МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

«МОСКОВСКИЙ ПОЛИТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

(МОСКОВСКИЙ ПОЛИТЕХ)

**«РАЗРАБОТКА ПОЛЬЗОВАТЕЛЬСКОГО ИНТЕРФЕЙСА «ТЕПЛОВОЙ УЗЕЛ» С ИСПОЛЬЗОВАНИЕМ UNREAL ENGINE 5»**

Курсовой проект по дисциплине «Инженерное проектирование»

по направлению 09.03.01 Информатика и вычислительная техника

Образовательная программа (профиль)

«Интеграция и программирование в САПР»

Студент: / Муравьёв А. Г., 191-324 /

*подпись ФИО, группа*

Преподаватель: / Толстиков А.В., к.т.н. /

*подпись ФИО, уч. звание и степень*

Москва, 2022

**МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ**

**РОССИЙСКОЙ ФЕДЕРАЦИИ**

Федеральное государственное автономное образовательное учреждение

высшего образования  
«МОСКОВСКИЙ ПОЛИТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

ЗАДАНИЕ НА КУРСОВОЙ ПРОЕКТ

по дисциплине «Инженерное проектирование» 7 семестр

по направлению 09.03.01 Информатика и вычислительная техника

Образовательная программа (профиль)

«Интеграция и программирование в САПР»

Кафедра «СМАРТ технологии»

Муравьёв Андрей Геннадьевич

191-324

|  |  |
| --- | --- |
| **ТЕМА КУРСОВОГО ПРОЕКТА** | Разработка элементов управления интерактивного тренажера "Тепловой узел" с использованием Unreal Engine 5 |
| **ПРАКТИЧЕСКИЙ РЕЗУЛЬТАТ** | |
| Назначение | Виджет интерфейса для управления приложением и взаимодействия с объектом на сцене |

|  |  |
| --- | --- |
| Основные функции | 1. Интерфейс главного меню для запуска свободной сцены, сцены с заданиями для обучения, настроек разрешения экрана приложения и выход из приложения 2. Интерфейс выбора заданий, содержащий 4 задания, параметры которого меняются с каждым перезапуском 3. Интерфейс управления объектом на сцене, отвечающий за включения и выключения устройства, все необходимые кнопки взаимодействия, лампочки демонстрирующие корректную работу устройства и дисплеи для отображения параметров компонентов устройства |
| Используемые технологии и платформы | Unreal Engine 5, Blueprints, C++ |
| **ВЫПОЛНЕНИЕ РАБОТЫ** | |
| Решаемые задачи | Разработка интерфейса приложения для взаимодействия с устройством. |
| Состав технической документации | Пояснительная записка к курсовому проекту |
| Состав графической части | Презентация |

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Задача | Неделя | | | | | | | | | | | | | | | | | |
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 |
| Анализ проблемы, поиск аналогов, выбор методов решения задачи |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Изучение документации, технической литературы |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| Создание и базовая настройка проекта |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| Создание подсистемы контроллера |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| Создание базового и управляемого персонажа |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| Настройка параметров мира, настройка параметров управления и ввода |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| Создание пользовательского интерфейса |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |

Преподаватель: / /

*подпись ФИО, уч. звание и степень*

Студент:  / Муравьёв А.Г., 191-324 /

**АННОТАЦИЯ**

Разработал студент Муравьёв А.Г.

Тема курсового проекта – «Разработка элементов управления интерактивного тренажера "Тепловой узел" с использованием Unreal Engine 5».

Кафедра «СМАРТ технологии», группа 191-324.

Работа представлена на 45 страницах, количество приложений – 1.

В результате проделанной работы был создан пользовательский интерфейс для взаимодействия пользователя с программой и установками теплового узла, реализована подсистема контроллера, отвечающая за входную информацию и вызывающая методы ее обработки, а также настройка персонажа и связь с подсистемами контроллера и интерфейса.

Практическая ценность полученных результатов заключается в визуальном представлении механизма теплового узла и его составных частей, возможность установить полупрозрачность компонентов для возможности наблюдения потока воды, возможность изучить название, назначение и принципы работы основных компонентов установок.

