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 программного модуля для приема и обработки команд от манипулятора ручного управления бульдозером

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА К ВЫПУСКНОЙ КВАЛИФИКАЦИОННОЙ РАБОТЕ

ЮУРГУ-090301.2024.308/384 ПЗ ВКР

Руководитель работы,
к.п.н., доцент каф. ЭВМ
_____ Ю.Г. Плаксина
«___»_____2024 г.

Автор работы,
студент группы КЭ-405
_____М.С. Леонов
«___»_____2024 г.

Нормоконтролёр,
ст. преп. каф. ЭВМ
_____ С.В. Сяськов
«___»_____2024 г.

Челябинск-2024

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
Леонов Максим Сергеевич
обучающемуся по направлению
09.03.01 «Информатика и вычислительная техника»

Тема работы: «Разработка программного модуля для приема и обработки команд от манипулятора ручного управления бульдозером» утверждена приказом по университету от «__» _____ 2024 г. №

Срок сдачи студентом законченной работы: 01 июня 2024 г.

Исходные данные к работе:

1. Манипулятор ручного управления бульдозером.
2. Требования к функционалу разрабатываемого приложения:
    2.1. Возможность управления с помощью манипулятора.
    2.2. Симуляция движения элементов бульдозера.
3. Системные требования:
    3.1. Операционная система: не ниже Windows 10.
    3.2. Процессор: i5-3570 3.4 GHz 4 Core.
    3.3. Оперативная память: 4 ГБ ОЗУ.

3.4. Видеокарта: Nvidia GTX 1050-ti или эквивалентная производительность.

Перечень подлежащих разработке вопросов:

1. Аналитический обзор литературы по работе с инструментами реализации, сравнение разрабатываемого продукта с аналогами;
2. Определение требований и проектирование и программного модуля для приема, обработки команд от манипулятора ручного управления бульдозером;
3. Реализация программного модуля для приема, обработки команд от манипулятора ручного управления бульдозером.
4. Тестирование программного модуля для приема, обработки команд от манипулятора ручного управления бульдозером.

Дата выдачи задания: ___ декабря 2023 г.

Руководитель работы _____/ *Ю.Г. Плаксина /*

Студент _____/ М.С. Леонов /

КАЛЕНДАРНЫЙ ПЛАН

Этап	Срок сдачи	Подпись руководителя
Введение и обзор литературы	29.02.2024	
Проектирование и определение требований программного модуля для приема, обработки команд от манипулятора ручного управления бульдозером	03.03.2024	
Реализация программного модуля для приема, обработки команд от манипулятора ручного управления бульдозером	24.03.2024	
Тестирование программного модуля для приема, обработки команд от манипулятора ручного управления бульдозером	07.04.2024	
Компоновка текста работы и сдача на нормоконтроль	24.05.2024	
Подготовка презентации и доклада	30.05.2024	

Руководитель работы _____/ *Ю.Г. Плаксина* /

Студент _____/ *М.С. Леонов* /

АННОТАЦИЯ

М.С.Леонов «Разработка программного модуля для приема и обработки команд от манипулятора ручного управления бульдозером» Челябинск: ФГАОУ ВО «ЮУрГУ (НИУ)», ВШ ЭКН; 2024, 48 с., 18 ил., библиогр. список – 15 наим.

Выпускная квалификационная работа посвящена разработке программного модуля для приема и обработки команд от манипулятора ручного управления бульдозером для виртуального тренажера управления спецтехникой с использованием кроссплатформенной среды разработки Unity.

Проведен анализ предметной области, рассмотрены аналоги разрабатываемого симулятора. Обоснованы функциональные и нефункциональные требования, а также освещены вопросы выбора библиотек. В модуле управления будут использоваться Input Manager, встроенный в Unity и библиотека “System.IO.Ports”. Для модуля управления применен шаблон проектирования "Команда".

После анализа реализован программный код модуля для приема, обработки команд от манипулятора ручного управления бульдозером. Проведено функциональное тестирование на соответствие требованиям.

ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ

Симулятор — это устройство или программное обеспечение, созданное для имитации управления определенным процессом, механизмом или транспортным средством. Термин "симулятор" чаще всего ассоциируется с компьютерными программами, особенно с видеоиграми. В компьютерных симуляторах, которые точно воспроизводят интерьер кабины соответствующего устройства, производится тренировка специалистов, таких как пилоты, космонавты или машинисты высокоскоростных поездов.

Симуляторы представляют собой средства, как программные, так и аппаратные, которые создают впечатление реальности, передавая часть реальных явлений и свойств в виртуальной среде. Компьютерные эксперименты часто используются для изучения имитационных моделей. Симулирование также применяется в научном моделировании природных систем или человеческих систем для получения представления о их функционировании. Моделирование может быть полезным для демонстрации возможных эффектов различных условий и вариантов действий. Также симуляция используется в случаях, когда реальная система недоступна, опасна, еще не построена или вовсе не существует.

Часто на производстве используется спецтехника, механизм которой включает в себя множество рычагов управления со специальным назначением для каждого. Научится управлять такой техникой в реальной жизни крайне сложно, так как требует значительного опыта. В связи с чем актуально разработка программной системы, позволяющей имитировать работу сложных механизмов, то есть рычагов управления, что позволит существенно сократить время для обучения специалистов управления бульдозером.

1. АНАЛИТИЧЕСКИЙ ОБЗОР НАУЧНО-ТЕХНИЧЕСКОЙ ЛИТЕРАТУРЫ

Существует множество симуляторов дорожно-строительной техники. Рассмотрим несколько из них.

Рисунок 1 – Учебный симулятор бульдозера "Sim Bulldozer 2021"

На рисунке 1 изображен – Учебный симулятор бульдозера "Sim Bulldozer 2021", разработчик simgenix [1]. У данного продукта присутствует несколько достоинств: имеются различные сценарии, возможность перемещения грунта, создание насыпей, планирование. Реализовано простое изменение ландшафта. Также присутствуют некоторые недостатки. Это отсутствие поддержки очков виртуальной реальности, отсутствие поддержки подвижной платформы. А также могут возникнуть некоторые проблемы с покупкой, потому что разработчик не из России.

Рисунок 2 – Dozer Simulator Training Pack

На рисунке 2 изображен – Dozer Simulator Training Pack, разработчик CM Labs Simulations [2]. У Dozer Simulator Training Pack присутствует несколько достоинств. Созданы различные сценарии. Это перемещение грунта, создание насыпей, планирование, выравнивание площадки, погрузка техники, движение по маршруту. Это лучше перемещение грунта в сравнении с прошлым аналогом. После выполнения сценариев показывается статистика оператора. Присутствуют аналогичные недостатки. Это отсутствие поддержки очков виртуальной реальности, отсутствие поддержки подвижной платформы. А также могут возникнуть некоторые проблемы с покупкой, потому что разработчик не из России.

Рисунок 3 – Кабина симулятора Четра

Рисунок 4 – Симулятор Четра

На рисунках 3 и 4 изображен симулятора Четра [3]. У следующего продукта присутствует несколько достоинств. Созданы различные сценарии, подготовка

машины к работе, перемещение по маршруту. Также управление машиной на месте работы: на равнине, на подъеме 30 градусов, на склоне до 20 градусов, заезд на трал на различных грунтах, в разные периоды года и суток, в различных условиях освещенности, в различных погодных условиях. Это перемещение грунта. В процессе обучения показываются и записываются ошибки. Из недостатков отсутствие поддержки очков виртуальной реальности. Симуляция сделана только для бульдозеров модели «Четра». Большая потребляемая мощность.

В таблице 1 приведено сравнение существующих аналогов. В сравнение проведено по критериям, отсутствующим у аналогов.

Таблица 1 – Сравнение существующих аналогов

Симулятор	Поддержка очков виртуальной реальности	Поддержка подвижной платформы	Различные сценарии
Учебный симулятор бульдозера "Sim Bulldozer 2021"	Нет	Нет	Нет
Dozer Simulator Training Pack	Нет	Да	Да
Кабина симулятора Четра	Нет	Да	Да

Выводы по разделу один

После проведенного анализа симуляторов дорожно-строительной техники были выявлены общие недостатки. Они не имеют поддержки очков виртуальной реальности. Также есть частные недостатки. Это отсутствие поддержки подвижной платформы или разработка симулятора только для бульдозеров определенной техники. Из-за этих недостатков была начата разработка учебного симулятора бульдозера, для которого необходимо создать программный модуль для приема, обработки команд от манипулятора ручного управления бульдозером.

2. ОПРЕДЕЛЕНИЕ ТРЕБОВАНИЙ. ПОСТАНОВКА ЗАДАЧИ. ПРОЕКТИРОВАНИЕ.

Требуется спроектировать модуль для приема, обработки команд от манипулятора ручного управления бульдозером. На основании проведенного аналитического обзора и требований заказчика можно выделить функциональные и нефункциональные требования к программному модулю для приема, обработки команд от манипулятора ручного управления бульдозером.

2.1. Функциональные требования

1. Программный модуль должен принимать сигналы от управляющего устройства.
2. Должна быть возможность переключения между устройствами в реальном времени.
3. Управление движением ковша и бульдозера с управляющего устройства.
4. При нажатии на кнопку должно переключаться управление между ковшом и рыхлителем.
5. В программный модуль передвижения бульдозером, должна отправляться информация о состоянии манипулятора (значение четырех осей).

2.2. Требования к функционалу системы приема сигналов

1. Необходимо определить четыре оси для управления. Две оси на каждом джойстике. Вперед и назад, вправо и влево.
2. Значение каждой оси изменяется от -1 до 1.
3. Кнопка для переключения между ковшом и рыхлителем.

2.3. Требования к функционалу системы движения ковша и рыхлителя

1. При нажатии соответствующих кнопок на клавиатуре или движение джойстика должны двигаться ковш или рыхлитель.
2. У ковша и рыхлителя должны быть ограничения поворота.
3. Возможность настраивать ограничения поворота ковша и рыхлителя.
4. Вместе с ковшом или рыхлителем должны двигаться гидроцилиндры, прикрепленные к ним.
5. Кнопка должна переключать управление на ковш или рыхлитель.

6. Возможность получить состояние ковша и рыхлителя для системы миссий и интерфейса.

