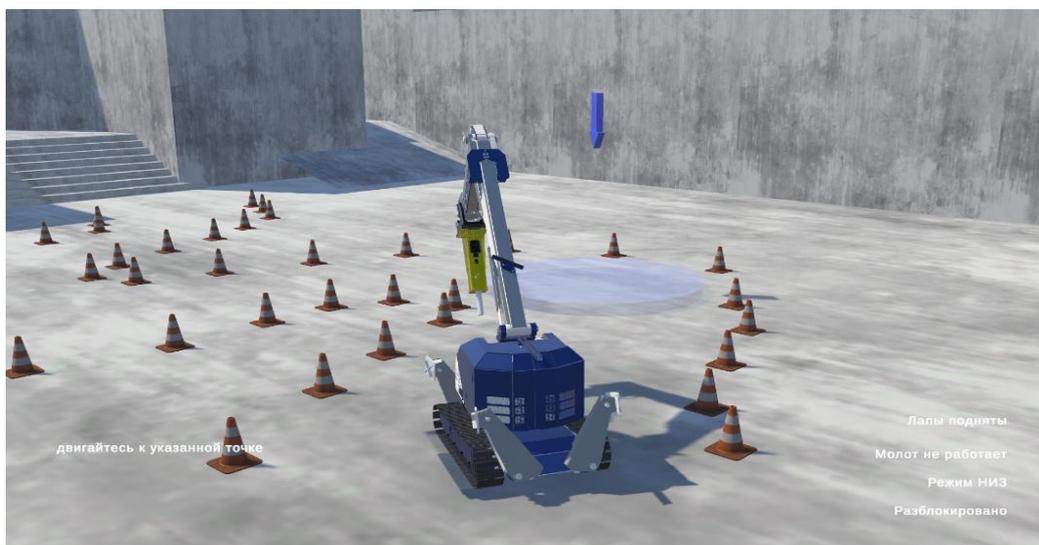


Рисунок 27 – Панель состояний сценария на передвижение техники

После настройки, добавим сценарий на сцену с главным меню.

#### 4.2.1. Тестирование сценария на передвижение

Таблица 2 – Функциональное тестирование сценария на передвижение техники

| Действие   | Результат  | Тест пройден? |
|--|--|---------------|
| Загрузка сценария из главного меню                                 | Специалист успешно загрузил сценарий                             | Да            |
| Загрузка объектов на сцену, включая конусы и маркеры точек на пути | На сцене появляются необходимые объекты с заданными координатами | Да            |
| Перемещение робота к первой точке                                  | Робот достигает первой точки, задача выполняется                 | Да            |

Продолжение таблицы 2

|  |  |    |
|--|--|----|
| Отображение новой цели после достижения первой                                   | На маршруте появляется новая цель с другими координатами   | Да |
| Обновление информации на панели состояний  | Панель состояний информирует о текущей задаче; отображает режим работы   | Да |
| Проверка точности выполнения пути  | Происходит расчет точности в зависимости задеты конусы или нет   | Да |
| Завершение последовательности целей  | Все заданные цели достигнуты; техника останавливается  | Да |
| Отображение окна завершения с 2D картой конусов                                  | На экране отображается 2D карта пройденного пути и расположения конусов; время выполнения; точность прохождения маршрута | Да |
| Активация окна паузы через интерфейс   | При нажатии на заданную клавишу появляется экранная форма паузы  | Да |
| Нажатие на кнопку «Начать заново» в окне паузы и окне завершения сценария        | Сценарий начинается заново с начальной точки   | Да |
| Нажатие на кнопку «Выйти в главное меню» в окне паузы и окне завершения сценария | Пользователь возвращается в главное меню   | Да |
| Нажатие на кнопку «Продолжить» в окне паузы                                      | Сценарий продолжается, техника возобновляет движение   | Да |

### 4.3. Нефункциональное тестирование

Одним из требований при разработке виртуального тренажера было то, что количество кадров в секунду (Frames Per Second) не будет опускаться ниже 30. При тестировании сценариев мы можем отслеживать данный параметр через встроенный инструмент статистики Unity.

Тестирование проводилось на персональном компьютере, конфигурация которого включала в себя процессор Intel Core i5 с тактовой частотой около 3.0 ГГц, графический ускоритель NVIDIA GeForce GTX 1660, 8 гигабайт оперативной памяти DDR4 и жесткий диск емкостью 500 гигабайт. Из результатов, представленных на рисунке 28, можно увидеть, что диапазон значений кадров в

секунду не опускался ниже 76 FPS, что соответствует заданным требованиям для симуляции.

| Statistics                            |                   | Statistics                           |                     |
|---------------------------------------|-------------------|--------------------------------------|---------------------|
| <b>Audio:</b>                         |                   | <b>Audio:</b>                        |                     |
| Level: -74.8 dB                       | DSP load: 0.1%    | Level: -74.8 dB                      | DSP load: 0.1%      |
| Clipping: 0.0%                        | Stream load: 0.0% | Clipping: 0.0%                       | Stream load: 0.0%   |
| <b>Graphics:</b>                      |                   | <b>Graphics:</b>                     |                     |
| 76.2 FPS (13.1ms)                     |                   | 157.3 FPS (6.4ms)                    |                     |
| CPU: main 13.1ms render thread 10.9ms |                   | CPU: main 6.4ms render thread 4.7ms  |                     |
| Batches: 3076 Saved by batching: 0    |                   | Batches: 1133 Saved by batching: 713 |                     |
| Tris: 5.5M                            | Verts: 5.7M       | Tris: 1.1M                           | Verts: 1.2M         |
| Screen: 1920x1080 - 23.7 MB           |                   | Screen: 1920x1080 - 23.7 MB          |                     |
| SetPass calls: 2772                   | Shadow casters: 0 | SetPass calls: 100                   | Shadow casters: 663 |
| Visible skinned meshes: 0             |                   | Visible skinned meshes: 0            |                     |
| Animation components playing: 0       |                   | Animation components playing: 0      |                     |
| Animator components playing: 0        |                   | Animator components playing: 0       |                     |