**ОГЛАВЛЕНИЕ**

[ВВЕДЕНИЕ 8](#_Toc125547563)

[1 АНАЛИЗ ПРОБЛЕМЫ 9](#_Toc125547564)

[1.1 Обзор литературы по теме курсового проекта 9](#_Toc125547565)

[1.2 Постановка задачи 9](#_Toc125547566)

[1.2.1 Цель разработки 9](#_Toc125547567)

[1.2.2 Выполняемые программой функции 9](#_Toc125547568)

[1.2.3 Решаемые задачи 10](#_Toc125547569)

[1.2.4 Описание 3D-модели теплового узла 10](#_Toc125547570)

[1.3 Метод решения задач 12](#_Toc125547571)

[1.4 Описание программных и технических средств для разработки приложения 12](#_Toc125547572)

[1.5 Входная информация 12](#_Toc125547573)

[1.6 Выходная информация 13](#_Toc125547574)

[2 ПРОГРАММНАЯ РЕАЛИЗАЦИЯ ПОСТАВЛЕННОЙ ЗАДАЧИ 14](#_Toc125547575)

[2.1 Создание проекта 14](#_Toc125547576)

[2.2 Создание управляемого персонажа 15](#_Toc125547577)

[2.3 Подсистема контроллера 21](#_Toc125547578)

[2.4 Создание пользовательского интерфейса 30](#_Toc125547579)

[3 ОПИСАНИЕ ПРОГРАММЫ 40](#_Toc125547580)

[3.1 Структура программы 40](#_Toc125547581)

[3.2 Сведения об операционной системе 41](#_Toc125547582)

[4 ЗАКЛЮЧЕНИЕ 42](#_Toc125547583)

[БИБЛИОГРАФИЧЕСКИЙ СПИСОК 43](#_Toc125547584)

[СПИСОК СОКРАЩЕНИЙ 45](#_Toc125547585)

[ПРИЛОЖЕНИЯ 46](#_Toc125547586)

[*Приложение А* 46](#_Toc125547587)

# ВВЕДЕНИЕ

В настоящее время большую популярность набирают интерактивные тренажеры по изучению механизмов. Анализируя рынок тренажеров по работе с тепловым узлом, было обнаружено, что среди программ отсутствуют интерактивные тренажеры, позволяющие изучить строение теплового узла, описание и демонстрацию работы его компонентов. Преимуществами данного тренажера являются:

* Сокращение временных затрат специалистов на обучение новых сотрудников.
* Исключение затрат на создание тренировочного стенда.
* Возможность работы на любом совместимом ПК

Функционал программы заключается в использовании управляемого контроллером персонажа для взаимодействия с интерфейсом и с 3D-моделью установки теплового узла. Возможность использовать интерактивный пульт управления для взаимодействия с моделью теплового узла, отслеживанием изменение его состояния и управления его компонентами.

# АНАЛИЗ ПРОБЛЕМЫ

## **Обзор литературы по теме курсового проекта**

В процессе изучения проблемы были рассмотрены такие источники, как:

* документация Unreal Engine 5;
* теоретические сведения о системах промышленной автоматизации;
* сведения о принципах работы и функционирования тепловых узлов.

В процессе поиска и анализа аналогов был изучен рынок интерактивных тренажеров. Было выявлено, что на рынке отсутствуют тренажеры по работе с тепловым узлом, позволяющие изучить его строение, составные компоненты и их работу. В предлагаемых решениях рассматривается проектирование системы без описания компонентов и не демонстрирующую их работу.

Для достижения необходимых требований необходимо создать пользовательский интерфейс, в котором будут отображаться кнопки управления и текст, который показывает значения таких параметров как: входная температура, выходная температура, а также подсистему контроллера для управления персонажем и взаимодействием с пользовательским интерфейсом.

## **Постановка задачи**

### Цель разработки

Разработка пользовательского интерфейса программы для взаимодействия пользователя с 3D-моделью установки, реализация персонажа и его контроллера.