2.4. Нефункциональные требования

1. Программный модуль должен быть написана на Unity.

2. Программный модуль должен использоваться на операционной системе Windows.

3. Время обработки сигнала от манипулятора не должно составлять более 25 мс.

4. Подключаемые устройства должны быть USB или подключатся с использованием последовательного порта.

2.5. Выбор используемых библиотек для модуля управления бульдозера

В рамках данного проекта симулятор будет работать с двумя типами внешних устройств: USB-устройствами (клавиатура) и устройствами, подключаемыми через последовательный порт (джойстик).

Для работы с USB-устройствами необходимо использовать встроенную систему управления в Unity [4]. Последовательный порт передает информацию посылками в один или более байтов за один раз. Этот набор байтов содержит информацию о состоянии джойстика [5]. Для работы с последовательным портом необходимо использовать библиотеку под названием “System.IO.Ports” [6].

2.6. Выбор используемых библиотек для модуля движения элементов бульдозера

Также необходимо обрабатывать полученные сигналы от внешних устройств для корректного измения состояния элементов бульдозера.

В Unity существует ряд инструментов для визуализации изменения объектов, однако в данном случае прямая кинематика, используемая по умолчанию, не подходит, так как каждый объект анимации будет двигаться независимо от других, что не соответствует задаче [7].

Вместо этого потребуется использовать инверсную кинематику. Инверсная кинематика – это когда при движение каждого объекта зависит от движения других объектов [8]. Unity имеет встроенную поддержку инверсной кинематики. Этот инструмент можно использовать, но у него есть более серьезный минус. Для

использования данного инструмента на каждом элементе анимации должен присутствовать компонент “Rigidbody” [9]. Этот компонент позволяет объектам взаимодействовать с помощью физики. Использование физики в анимации может вызвать проблемы с точностью анимации. Это происходит из-за того, что физика рассчитывается с определенной периодичностью и между этими промежутками может произойти какое-то изменение, что может создать непредсказуемые результаты, поэтому откажемся от данного способа.

Еще один инструмент — это “Joints” [10], которые позволяют создавать соединения между объектами на основе физики. Однако здесь такие же проблемы использования физики в анимации, так что этот инструмент не подходит.

Следующий вариант — это создание собственного инструмента, использующего инверсную кинематику [8], чтобы обеспечить корректное движение объектов в симуляторе. Будем перемещать объекты напрямую через скрипты. Необходимо будет перемещать объекты с учетом положения связанных объектов.

2.7. Проектирование

Манипулятор ручного управления бульдозером должен взаимодействовать с множеством модулей техники (отвал, рыхлитель, управление движением бульдозера, позиционирование движения виртуальных джойстиков). На рисунке 5 показана общая схема модуля для приема и обработки команд от манипулятора ручного управления бульдозером. Управление ковшом и рыхлителем можно объединить так как управление ими одинаковое и между ними будет происходить переключение.

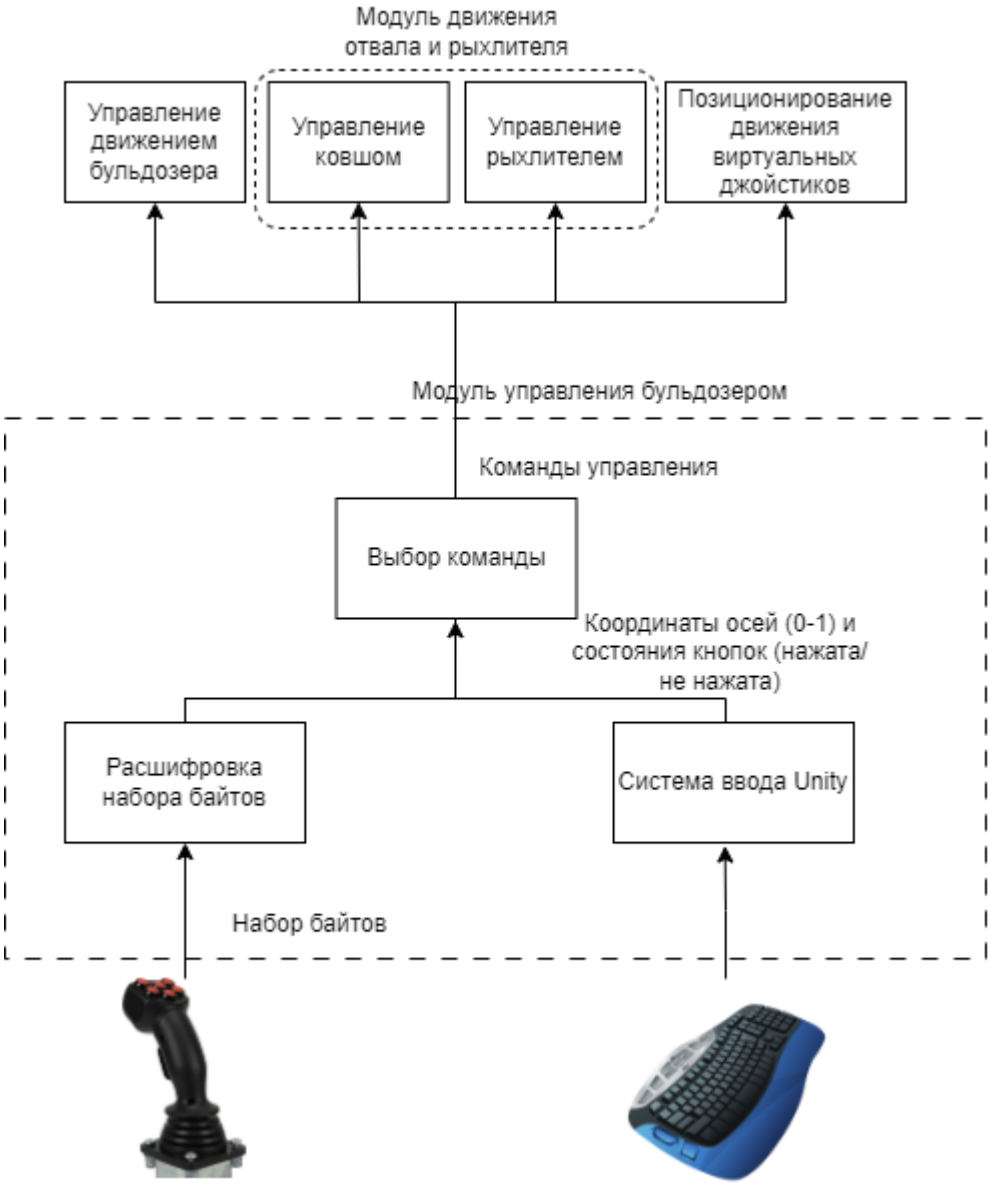

Рисунок 5 – Общая схема модуля для приема и обработки команд от манипулятора ручного управления бульдозером

Система ввода позволяет определить оси ввода и связанные с ними действия для модуля, такую возможность дает функционал Unity. Система ввода использует следующие типы элементов управления:

- Клавиша — это любая клавиша на физической клавиатуре, например W, Shift или пробел. Под кнопкой понимается любая кнопка на физическом контроллере (например, геймпадах), например кнопка X на контроллере Xbox One.
- Виртуальная ось (во множественном числе: оси) сопоставляется с элементом управления, например кнопкой или клавишей. Когда

пользователь активирует элемент управления, ось получает значение в диапазоне [–1..1]. Это значение можно использовать в программном коде [11].

От контроллера джойстика приходит набор байтов. В этих байтах содержится информация о координатах осей и состоянии кнопок. Из этого набора необходимо извлечь нужны байты, находящиеся после контрольного байта. Контрольный байт — это байт, после которого находится нужная информация. Контрольный байт будет задан в коде.

Рассмотрим требования модуля управления бульдозером. Необходимо иметь возможность управлять техникой с помощью клавиатуры и джойстика, так как у разработчиков должна быть возможность протестировать симулятор без джойстика. Система должна переключаться в реальном времени между устройствами ввода. Необходима уникальная настройка управления для каждого устройства ввода, так как управление с клавиатуры и джойстика сильно отличается. У джойстика имеются оси, которые имеют значение от 0 до 1, у клавиатуры есть только два значения: нажата и не нажата.

Существуют несколько шаблонов программирования, которые могут помочь реализовать модуль управления бульдозером: "Цепочка Обязанностей" [12] и "Команда" [13].

Шаблон проектирования "Цепочка Обязанностей" [12] позволяет создать цепочку из возможных обработчиков запросов, где отправитель запроса не привязан к конкретному получателю. Запрос перемещается вдоль цепочки обработчиков, каждый из которых может либо обработать запрос, либо передать его следующему объекту в цепочке. Это способствует гибкой иерархии обработки запросов, где каждый объект может принять решение о дальнейшей обработке или передаче запроса следующему звену цепочки. Схематичное представления паттерна изображено на рисунке 6.

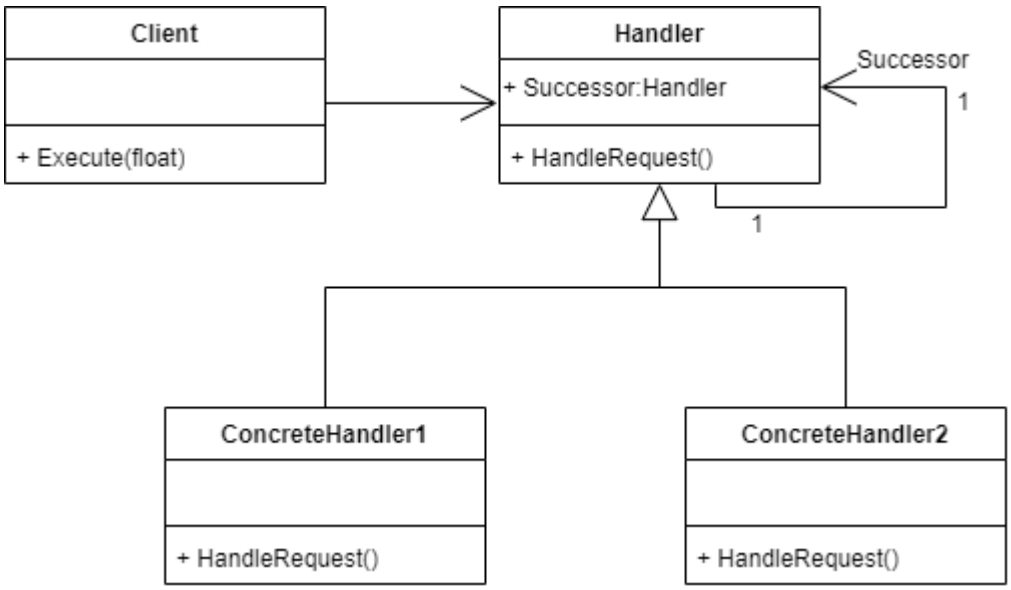

Рисунок 6 – Схематичное представления шаблона проектирования "Цепочка Обязанностей"

Шаблон проектирования "Команда" [13] позволяет создавать объекты, которые представляют собой запросы на выполнение определенных действий. Такие объекты могут быть отправлены другим объектам для выполнения этих действий, при этом отделяясь от самого выполнения. Это позволяет достичь инкапсуляции запросов на выполнение действий и отделить объекты, инициирующие запросы, от объектов, осуществляющих их выполнение. Схематичное представления паттерна изображено на рисунке 7.