Рисунок 28 – Встроенный инструмент статистики Unity, показатели FPS (минимальное и максимальное значение)

#### Вывод по разделу 4

Четвертый раздел посвящен разработке и тестированию комплекса обучающих сценариев. В рамках этой главы были разработаны различные сценарии, каждый из которых направлен на отработку конкретных навыков оператора.

Сценарии включают в себя задания по маневрированию, точности движения и управлению различными функциями машины. Для каждого сценария проведены функциональные тесты, результаты которых подтверждают, что заданные цели достигнуты, а функции тренажера работают корректно. Это подтверждается успешной реализацией и проверкой сценариев, таких как движение стрелой и передвижение техники по заданной траектории.

## ЗАКЛЮЧЕНИЕ

Анализ предметной области показал, что аналогичные разработки существуют, однако, представленные на рынке продукты, на данный момент, не могут быть использованы на отечественном производстве.

Анализ требований и технологических возможностей показывает, что разработка удобных и функциональных интерфейсов для виртуальных тренажеров является ключевой для подготовки квалифицированных специалистов в различных отраслях. В работе был разработан интерфейс виртуального тренажера для управления спецтехникой, что позволяет эффективно обучать операторов без риска для жизни и здоровья. Основная проблема заключалась в том, что сложно адаптировать интерфейс под разнообразные задачи и типы техники. Эта проблема была решена за счёт использования гибкой архитектуры и масштабируемого дизайна интерфейса.

Архитектура «Модель-Представление-Контроллер», которая используется в симуляции, обеспечивает быструю и гибкую настройку тренажера под конкретные нужды обучения за счет адаптивного дизайна и возможности интеграции различных типов управления и сценариев. Это достигается при помощи использования современных подходов к проектированию пользовательских интерфейсов и внедрения универсальных компонентов, облегчающих разработку и последующую поддержку системы.

Тестирование интерфейса, в разных сценариях, показало его эффективность и удобство использования. Применялись различные методы тестирования, включая функциональное и нефункциональное тестирование, в результате которых была проверена работоспособность тренажера и всех его функций.

Таким образом, была реализована модель виртуального тренажера, разработанный интерфейс поддерживает различные режимы обучения и предоставляет пользователю интуитивно понятные инструменты управления, что способствует быстрому освоению необходимых навыков.

## БИБЛИОГРАФИЧЕСКИЙ СПИСОК

1. Российская ассоциация производителей специализированной техники и оборудования [Электронный ресурс]. – URL: <https://rosspetsmash.ru/> (дата обращения: 20.12.2023).
2. Симулятор экскаватора «CYBERMINE» [Электронный ресурс]. – URL: <https://www.thoroughtec.com/ru/cybermine-симуляторы-лопаты-и-экскаватора/> (дата обращения: 20.12.2023).
3. CM Labs Simulation Software & Heavy Equipment Simulation [Электронный ресурс]. – URL: <https://www.cm-labs.com/en/> (дата обращения: 20.12.2023).
4. Учебный тренажер-симулятор бульдозера ЧЕТРА [Электронный ресурс]. – URL: <https://service-im.ru/training/oborudovanie-dlya-obucheniya.php> (дата обращения: 20.12.2023).
5. Графический редактор Figma [Электронный ресурс]. – URL: <https://www.figma.com> (дата обращения: 20.12.2023).
6. Unreal Engine [Электронный ресурс]. – URL: <https://www.unrealengine.com/en-US> (дата обращения: 20.12.2023).
7. Godot [Электронный ресурс]. – URL: <https://godotengine.org/> (дата обращения: 20.12.2023).
8. Cryengine [Электронный ресурс]. – URL: <https://www.cryengine.com/> (дата обращения: 20.12.2023).
9. Кроссплатформенная среда разработки Unity [Электронный ресурс]: Unity User Manual. – URL: <https://docs.unity3d.com/Manual/index.html> (дата обращения: 20.12.2023).
10. Zenject [Электронный ресурс]. – URL: <https://github.com/modesttree/Zenject> (дата обращения: 12.02.2024 г.).
11. Библиотека для реактивного программирования [Электронный ресурс] : Репозиторий и документация UniRx. – URL: <https://github.com/neuecc/UniRx> (дата обращения: 20.12.2023).

12. Приемы объектно ориентированного проектирования. Паттерны проектирования : справочник / Э. Гамма, Р. Хелм, Р. Джонсон, Д. Влиссидес. – Москва : ДМК Пресс, 2007. – 368 с. – ISBN 5-93700-023-4. – Текст : электронный // Лань : электронно-библиотечная система. – URL: <https://e.lanbook.com/book/1220> (дата обращения: 26.12.2023).
13. Документация UI toolkit [Электронный ресурс]. – URL: <https://docs.unity3d.com/Manual/UIElements.html> (дата обращения: 20.12.2023).
14. Бейзер Б. Тестирование черного ящика: Технологии функционального тестирования программного обеспечения и систем. // Питер. 2004. – 320 с.