### Выполняемые программой функции

Разрабатываемое программное обеспечение должно выполнять следующие функции:

* Интерфейс главного меню для запуска свободной сцены, сцены с заданиями для обучения, настроек разрешения экрана приложения и выход из приложения;
* Интерфейс выбора заданий, содержащий 4 задания;
* Интерфейс управления объектом на сцене, отвечающий за включения и выключения устройства, все необходимые кнопки взаимодействия, лампочки, демонстрирующие корректную работу устройства, и дисплеи для отображения параметров компонентов устройства.

### Решаемые задачи

Необходимо разработать пользовательский интерфейс приложения для взаимодействия пользователя с 3D-моделью установки, а также необходимые компоненты управления для взаимодействия с пользовательским интерфейсом.

### Описание 3D-модели теплового узла

Тепловой узел (рисунок 1) – комплекс оборудования, монтаж проекта которых обеспечивается с целью предоставления принципиального учета и регулирования энергии, объема теплоносителя, а также произведение регистрации и контроля его параметров [8].

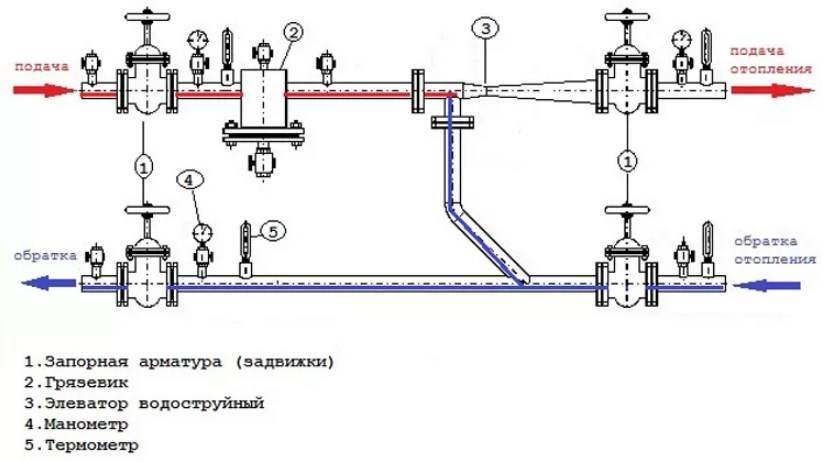


Рисунок 1 – схема теплового узла

Тепловой узел способен выполнять от одной до нескольких функций, таких как:

* Измерение количества и массы тепловой энергии, ее давления, температуры жидкости, циркулирующей по трубопроводу и времени функционирования.
* Накопление и хранение этой информации на локальном носителе.
* Отображение ее на приборах учета.

На основе полученных данных осуществляется проверка за работой отопительного оборудования, его регулирование и обслуживание.

Учетным прибором выступает такое устройство, как счетчик, схема которого состоит из:

* Термопреобразователя сопротивлений.
* Тепловычислителя.
* Первичного преобразователя расхода.

Зависимо от того, установка какой модели первичного преобразователя имела место, теплосчетчик может иметь в своем составе фильтры и датчики давления.

Узел учета тепловой энергии состоит из следующих элементов:

* Запорной арматуры.
* Теплового счетчика.
* Термопреобразователя.
* Грязевика.
* Расходомера.
* Теплового датчика обратного трубопровода.
* Дополнительного оборудования.

## Метод решения задач

В качестве современных методов разработки в выбранной области используется движок Unreal Engine, движок Unity или платформа Unigine. Был выбран движок Unreal Engine, так как он соответствует всем необходимым требованиям к разработке, в нем используется язык программирования С++, позволяющий разработать более оптимизированный продукт, наличие Blueprint для разработки визуальной составляющей приложения, а также гибкие инструменты для создания и настройки пользовательского интерфейса.

## Описание программных и технических средств для разработки приложения

Для работы в движке Unreal Engine 5 требуется дисковое пространство объемом более 115 ГБ, четырех-ядерный процессор Intel или AMD с частотой не менее 2.5 ГГЦ, объем оперативной памяти не менее 8 ГБ и видеокарта, совместимая с DirectX 11, DirectX12, Vulkan (для видеокарты AMD версии 21.11.3+ и для видеокарты Nvidia версии 496.76+) [7].