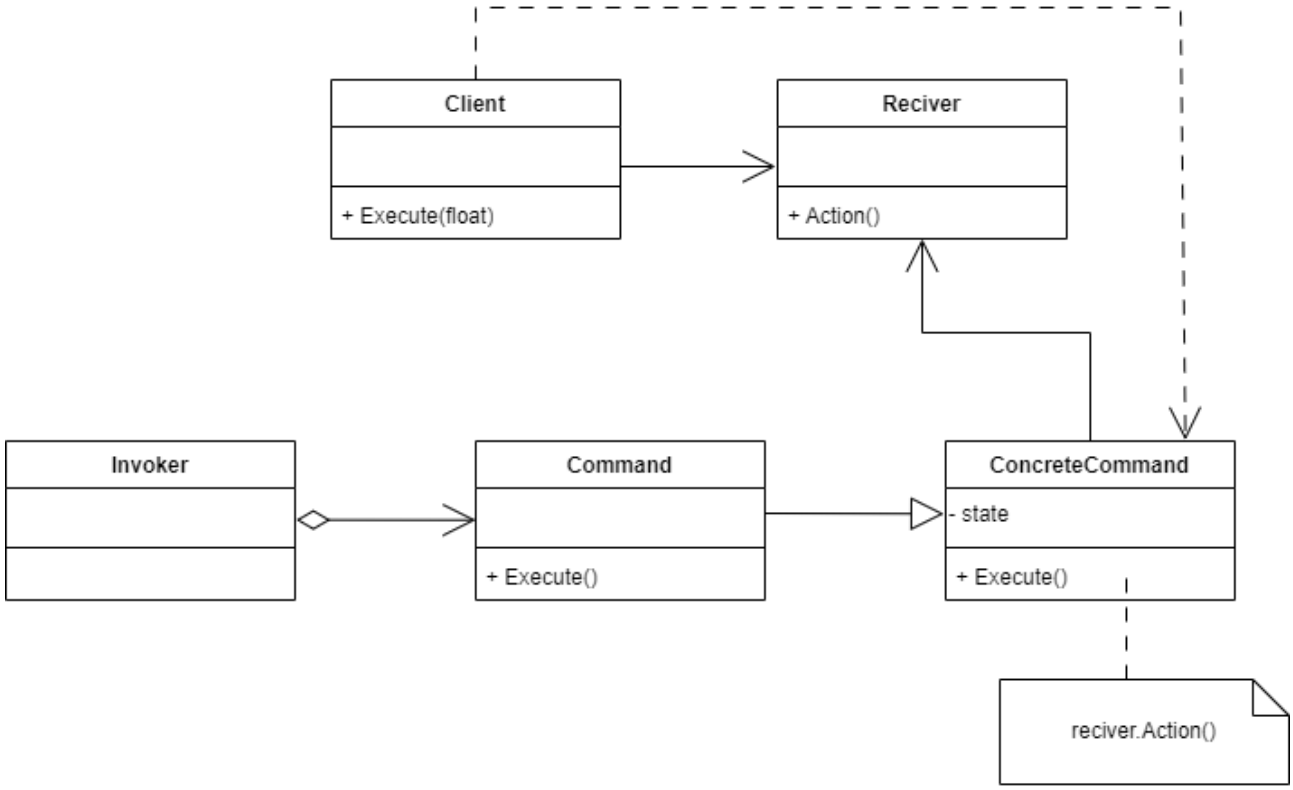

Рисунок 7 – Схематичное представления шаблона проектирования "Команда"

Из двух приведенных шаблонов проектирования больше подходит шаблон "Команда". Шаблон "Цепочка Обязанностей" используется, когда определенный запрос может обработать более одного объекта. Необходимо менять логику приема сигнала от управляющего устройства. Поэтому шаблон "Цепочка Обязанностей" не подходит.

Одним из важных преимуществ использования шаблона проектирования "Команда" является возможность замены команд на другие. Это означает, что можно создавать различные команды для управления объектами, такие как команды для управления с клавиатуры, джойстика и других устройств. После этого можно легко подставлять необходимые команды в коде без изменения самого кода управляемого объекта. В командах будут передаваться значения координат осей и состояния кнопок.

На основе выбранного шаблона проектирования можно составить диаграмму потоков данных. Диаграмма потоков данных модуля управления бульдозером приведена на рисунке 8.

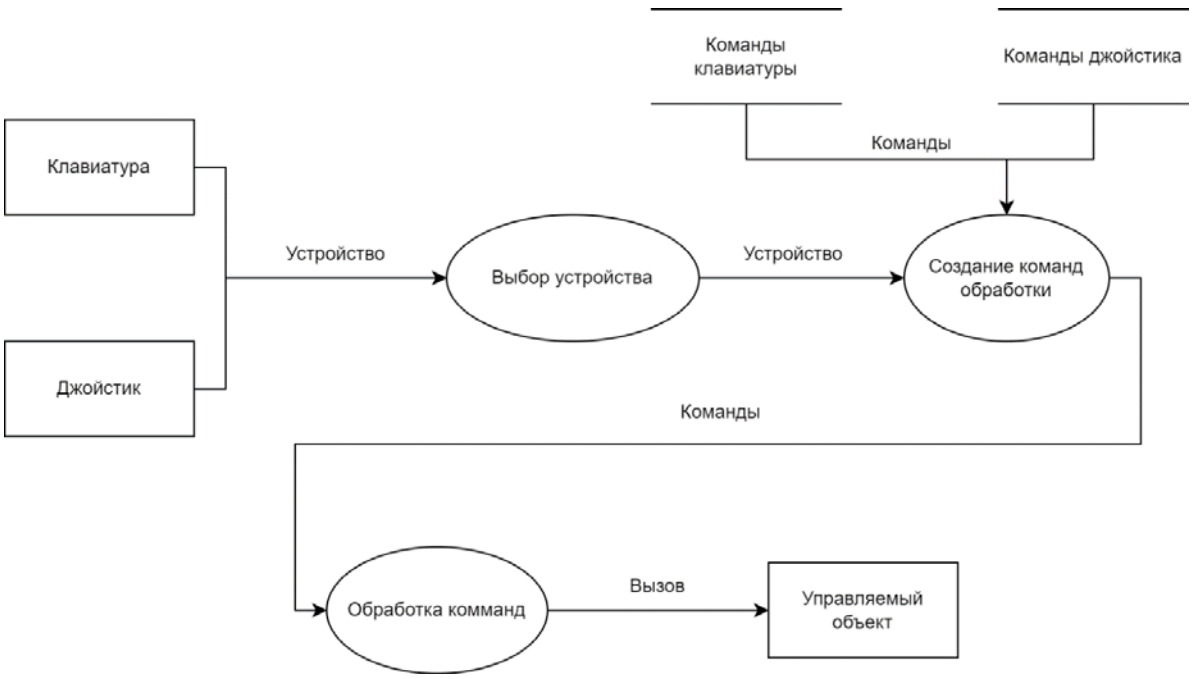

Рисунок 8 – Диаграмма потоков данных модуля управления бульдозером

Далее рассмотрим модуль движения отвала и рыхлителя. На рисунке 9 приведена диаграмма потоков данных модуля. Необходимо поворачивать объект, когда поступает команда. Будем изменять ось поворота объекта (отвала, рыхлителя). Совместно с поворотом отвала и рыхлителя также должны изменять положение гидроцилиндра на поршень. Необходимо следить за углом поворота, так как у элементов существует ограничение поворота. У модели техники должны правильно настроены оси поворота. Отвал должен двигаться как на рисунках 10 и 11.