Для разработки приложения требуются:

* Операционная система Microsoft Windows 10;
* Интегрированная среда разработки Visual Studio 2019 версии 16.11.5 или более позднюю, Visual Studio 2022;

Для успешной сборки проекта так же требуется установить рабочие нагрузки для Visual studio:

* Разработка игр на языке C++;
* Разработка классических приложений .NET;
* Разработка классических приложений на C++.

## Входная информация

Для управления контроллером в качестве входной информации используются клавиши W, A, S, D.

Для отображения курсора на экране используется клавиша Alt, которая является входной информацией.

Нажатия на ЛКМ и ПКМ тоже используются как вводная информация для взаимодействия с интерактивными объектами виртуальной системы.

В главном меню присутствуют три варианта кнопки: Свободный режим, Задания, Выход. Нажатие на каждую из этих кнопок является вводом информации.

Далее взаимодействие с программой будет происходить за счет нажатий на кнопки пульта управления, для раскрытия которого необходимо нажать на кнопку, отвечающую за его раскрытие, это действие будет считаться вводом информации.

## Выходная информация

Выходными данными являются изменения значений переменных и отображение на экране поведения установки.

Клавиши W, A, S, D управляют контроллером, выходными данными в данном случае являются перемещения камеры, соответственно, вперед, влево, назад и вправо.

Клавиша Alt отображает курсор на экране.

Нажатие на ПКМ запускает метод изменения прозрачности компонента, и деталь становится прозрачной.

С помощью ЛКМ осуществляется взаимодействие с интерфейсом приложения.

Кнопка свободного режима в главном меню очищает экран от неиспользуемых виджетов и добавляет на сцену необходимые для свободного режима виджеты. Кнопка заданий очищает экран от неиспользуемых виджетов и добавляет на экран необходимые для режима заданий виджеты. Кнопка выхода завершает работу программы.

# ПРОГРАММНАЯ РЕАЛИЗАЦИЯ ПОСТАВЛЕННОЙ ЗАДАЧИ

## Создание проекта

Для создания проекта необходимо запустить Unreal Engine 5 и в появившемся окне выбрать следующие пункты:

* Games
* blank
* C++
* Desktop
* Maximum
* Starter Content

Указываем расположение проекта и его название. После этого нажатием на кнопку *Create* (рисунок 2) [1].

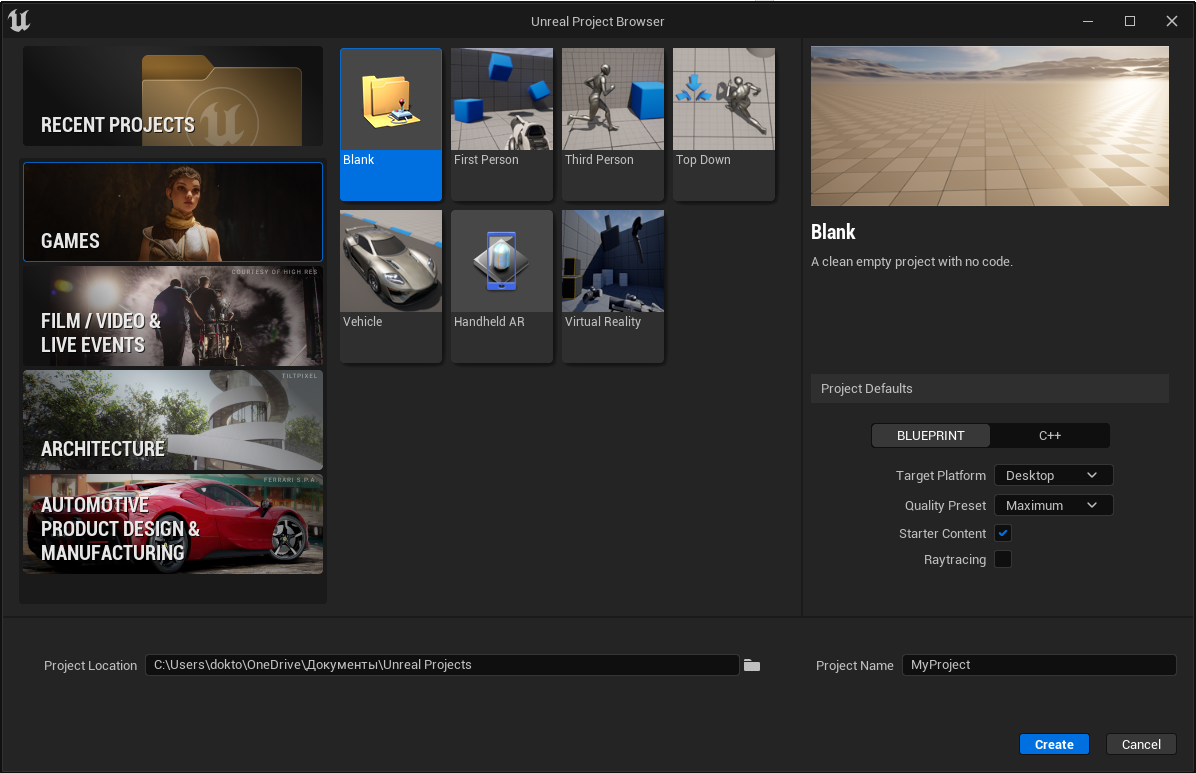


Рисунок 2 – окно создания проект Unreal Engine

## Создание управляемого персонажа

Для того, чтобы пользователь мог перемещаться по сцене, взаимодействовать с объектами или использовать пользовательский интерфейс необходимо создать персонажа.

Первым делом необходимо создать класс базового персонажа, в котором будут реализованы основные методы, присущие игровому персонажу.

Для создания класса необходимо в *asset browser* перейти в папке С++ Classesи в ней нажатием ЛКМ вызвать окно выбора и в нем нажать *New C++ Class…* (рисунок 3).

Для удобного взаимодействия с С++ классами лучше давать им корректные имена, кратко описывающие суть класса и придерживаться файловой структуры путем разделения классов по папкам [4].

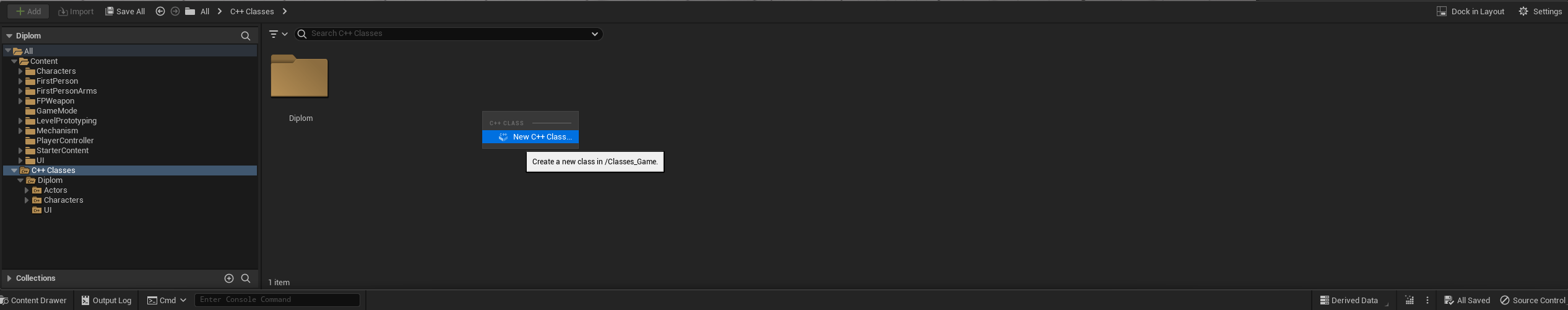


Рисунок 3 – создание нового С++ класса

Откроется окно настроек создаваемого класса. В нем мы выбираем от какого класса будет наследоваться новый класс. Выбираем в качестве родительского класса *Character*, нажимаем *Next*, выбираем путь, в котором создаться класс и нажимаем *Create Сlass* (рисунок 4).