Рисунок 9 – Диаграмма потоков данных модуля движения отвала и рыхлителя

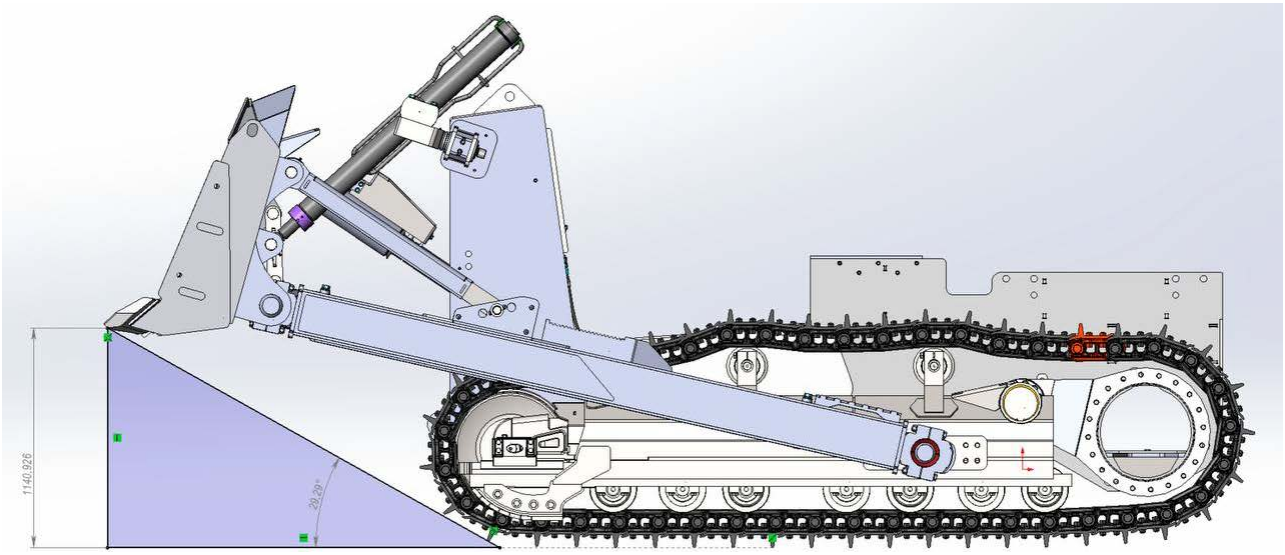

Рисунок 10 – Верхнее положение отвала

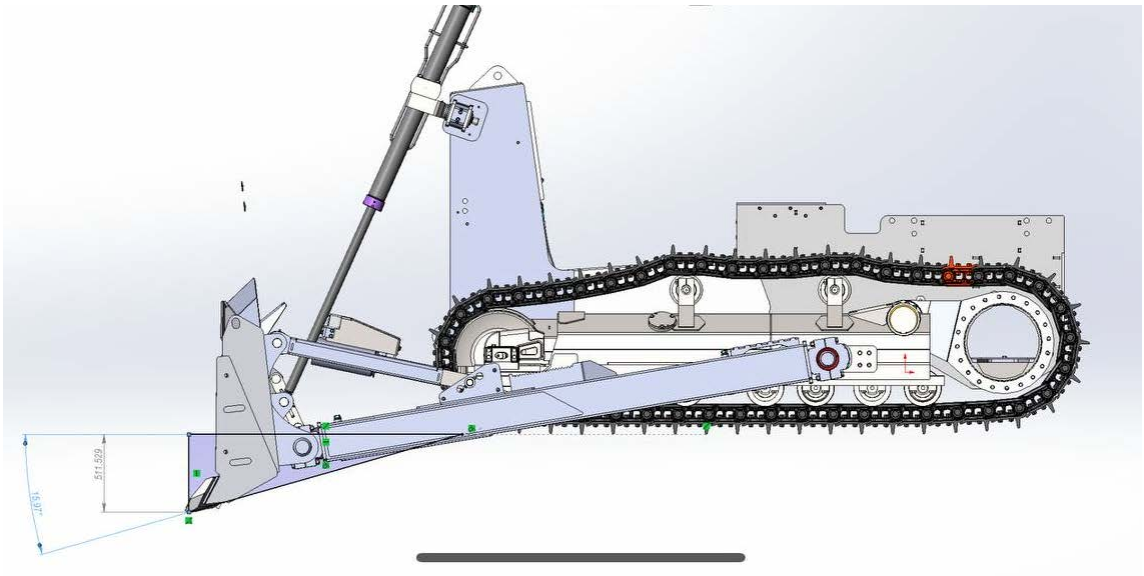

Рисунок 11 – Нижнее положение отвала

При разделении гидроцилиндра на поршень и цилиндр с привязкой к различным объектам, таким как отвал и корпус бульдозера, их движение становится взаимосвязанным. Например, при подъеме отвала происходит также движение поршня. Таким образом, для поворота гидроцилиндра и поршня состоит в определении угла поворота для поршня и гидроцилиндра для обеспечения совместного движения.

Для определения необходимого угла поворота используем встроенный метод "Vector3.Angle" [14] для нахождения угла между текущим и необходимым векторами цилиндра и поршня. Поскольку поворот происходит в отдельных

плоскостях, углы поворота необходимо определять отдельно для каждой плоскости. Это достигается путем проецирования векторов на выбранные плоскости с использованием метода "Vector3.ProjectOnPlane". В случае необходимости поворота гидроцилиндра по нескольким осям углы поворота определяются для каждой плоскости, после чего производится поворот по каждой оси в соответствии с найденными углами.

Реализация проецирования векторов на выбранные плоскости с использованием метода "Vector3.ProjectOnPlane" позволит точно определять углы поворота для поршня и цилиндра в каждой плоскости, обеспечивая согласованное и скоординированное движение при управлении гидроцилиндром.

Перемещать поршень нет необходимости, так как он связан с отвалом и перемещается вместе с ним. Поршень будет поворачиваться в нужную сторону.

Выводы по разделу два

В данной главе были описаны функциональные и нефункциональные требования к проектированию симуляции движения элементов бульдозера и модуля управления. Были выделены общие требования к устройству, связанные с приёмом сигналов, управлением, движением ковша и рыхлителя, а также выбором используемых библиотек.

Для работы с USB-устройствами будут использоваться встроенные возможности Unity. Для обработки сигналов от устройств, подключенных через последовательный порт, будет использоваться библиотека “System.IO.Ports”. Обработку входящих сигналов будем осуществлять с помощью реализованной инверсной кинематики, которая позволит изменять положение в зависимости от полученных данных.

В данной главе рассмотрели два шаблона проектирования "Цепочка Обязанностей" и "Команда", для модуля управления бульдозером наиболее подходит шаблон "Команда". Этот шаблон позволяет создавать объекты-команды для выполнения определенных действий, отделяя запросы от их выполнения и обеспечивая гибкую настройку управления.

Для управления движением отвала и рыхлителя будет использован поворот объекта по осям с ограничением углов поворота. Гидроцилиндр разделен на два объекта. Гидроцилиндр и поршень присоединены к разным объектам в бульдозере. Находится угол поворота и на этот угол поворачивается гидроцилиндр.

3. РЕАЛИЗАЦИЯ

Составим общую схему для модуля, обрабатывающего сигнал с управляющего устройства. На рисунке 12 представлена общая схема.

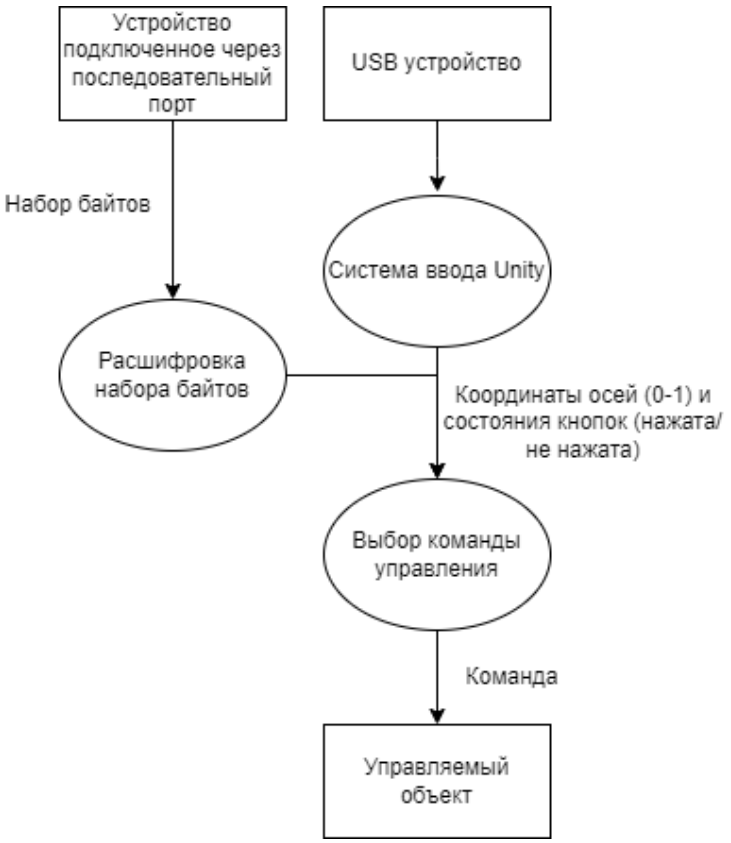

Рисунок 12 – Общая схема модуля, обрабатывающего сигнал с управляющего устройства

На основе схемы на рисунке 12 напишем программный код. Была создана система классов, обрабатывающая сигнал с управляющего устройства. На рисунке 13 описаны классы, связанные с выбором устройства ввода и настройкой управления.

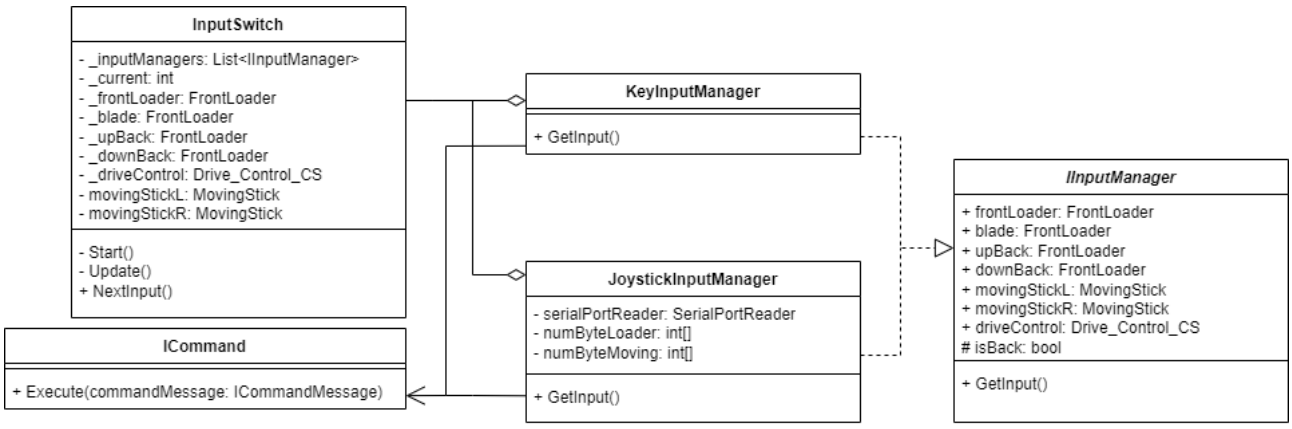

Рисунок 13 – Диаграмма классов

В листинге 1 приведен фрагмент кода класса InputSwitch. В методе Update, вызов которого происходит каждый кадр. Происходит вызов GetInput который получает значение с управляющих устройств. При нажатии кнопки изменяется переменная _current, то есть выбирается другой элемент массива, то есть другое устройство. В приложении А.1 написан полный код InputSwitch.

Листинг 1 – фрагмент InputSwitch

```
private void Update()
{
    _inputManagers[_current].GetInput();
    if(Input.GetKeyDown(KeyCode.RightBracket)) NextInput();
}
```

В классах KeyInputManager и JoystickInputManager описанно управление для конкретных устройств. В листинге 2 указан пример кнопки клавиатуры. При нажатии кнопки R произойдет вызов команды объекта FrontLoader. В команде передается значение оси и переменная выбора между отвалом и рыхлителем. В приложении А.2 написан полный код KeyInputManager.

Листинг 2 – фрагмент KeyInputManager

```
if (Input.GetKey(KeyCode.R))
{
    FrontLoader.Execute(new CommandMessageLoader(1, isBack));
    UpBack.Execute(new CommandMessageLoader(1, isBack));
    commandMessageJoysticksL._horizontal = 1;
}
```

В классе JoystickInputManager устроено аналогично, только значение берется из COM порта. В листинге 3 пример считывания оси джойстика. В приложении А.3 написан полный код JoystickInputManager. Классы KeyInputManager и JoystickInputManager наследуются от IInputManager код которого приведен в приложении А.4.

Листинг 3 – фрагмент JoystickInputManager

```
if (numByteLoader[0] > 0 && numByteLoader[0] != 63)
{
    FrontLoader.Execute(new CommandMessageLoader( – numByteLoader[0]
* 0.01f, isBack));
    UpBack.Execute(new CommandMessageLoader(–numByteLoader[0] *
0.01f, isBack));
}
```

На рисунке 14 описаны классы, связанные с командами.

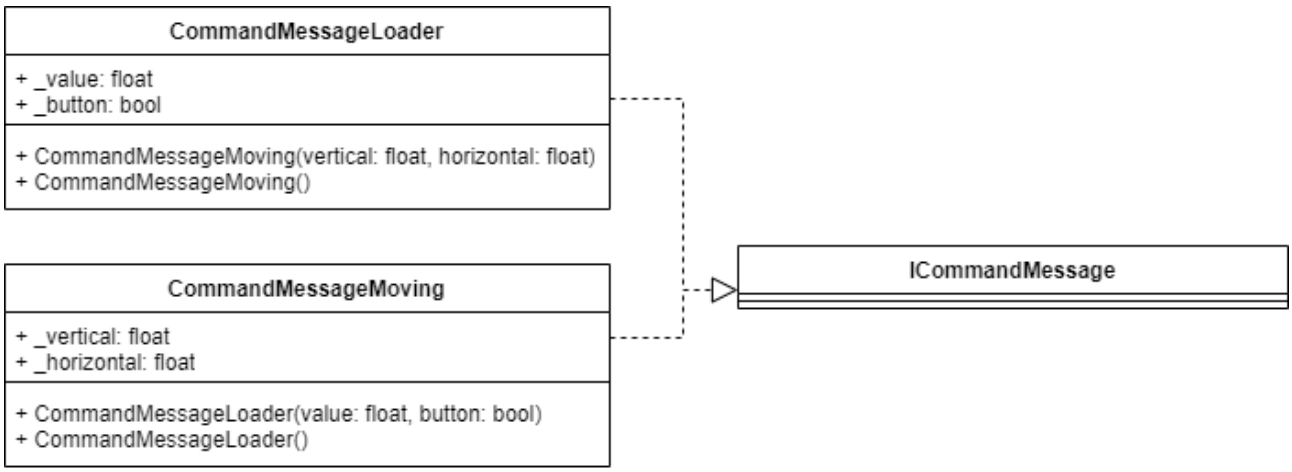

Рисунок 14 – Диаграмма классов

В ICommand передается отдельный класс, созданный для сообщений. В листинге 4 описан интерфейс ICommand. Создан интерфейс сообщений, от которого наследуются конкретные сообщения. Благодаря этому нам не надо создавать отдельные команды под каждое сообщение. В листинге 5 описан класс, хранящий информацию для модуля передвижения. В листинге 6 описан класс, хранящий информацию поворота объектов.

Листинг 4 – содержание класса ICommand

```
namespace Assets.PersonnelsFolder.LMS.Inputs.Managers
{
    public interface ICommand
    {
        public void Execute(ICommandMessage commandMessage);
    }
}
```

Листинг 5 – содержание класса CommandMessageMoving

```
public class CommandMessageMoving : ICommandMessage
{
    public CommandMessageMoving() { }
    public CommandMessageMoving(float vertical, float horizontal)
    {
        _vertical = vertical;
        _horizontal = horizontal;
    }
    public float _vertical { get; set; }
    public float _horizontal { get; set; }
}
```

Листинг 6 – содержание класса CommandMessageLoader

```
public class CommandMessageLoader : ICommandMessage
{
    public CommandMessageLoader() { }
    public CommandMessageLoader(float value, bool button)
    {
        _value = value;
        _button = button;
    }

    public float _value { get; set; }
    public bool _button { get; set; }
}
```

На рисунке 15 описаны классы, связанные с чтением информации из COM порта.

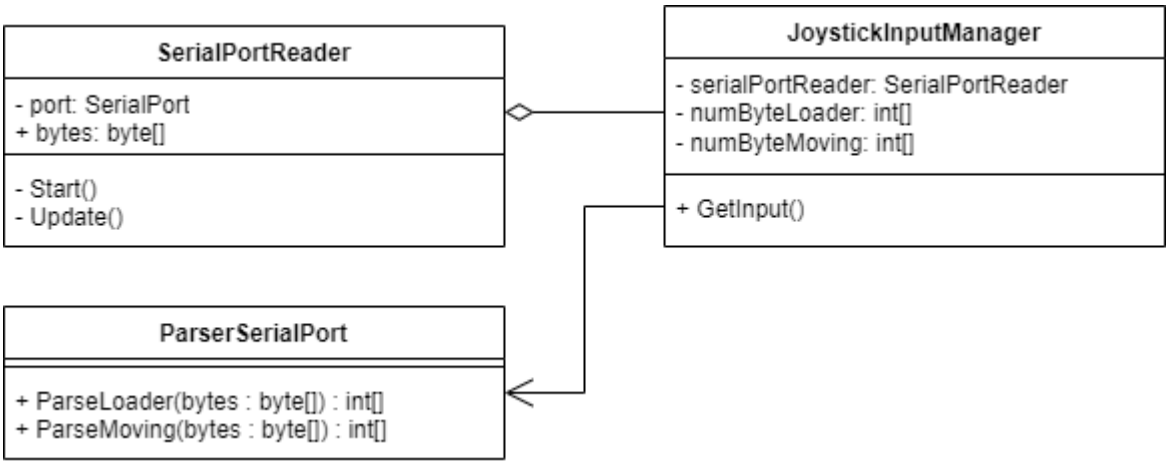

Рисунок 15 – Диаграмма классов

JoystickInputManager берет информацию из COM порта. Из SerialPortReader поступает массив байтов. В листинге 7 создается подключение COM порта. При создании необходимо указать нужные настройки. Далее каждый кадр будем получать информацию. В листинге 8 показано получение информации. В приложении А.5 написан полный код SerialPortReader.

Листинг 7 – фрагмент SerialPortReader

```
private static SerialPort port = new SerialPort("COM3",115200,
Parity.None, 8, StopBits.One);
```

Листинг 8 – фрагмент SerialPortReader

```
private void Update()
{
    if (port != null)
    {
        if (port.IsOpen)
        {
            if (port.BytesToRead > 37)
            {
                port.Read(bytes, 0, 39);
            }
        }
    }
}
```

Из массива байтов необходимо выбрать нужные. ParserSerialPort возвращает преобразованный массив чисел. В листинге 9 получение байтов, связанных с движением бульдозера. В приложении А.6 написан полный код ParserSerialPort.

Листинг 9 – фрагмент ParserSerialPort

```
public static int[] ParseMoving(byte[] bytes)
{
    int[] numByte = new int[4];
    for (int i = 0; i < bytes.Length; i++)
    {
        if (bytes[i] == 144 && bytes[i – 1] == 1)
        {
            numByte[0] = bytes[i + 1]; //byteLeft
            numByte[1] = bytes[i + 2]; //byteUp
        }
        if (bytes[i] == 144 && bytes[i – 1] == 2)
        {
            numByte[2] = bytes[i + 1]; //byteRight
            numByte[3] = bytes[i + 2]; //byteDown
        }
    }
    return numByte;
}
```

Управляемые объекты наследуются от класса ICommand. Поэтому мы можем их добавлять в классы KeyInputManager и JoystickInputManager. В управляемых объектах мы реализуем метод Execute. Метод Execute в управляемом объекте представляет собой ключевую точку взаимодействия с командой. Он вызывается из объекта команды и предоставляет место для последующей логики обработки. В листинге 10 описан метод Execute созданый в классе FrontLoader. В приложении А.7 написан полный код FrontLoader.

Листинг 10 – фрагмент FrontLoader

```
public void Execute(ICommandMessage commandMessage)
{
    CommandMessageLoader commandMessageLoader =
(CommandMessageLoader)commandMessage;
    if(isBack) speed = commandMessageLoader._value * maxSpeed;
    if(commandMessageLoader._button) isBack = !isBack;

    angle = transform.localEulerAngles.y;
    angle = Mathf.Repeat(angle + 180 + offset, 360) – 180;

    if (angle > maxAngle) speed = Mathf.Clamp(speed, –1000, 0);
    else if (angle < minAngle) speed = Mathf.Clamp(speed, 0, 1000);

    transform.localEulerAngles += new Vector3(0, speed *
Time.fixedDeltaTime, 0);
    speed = 0;
}
```

Через метод Execute в коде класса MovingStick происходит прием данных манипулятора, а затем поворот в необходимое положение виртуальных джойстиков. В приложении А.8 написан полный код MovingStick.

На рисунке 16 описаны классы, связанные с получение команд.