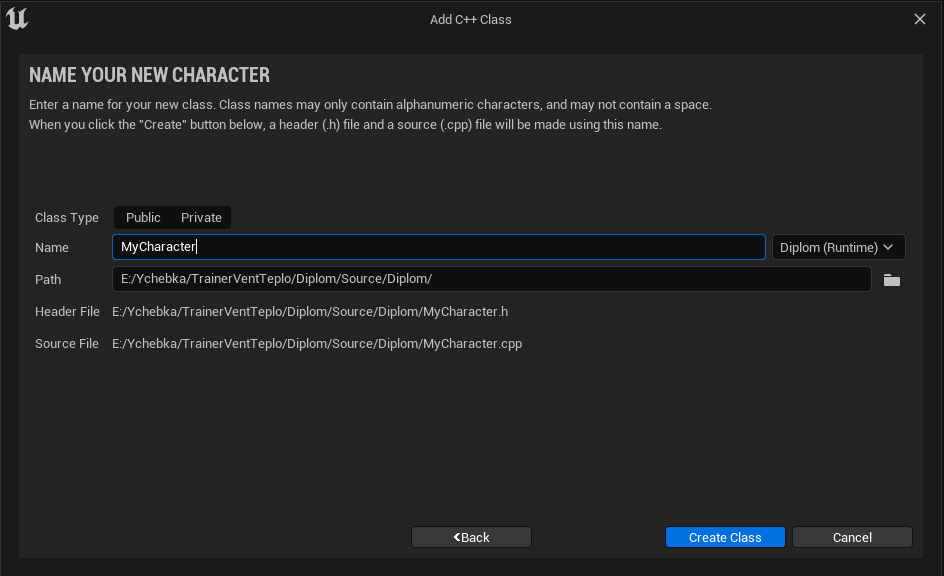


Рисунок 4 – окно настроек создаваемого С++ класса

После загрузки откроется Visual Studio с созданным классом, содержащим в себе базовые методы, унаследованные от родительского класса – их можем стереть, пока что они не нужны. Кроме них будут добавлены включения родительского класса и так же файл генерации класса – он всегда должен быть последним в директиве препроцессора. Так же обязательно должны присутствовать строки:

* *UCLASS(…)* при помощи нее компилятор определяет классы, отображаемые и встраиваемые в приложение Unreal Engine. В скобках указываются параметры для этого класса.
* *GENERATED\_BODY()* – при помощи нее компилятор создает класс, принадлежащий структуре приложения Unreal Engine

Для облегчения работы с кодом для Unreal Engine можно использовать расширение Visual Assist.

В заголовочном файле определяем необходимые методы и свойства. Важно разделять их на публичные, защищенные и приватные уровни доступа.

Необходимые методы и свойства представлены в листинге 1.

Листинг 1 – заголовочный файл класса базового персонажа



В заголовочном файле декларируем делегат для определения найденного объекта, используем декларирование интерфейса *IInteractable* [11] объявляем следующие публичные методы:

* *virtual void EndPlay(const EEndPlayReason::Type Reason) override* – нужен для завершения взаимодействия с объектом с указанием причины. Переопределен из родительского класса.
* *virtual void Tick(float DeltaSeconds)* – метод, срабатывающий каждый тик с момента создания экземпляра класса на сцене.
* *virtual void MoveForward(float Value){}, virtual void MoveRight(float Value){}, virtual void Turn(float Value){}, virtual void LookUp(float Value) {}* – методы, необходимые для управления персонажем. Являются пустыми так как реализованы будут в классе управляемого персонажа.
* *virtual void ChangeMouseCursor() {}* – метод, отвечающий за отображение или скрытие курсора на экране пользователя.
* *virtual void Interact()* – метод, реализующий взаимодействие с объектом.
* *virtual void SetOpacity()* – метод, настраивающий прозрачность у объекта
* *virtual void SetHighlightObject(), virtual void RemoveHighlightObject()* – методы, вызывающие выделение объекта и скрывающие выделение объекта соответственно.
* *FOnIntractableObjectFound OnIntractableObjectFound* – объявление делегата.

Далее инициализируем защищенные методы:

* *UPROPERTY(EditDefaultsOnly, BlueprintReadOnly, Category = "Character | LineTrace") float LineOfSightDistance = 300.0f* – переменная, отвечающая за дальность выпускаемого персонажем луча [11].
* *void TraceLineOfSight()* – метод выпускания луча [11].
* *TScriptInterface<IInteractable> LineOfSightObject, TScriptInterface<IInteractable> OldLineOfSightObject* – объекты, реализующие интерфейс *IIneractable* [11].