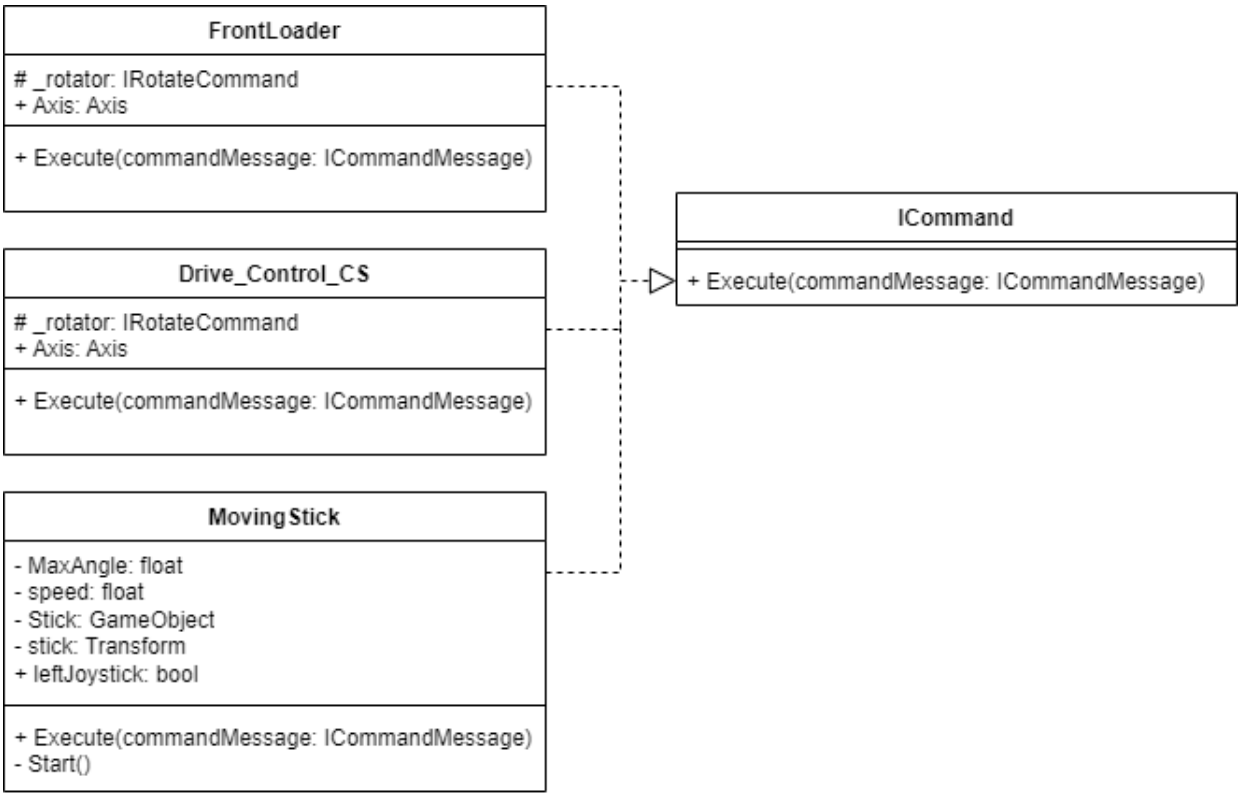

Рисунок 16 – Диаграмма классов

В листинге 11 находится векторы текущего положения и необходимого. Вектор находится по двум точкам. Поэтому заранее необходимо расставить точки в нужных местах.

Листинг 11 – фрагмент RotatePiston

```
Vector3 curVec = (endp.position –startp.position).normalized;
Vector3 needVec = (end.position – startp.position).normalized;
```

Далее в листинге 12 находится проекция векторов на определенную плоскость. Находится угол между найденными векторами.

Листинг 12 – фрагмент RotatePiston

```
curVec = Vector3.ProjectOnPlane(curVec, parent.right);
needVec = Vector3.ProjectOnPlane(needVec, parent.right);

curAngle = Vector3.Angle(needVec, parent.up);
needAngle = Vector3.Angle(curVec, parent.up);
```

После происходит поворот на найденный угол. В листинге 13 показан пример кода. В приложении А.9 написан полный код RotatePiston.

Листинг 13 – фрагмент RotatePiston

```
transform.RotateAround(startp.position, axisVec3, angle);
```

Выводы по разделу четыре

В данной главе были реализованы системы движение отвала и рыхлителя с возможностью настройки, и системы управления бульдозером.

4. ТЕСТИРОВАНИЕ РАБОТОСПОСОБНОСТИ ПРОГРАММНОГО МОДУЛЯ

Необходимо протестировать модуль для приема, обработки команд от манипулятора. В тесты должна входить проверка управления, то есть правильная реакция бульдозера на движения манипулятора. Проверка перемещение отвала и рыхлителя. Также проверка на соответствие требованиям. Проверка будет проходить методом функционального тестирования. Функциональное тестирование это — тестирование, которое игнорирует внутренний механизм системы или компонента и фокусируется исключительно на выходных данных, генерируемых в ответ на выбранные входные данные и условия выполнения. Тестирование, проводимое для оценки соответствия системы или компонента заданным функциональным требованиям [15]. Будут проверены поставленные требования к разрабатываемому модулю и возможные сценарии использования манипуляторов управления.

В таблице 2 представлены необходимые критерии и итоги тестирования модуля. В таблице 3 приведено соответствие требований заказчика выполненной работе.

Таблица 2 – Описание тестов и результатов тестов проекта

Название теста	Ожидаемый результат	Полученный результат	Результат
Оператор наклонил правый джойстик вправо	Бульдозер начал разворот по часовой стрелке	Бульдозер развернулся по часовой стрелке	Тест пройден
Оператор наклонил правый джойстик влево	Бульдозер начал разворот против часовой стрелки	Бульдозер развернулся против часовой стрелки	Тест пройден
Оператор наклонил правый джойстик вперед	Бульдозер начал движение вперёд	Бульдозер двигается вперёд	Тест пройден

Продолжение таблицы 2

Название теста	Ожидаемый результат	Полученный результат	Результат
Оператор наклонил правый джойстик назад	Бульдозер начал движение назад	Бульдозер двигается назад	Тест пройден
Оператор нажал кнопку переключения режима отвал/рыхлитель	Бульдозер переключил режим на противоположный	Бульдозер переключил режим на противоположный	Тест пройден
Оператор наклонил левый джойстик вправо	Отвал двигается вверх	Отвал двигается вверх	Тест пройден
Оператор наклонил левый джойстик влево	Отвал двигается вниз	Отвал двигается вниз	Тест пройден
Оператор наклонил левый джойстик вперед	Отвал наклонился вперед	Отвал наклонился вперед	Тест пройден
Оператор наклонил левый джойстик назад	Отвал наклонился назад	Отвал наклонился назад	Тест пройден
Оператор нажал кнопку переключения отвал/рыхлитель	Управление переключилось с отвала на рыхлитель	Управление переключилось с отвала на рыхлитель	Тест пройден

Таблица 3 – таблица выполненных требований

Требования	Реализация
Движение ковша, отвала и рыхлителя с возможностью настройки.	Реализован алгоритм движение отвала и рыхлителя с возможностью настройки ограничителя.
Возможность выбора из нескольких устройств ввода.	Реализовано управление с помощью клавиатуры и джойстика.

На рисунке 17 показано перемещение отвала.

Рисунок 17 – Перемещение отвала

На рисунке 18 показано перемещение рыхлителя.

Рисунок 18 – Перемещение рыхлителя

Выводы по разделу четыре

По результатам тестов и сравнением с требованиями требованиям, программный модуль полностью прошел тестирование и выполнил требования.

ЗАКЛЮЧЕНИЕ

В результате исследований был создан программный модуль для приема и обработки команд от ручного манипулятора бульдозера. Прототип успешно прошел все тесты и готов к использованию. Для достижения поставленных целей были выполнены следующие этапы исследования:

1. Проведен аналитический обзор существующей литературы и методов, а также проанализированы аналоги на рынке.
2. Спроектирован программный модуль и выбраны необходимые компоненты.
3. Реализован программный модуль для приема и обработки команд от ручного манипулятора бульдозера.
4. Реализована программная часть модуля, после чего проведено тестирование программного модуля для приема и обработки команд от ручного манипулятора бульдозера. Результаты показали, что все задачи выполнены, и прототип работает корректно.

Таким образом, в данной работе были выполнены все требования и реализованы все задачи. Данное исследование может быть использовано для дальнейшей научной разработке в рамках данной предметной области.

В дальнейшем для данного прототипа планируется улучшение функционала.

БИБЛИОГРАФИЧЕСКИЙ СПИСОК

1. Учебные тренажеры и симуляторы [Электронный ресурс] : Simgenix. – URL: https://www.simgenix.com/ (дата обращения: 20.12.2023).
2. CM Labs Simulation Software & Heavy Equipment Simulation [Электронный ресурс] : CM Labs. – URL: https://www.cm–labs.com/en/ (дата обращения: 20.12.2023).
3. Учебный тренажер–симулятор бульдозера [Электронный ресурс] : ЧЕТРА. – URL: https://service–im.ru/training/oborudovanie–dlya–obucheniya.php (дата обращения: 20.12.2023).
4. Input Manager [Электронный ресурс] : Unity User Manual. – URL: https://docs.unity3d.com/Manual/class-InputManager.html (дата обращения: 20.12.2023).
5. Serial Programming Guide for POSIX Operating Systems [Электронный ресурс] : Федеральное государственное бюджетное учреждение науки институт теоретической и прикладной механики им. С.А. Христиановича сибирского отделения российской академии наук. – URL: https://web.archive.org/web/20100221061304/http://linuxland.itam.nsc.ru/misc/other19/index.html#CONTENTS (дата обращения: 20.12.2023).
6. System.IO.Ports [Электронный ресурс] : Техническая документация Microsoft. – URL: https://learn.microsoft.com/ru–ru/dotnet/api/system.io.ports?view=dotnet–plat–ext–8.0 (дата обращения: 20.12.2023).
7. Animation [Электронный ресурс] : Unity User Manual. – URL: https://docs.unity3d.com/Manual/AnimationSection.html (дата обращения: 20.12.2023).
8. Inverse Kinematics: a review of existing techniques and introduction of a new fast iterative solver / Andreas Aristidou, Joan Lasenby // University of Cambridge – 2009. − С. 1–3
9. Rigidbody [Электронный ресурс] : Unity User Manual. – URL: https://docs.unity3d.com/ru/2019.4/Manual/class–Rigidbody.html (дата обращения: 28.02.2023).