Используем *UPROPERTY()* для указания тех свойств, которые должны быть у открыты для редактирования в редакторе Unreal Engine.

Далее переходим к реализации ранее объявленных методов. При использовании Visual Assist нажимаем ПКМ по методу, в открывшемся окне выбираем список *Quick action and refactoring (VA)* и в нем выбираем *Create Implementation* (рисунок 5). Если Visual Assist не используется, то вручную пишется реализация для метода в исполняемом файле.

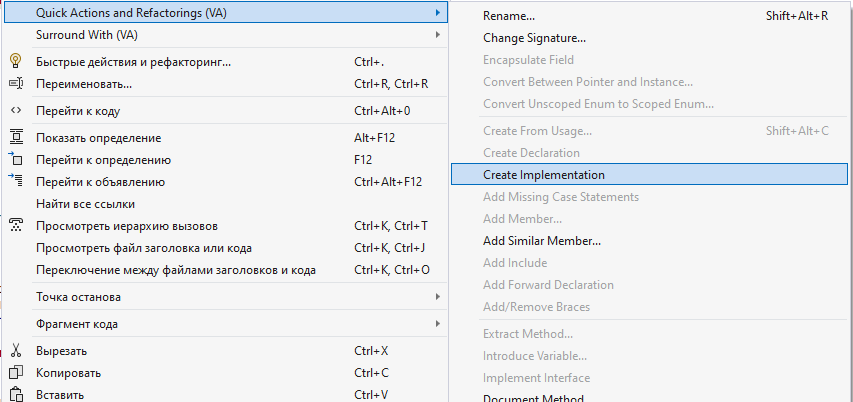
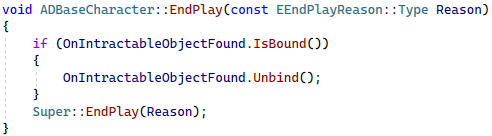


Рисунок 5 – создание реализации класса

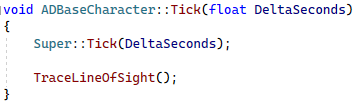
В реализации метода *EndPlay* проверяем, привязан ли делегат, и, если да – отвязываем его от персонажа. Вызываем метод базового класса, в который передаем причину (листинг 2).

Листинг 2 – реализация метода *EndPlay*



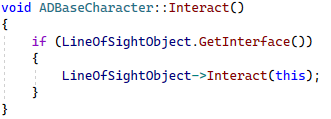
В реализации метода *Tick* вызываем метод базового класса и метод *TraceLineOfSight* [11] (листинг 3).

Листинг 3 – реализация метода Tick



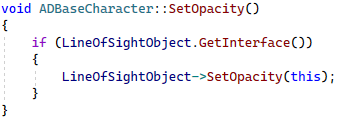
В реализации метода *Interact* проверяем, реализует ли объект *LineOfSightObject* интерфейс и вызываем метод из класса этого объекта [11] (листинг 4).

Листинг 4 – реализация метода Interact



В реализации метода *SetOpacity* проверяем, реализует ли объект *LineOfSightObject* интерфейс и вызываем метод из класса этого объекта [11] (листинг 5).

Листинг 4 – реализация метода SetOpacity



В реализации методов *SetHighlightObject* и *RemoveHighlightObject* проверяем, реализуют ли объекты *LineOfSightObject* и *OldLineOfSightObject* интерфейс и вызываем методы из этого класса. В случае *SetHighlightObject* приравниваем *LineOfSightObject* и *OldLineOfSightObject*, а в *RemoveHighlightObject* присваиваем объекту *OldLineOfSightObject* значение *nullptr* (листинг 6).

Листинг 6 – реализация методов *SetHighlightObject* и *RemoveHighlightObject*





Весь код файла *DBaseCharacter.cpp* представлен в приложении А.