10. Joints [Электронный ресурс] : Unity User Manual. – URL: https://docs.unity3d.com/Manual/joints–section.html (дата обращения: 20.12.2023).
11. Input Manager [Электронный ресурс] : Unity User Manual. – URL: https://docs.unity3d.com/Manual/class-InputManager.html (дата обращения: 20.12.2023).
12. Гамма Э., Хелм Р., Джонсон Р., Влиссидес Дж. Приемы объектно–ориентированного проектирования. Паттерны проектирования. — СПб.: Питер, 2015. — 368 с
13. Command [Электронный ресурс] : Game Programming Patterns. – URL: https://gameprogrammingpatterns.com/command.html (дата обращения: 20.12.2023).
14. Vector3 [Электронный ресурс] : Unity User Manual. – URL: https://docs.unity3d.com/ScriptReference/Vector3.html (дата обращения: 23.03.2024).
15. ISO/IEC/IEEE International Standard – Systems and software engineering. ISO/IEC/IEEE 24765:2010(E), 2010. – 410 с.

ПРИЛОЖЕНИЕ

ПРИЛОЖЕНИЕ А. КОД РАЗРАБОТАННОГО ПРОГРАМНОГО МОДУЛЯ

Листинг А.1 – исходный код скрипта InputSwitch

```
using System.Collections.Generic;
using ChobiAssets.PTM;
using D12;
using UnityEngine;

namespace Assets.PersonnelsFolder.LMS.Inputs.Managers
{
    /// <summary>
    /// Код для хранения и переключения контроллеров управления
    /// </summary>
    public class InputSwitch : MonoBehaviour
    {
        [SerializeField] private List<IInputManager> _inputManagers =
new List<IInputManager>(); //Список контроллеров
        private int _current = 0; //Текущий контроллер управления

        //Переменные для управляемых объектов
        [SerializeField] private FrontLoader _frontLoader; //Отвал
        [SerializeField] private FrontLoader _blade; //Отвал
        [SerializeField] private FrontLoader _upBack; //Рыхлитель
        [SerializeField] private FrontLoader _downBack; //Рыхлитель
        [SerializeField] private MovingStick _movingStickL; //Левый
виртуальный джойстик
        [SerializeField] private MovingStick _movingStickR; //Правый
виртуальный джойстик

        [SerializeField] private Drive_Control_CS _driveControl;
//Программный код отвечающий за перемещение бульдозера

        private void Start()
        {
            //Присваивание всех объектов всем контроллерам управления
            foreach (IInputManager inputManager in _inputManagers)
            {
                inputManager.FrontLoader = _frontLoader;
                inputManager.Blade = _blade;
                inputManager.UpBack = _upBack;
                inputManager.DownBack = _downBack;
                inputManager.movingStickL = _movingStickL;
                inputManager.movingStickR = _movingStickR;
                inputManager.DriveControl = _driveControl;
            }
        }

        private void Update()
        {
            //Получение значений манипулятора происходит каждый кадр
```

Продолжение листинга А.1

```
            _inputManagers[_current].GetInput();

            if(Input.GetKeyDown(KeyCode.RightBracket)) NextInput();
        }

        /// <summary>
        /// Переключить контроллер на следующий
        /// </summary>
        public void NextInput()
        {
            _current++;
            if (_current >= _inputManagers.Count) _current = 0;
        }
    }
}
```

Листинг А.2 – исходный код скрипта KeyInputManager

```
using ChobiAssets.PTM;
using D12;
using UnityEngine;

namespace Assets.PersonnelsFolder.LMS.Inputs.Managers
{
    /// <summary>
    /// Код с настроенным управлением для клавиатуры
    /// </summary>
    public class KeyInputManager : IInputManager
    {
        public override void GetInput()
        {
            //Переключение между отвалом и рыхлителем
            if (Input.GetKeyDown(KeyCode.Q))
            {
                isBack = true;
            }
            else
            {
                isBack = false;
            }

            //Создание команды для виртуального джойстка
            CommandMessageMoving commandMessageJoysticksL = new
CommandMessageMoving(0, 0);

            //Отправка комманд взависимости от нажатой клавиши
            if (Input.GetKey(KeyCode.R))
            {
```

Продолжение листинга А.2

```
                FrontLoader.Execute(new      CommandMessageLoader(1,
isBack));
                UpBack.Execute(new CommandMessageLoader(1, isBack));
                commandMessageJoysticksL._horizontal = 1;
            }
            else if (Input.GetKey(KeyCode.F))
            {
                FrontLoader.Execute(new     CommandMessageLoader(-1,
isBack));
                UpBack.Execute(new CommandMessageLoader(-1, isBack));
                commandMessageJoysticksL._horizontal = -1;
            }
            else
            {
                UpBack.Execute(new CommandMessageLoader(0, isBack));
                FrontLoader.Execute(new      CommandMessageLoader(0,
isBack));
            }

            if (Input.GetKey(KeyCode.T))
            {
                Blade.Execute(new CommandMessageLoader(1, isBack));
                DownBack.Execute(new         CommandMessageLoader(1,
isBack));
                commandMessageJoysticksL._vertical = 1;
            }
            else if (Input.GetKey(KeyCode.G))
            {
                Blade.Execute(new CommandMessageLoader(-1, isBack));
                DownBack.Execute(new        CommandMessageLoader(-1,
isBack));
                commandMessageJoysticksL._vertical = -1;
            }
            else
            {
                DownBack.Execute(new         CommandMessageLoader(0,
isBack));
                Blade.Execute(new CommandMessageLoader(0, isBack));
            }

            movingStickL.Execute(commandMessageJoysticksL);

            CommandMessageMoving   commandMessageJoysticksR   =   new
CommandMessageMoving(0, 0);
            float vertical = 0;

            //Присвоение переменных взависимости от нажатой клавиши
            if (Input.GetKey(General_Settings_CS.Drive_Up_Key))
            {
```

Продолжение листинга А.2

```
                vertical = 1.0f;
                commandMessageJoysticksR._vertical = 1.0f;
            }
            else                                                    if
(Input.GetKey(General_Settings_CS.Drive_Down_Key))
            {
                vertical = -0.5f;
                commandMessageJoysticksR._vertical = -0.5f;
            }
            else
            {
                vertical = 0.0f;
            }

            float horizontal = 0;

            //Присвоение переменных взависимости от нажатой клавиши
            if (Input.GetKey(General_Settings_CS.Drive_Left_Key))
            {
                horizontal = -1.0f;
                commandMessageJoysticksR._horizontal = -1.0f;
            }
            else                                                    if
(Input.GetKey(General_Settings_CS.Drive_Right_Key))
            {
                horizontal = 1.0f;
                commandMessageJoysticksR._horizontal = 1.0f;
            }
            else
            {
                horizontal = 0.0f;
            }

            //Отправка комманд с переменными присвоеными ранее
            DriveControl.Execute(new   CommandMessageMoving(vertical,
horizontal));
            movingStickR.Execute(commandMessageJoysticksR);
        }
    }
}
```

Листинг А.3 – исходный код скрипта JoystickInputManager

```
using ChobiAssets.PTM;
using D12;
using System;
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
```

Продолжение листинга А.3

```
namespace Assets.PersonnelsFolder.LMS.Inputs.Managers
{
    /// <summary>
    /// Код с настроенным управлением для джойстиков
    /// </summary>
    public class JoystickInputManager : IInputManager
    {
        private int[] numByteLoader = new int[5]; //Массив байтов для управления элементами бульдозера
        private int[] numByteMoving = new int[4]; //Массив байтов для управления движением бульдозера

        private bool isPress = false; //Переменнная для того чтобы убрать дребезг кнопки

        [SerializeField] private SerialPortReader SerialPortReader; //Класс для чтения данных из последовательного порта

        public override void GetInput()
        {
            //Получения байтов для управления элементами бульдозера
            numByteLoader = ParserSerialPort.ParseLoader(SerialPortReader.bytes);

            //Переключение между отвалом и рыхлителем
            if (numByteLoader[4] == 0 && !isPress)
            {
                if (isBack) isBack = false;
                else isBack = true;

                isPress = true;
            }
            else if (numByteLoader[4] == 1 && isPress)
            {
                isPress = false;
            }

            //Отправка комманд взависимости от нажатой клавиши
            if (numByteLoader[0] > 0 && numByteLoader[0] != 63)
            {
                FrontLoader.Execute(new CommandMessageLoader(-numByteLoader[0] * 0.01f, isBack));
                UpBack.Execute(new CommandMessageLoader(-numByteLoader[0] * 0.01f, isBack));
            }
            else if (numByteLoader[1] > 0)
            {
                FrontLoader.Execute(new CommandMessageLoader(numByteLoader[1] * 0.01f, isBack));
```

Продолжение листинга А.3

```
                UpBack.Execute(new
CommandMessageLoader(numByteLoader[1] * 0.01f, isBack));
            }
            else
            {
                FrontLoader.Execute(new      CommandMessageLoader(0,
isBack));
                UpBack.Execute(new CommandMessageLoader(0, isBack));
            }

            if (numByteLoader[3] > 0 && numByteLoader[3] != 63)
            {
                Blade.Execute(new             CommandMessageLoader(-
numByteLoader[3] * 0.01f, isBack));
                DownBack.Execute(new          CommandMessageLoader(-
numByteLoader[3] * 0.01f, isBack));
            }
            else if (numByteLoader[2] > 0)
            {
                Blade.Execute(new
CommandMessageLoader(numByteLoader[2] * 0.01f, isBack));
                DownBack.Execute(new
CommandMessageLoader(numByteLoader[2] * 0.01f, isBack));
            }
            else
            {
                Blade.Execute(new CommandMessageLoader(0, isBack));
                DownBack.Execute(new         CommandMessageLoader(0,
isBack));
            }

            numByteMoving                                          =
ParserSerialPort.ParseMoving(SerialPortReader.bytes);

            float vertical = 0;
            if (numByteMoving[1] > 0 && numByteMoving[1] <= 100)
                vertical = numByteMoving[1] * 0.01f;
            else vertical = -numByteMoving[3] * 0.01f;

            float horizontal = 0;
            if (numByteMoving[0] > 0 && numByteMoving[0] <= 100)
                horizontal = -numByteMoving[0] * 0.01f;
            else horizontal = numByteMoving[2] * 0.01f;

            DriveControl.Execute(new   CommandMessageMoving(vertical,
horizontal));
        }
    }
}
```

Листинг А.4 – исходный код скрипта IInputManager

```
using D12;
using System.Collections;
using System.Collections.Generic;
using ChobiAssets.PTM;
using UnityEngine;

public abstract class IInputManager : MonoBehaviour
{
    //Переменные для управляемых объектов
    public FrontLoader FrontLoader { get; set; } //Отвал
    public FrontLoader Blade { get; set; } //Отвал
    public FrontLoader UpBack { get; set; } //Рыхлитель
    public FrontLoader DownBack { get; set; } //Рыхлитель
    public MovingStick movingStickL { get; set; } //Левый
виртуальный джойстик
    public MovingStick movingStickR { get; set; } //Правый
виртуальный джойстик
    protected bool isBack = false; //Переманная выбора между отвалом
и рыхлителем

    public Drive_Control_CS DriveControl { get; set; } //Програмный
код отвечающий за перемещение бульдозера

    public abstract void GetInput(); //Отправить комманды
}
```

Листинг А.5 – исходный код скрипта SerialPortReader

```
using UnityEngine;
using System;
using System.IO.Ports;
using System.Text;

namespace D12
{
    /// <summary>
     /// Чтение последовательного порта
     /// </summary>
    public class SerialPortReader : MonoBehaviour
    {
        private static SerialPort port = new SerialPort("COM3",
        115200, Parity.None, 8, StopBits.One);

        public byte[] bytes = new byte[39];

        [STAThread]
        void Start()
        {
            //Настройка последовательного порта
            port.Handshake = Handshake.None;
            port.DtrEnable = true;
```

Продолжение листинга А.5

```
            port.ReadTimeout = 500;
            port.WriteTimeout = 1000;
            port.Parity = Parity.None;
            port.StopBits = StopBits.One;
            port.DataBits = 8;
            port.NewLine = "\n";

            port.Open();
            Console.ReadLine();
        }

        //Чтение посоледовательного порта каждый кадр
        private void Update()
        {
            if (port != null)
            {
                if (port.IsOpen)
                {

                    if (port.BytesToRead > 37)
                    {
                        port.Read(bytes, 0, 39);
                    }
                }
            }
        }
    }
}
```

Листинг А.6 – исходный код скрипта ParserSerialPort

```
using D12;
using System;

namespace Assets.PersonnelsFolder.LMS.Inputs.Managers
{
    /// <summary>
    /// Поиск нужных байт из последовательного порта
    /// </summary>
    public static class ParserSerialPort
    {
        public static int[] ParseLoader(byte[] bytes)
        {
            int[] numByte = new int[5];

            for (int i = 0; i < bytes.Length; i++)
            {
                if (bytes[i] == 128 && bytes[i - 1] == 1)
                {
```

Продолжение листинга А.6

```
                    numByte[0] = bytes[i + 1]; // Ось вверх
                    numByte[1] = bytes[i + 2]; // Ось вниз
                    numByte[2] = bytes[i + 3]; // Ось влево
                    numByte[3] = bytes[i + 4]; // Ось вправо
                    numByte[4] = bytes[i + 6]; // Кнопка
                }
            }

            return numByte;
        }

        public static int[] ParseMoving(byte[] bytes)
        {
            int[] numByte = new int[4];
            for (int i = 0; i < bytes.Length; i++)
            {
                if (bytes[i] == 144 && bytes[i - 1] == 1)
                {
                    numByte[0] = bytes[i + 1]; //Ось влево
                    numByte[1] = bytes[i + 2]; //Ось вверх
                }
                if (bytes[i] == 144 && bytes[i - 1] == 2)
                {
                    numByte[2] = bytes[i + 1]; //Ось вправо
                    numByte[3] = bytes[i + 2]; //Ось вниз
                }
            }

            return numByte;
        }
    }
}
```

Листинг А.7 – исходный код скрипта FrontLoader

```
using Assets.PersonnelsFolder.LMS.Inputs.Managers;
using ChobiAssets.PTM;
using UnityEngine;

namespace D12
{
    /// <summary>
    /// Код для поворота отвала и рыхлителя
    /// </summary>
    public class FrontLoader : MonoBehaviour, ICommand
    {
        private float speed = 0; //Текущая скорость поворота
        [SerializeField] private float maxSpeed = 10f; //Максимальная
скорость поворота
```

Продолжение листинга А.7

```
        [SerializeField]  private  Transform  pointRotate;  //Точка
поворота объекта

        [SerializeField] private bool invert;
        [SerializeField]  float  maxAngle  =  0;  //Максимальный угол
поворота
        [SerializeField]  float  minAngle  =  0;  //Минимальный угол
поворота
        [SerializeField] float offset = 0; //Смещение угла

        [SerializeField] private float angle; //Текущий угол поворота
        public bool isBack = false; //Выбор отвала или рыхлителя

        void FixedUpdate()
        {

        }

        /// <summary>
        /// Поворот отвала и рыхлителя
        /// </summary>
        public void Execute(ICommandMessage commandMessage)
        {
            CommandMessageLoader        commandMessageLoader        =
(CommandMessageLoader)commandMessage;
            if(isBack) speed = commandMessageLoader._value * maxSpeed;
            if(commandMessageLoader._button) isBack = !isBack;

            angle = transform.localEulerAngles.y;
            angle = Mathf.Repeat(angle + 180 + offset, 360) - 180;

            if (angle > maxAngle) speed = Mathf.Clamp(speed, -1000,
0);
            else if (angle < minAngle) speed = Mathf.Clamp(speed, 0,
1000);

            transform.localEulerAngles  +=  new  Vector3(0,  speed  *
Time.fixedDeltaTime, 0);

            speed = 0;
        }
    }
}
```

Листинг А.8 – исходный код скрипта MovingStick

```
using Assets.PersonnelsFolder.LMS.Inputs.Managers;
using ChobiAssets.PTM;
using UnityEngine;
```

Продолжение листинга А.8

```
namespace D12
{
    /// <summary>
    /// Код для поворота виртуального джойстика
    /// </summary>
    public class MovingStick : MonoBehaviour, ICommand
    {
        [SerializeField] float MaxAngle = 30f; //Максимальный угол поворта виртуального джойстика
        [SerializeField] float speed = 1;
        [SerializeField] GameObject Stick; //Ссылка на объект
        private Transform stick; //Ссылка на параметры объекта

        public bool leftJoystick = false; //Выбор левого или правого джойстика

        public void Start()
        {
            stick = Stick.GetComponent<Transform>();
        }

        /// <summary>
        /// Поворот виртуального джойстика
        /// </summary>
        public void Execute(ICommandMessage commandMessage)
        {
            CommandMessageMoving commandMessageMoving = (CommandMessageMoving)commandMessage;

            float ver = -commandMessageMoving._vertical; float hor = commandMessageMoving._horizontal;

            float angleX = hor * MaxAngle;
            float angleZ = ver * MaxAngle;

            if (leftJoystick) stick.localRotation = Quaternion.Euler(angleX, 0, -angleZ);
            else stick.localRotation = Quaternion.Euler(angleX, -angleZ, 0);
        }
    }
}
```

Листинг А.9 – исходный код скрипта RotatePiston

```
using System;
using UnityEngine;
```

Продолжение листинга А.9

```
namespace D12
{
    /// <summary>
    /// Код для поворта гидроцилидров
    /// </summary>
    public class RotatePiston : MonoBehaviour
    {
        [SerializeField] private Transform startp; //Точка начала
вектора
        [SerializeField] private Transform endp; //Конечная точка
вектора
        [SerializeField] private Transform end; //Точка к которой
стремится вектор

        [SerializeField] private Transform parent; //Ссылка на
родительский объект

        [SerializeField] private Axis axis; //Ось поворота
        [SerializeField] private bool isInvert; //Инвертировать
поворот
        private enum Axis
        {
            right,
            forward,
            up,
            oldRight
        }

        [SerializeField] private float angle = 0; //Текуций угол
поворота

        //Поворот объекта по определенной оси к точке каждый кадр
        void Update()
        {
            Vector3 axisVec3 = Vector3.zero;
            if (axis == Axis.right) axisVec3 = startp.right;
            else if (axis == Axis.forward || axis == Axis.oldRight)
axisVec3 = startp.forward;
            else axisVec3 = startp.up;

            Vector3 curVec = (endp.position -
startp.position).normalized;
            Vector3 needVec = (end.position -
startp.position).normalized;

            float curAngle = 0;
            float needAngle = 0;
```

Продолжение листинга А.9

```
            switch (axis)
            {
                case Axis.right:
                    curVec = Vector3.ProjectOnPlane(curVec,
parent.right);
                    needVec = Vector3.ProjectOnPlane(needVec,
parent.right);

                    curAngle = Vector3.Angle(needVec, parent.up);
                    needAngle = Vector3.Angle(curVec, parent.up);
                    break;
                case Axis.up:
                    curVec = Vector3.ProjectOnPlane(curVec,
parent.up);
                    needVec = Vector3.ProjectOnPlane(needVec,
parent.up);

                    curAngle = Vector3.Angle(needVec, parent.right);
                    needAngle = Vector3.Angle(curVec, parent.right);
                    break;
                case Axis.forward:
                    curVec = Vector3.ProjectOnPlane(curVec,
parent.forward);
                    needVec = Vector3.ProjectOnPlane(needVec,
parent.forward);

                    curAngle = Vector3.Angle(needVec, parent.up);
                    needAngle = Vector3.Angle(curVec, parent.up);
                    break;
                case Axis.oldRight:
                    curAngle = Vector3.Angle(needVec, parent.right);
                    needAngle = Vector3.Angle(curVec, parent.right);
                    break;
                default:
                    throw new ArgumentOutOfRangeException();
            }

            angle = 0;
            if (!isInvert) angle = -curAngle + needAngle;
            else angle = curAngle - needAngle;

            transform.RotateAround(startp.position, axisVec3,
angle);
        }
    }
}
```