## Подсистема контроллера

Для управления персонажем, созданным ранее, необходим контроллер. Создаем новый класс, наследуемый от *PlayerController*. В заголовочном файле инициализируем следующие публичные методы и свойства:

* *virtual void SetPawn(APawn\* InPawn) override* – метод для выставления персонажа, управляемого контроллером.
* *UPlayerHUDWidget\* PlayerHUDWidget = nullptr* – ссылка на класс пользовательского интерфейса, в качестве значения по умолчанию указываем nullptr.

Далее инициализируем защищенные методы и свойства:

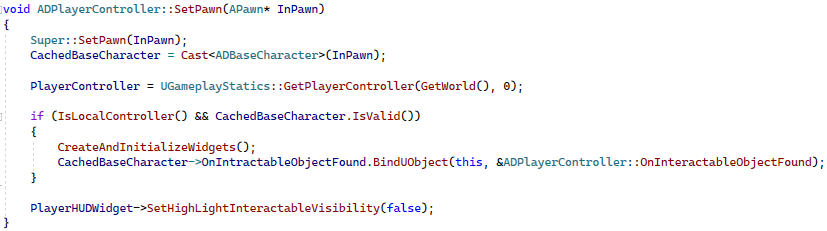
* *virtual void SetupInputComponent() override* – метод, отвечающий за привязку физических кнопок мыши, клавиатуры или геймпада к методам, реализующим управление персонажем.
* *UPROPERTY(EditDefaultsOnly, BlueprintReadOnly, Category = "Widgets") TSubclassOf<class UPlayerHUDWidget> PlayerHUDWidgetClass* – ссылка на родительский для блупринта пользовательского интерфейса класс.
* *UPROPERTY(EditDefaultsOnly, BlueprintReadOnly, Category = "Widgets") TSubclassOf<class AComponentOfMechanism> CopmonentOfMechanismClass* – ссылка на объект, наследуемый от класса интерактивного объекта [11].
* Далее инициализируем приватные методы и свойства:
* *void OnInteractableObjectFound(FName ActionName)* – метод, реализующий поиск поиск возможного пересечения с интерактивным объектом [11].
* *void MoveForward(float Value), void MoveRight(float Value), void Turn(float Value), void LookUp(float Value)* – методы, реализующие управление персонажем.
* *void ChangeMouseCursor()* – метод для изменения состояния курсора.
* *void Interact()* – метод взаимодействия с объектом.
* *void SetOpacity()* – метод, реализующий настройку прозрачности объекта.
* *void CreateAndInitializeWidgets()* – метод создания и добавления виджета пользовательского интерфейса.
* *APlayerController\* PlayerController* – ссылка на базовый контроллер.
* *AComponentOfMechanism\* ComponentsOfMechanism* – ссылка на класс интерактивных компонентов.
* *TSoftObjectPtr<class ADBaseCharacter> CachedBaseCharacter* – ссылка на класс базового персонажа.

Полный код заголовочного файла *PlayerController* представлен в приложении А.

Далее необходимо реализовать объявленные методы.

В методе *SetPawn* приводим ссылку на персонажа к типу Pawn, записываем значение базового контроллера. После этого проверяем персонажа на валидность и управляется ли контроллер пользователем и в теле условия вызываем метод создания виджета, а также привязываем персонажа к делегату поиска интерактивных объектов (листинг 7).

Листинг 7 – метод *SetPawn*



В методе *SetupInputComponent* привязываем действие к методу (листинг 8). Клавиши, отвечающий за действие можно настроить из редактора Unreal Engine, для этого нажимаем *Settings* и выбираем *Project Settings* (рисунок 6).

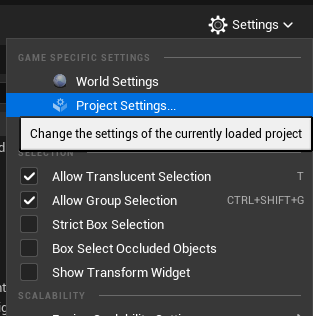


Рисунок 6 – открытие настроек

В открывшемся окне переходим на вкладку Input, в котором для *Action Mapping* и *Axis Mapping* указываем название действия, клавишу и для *Axis Mapping* значение (рисунок 7).

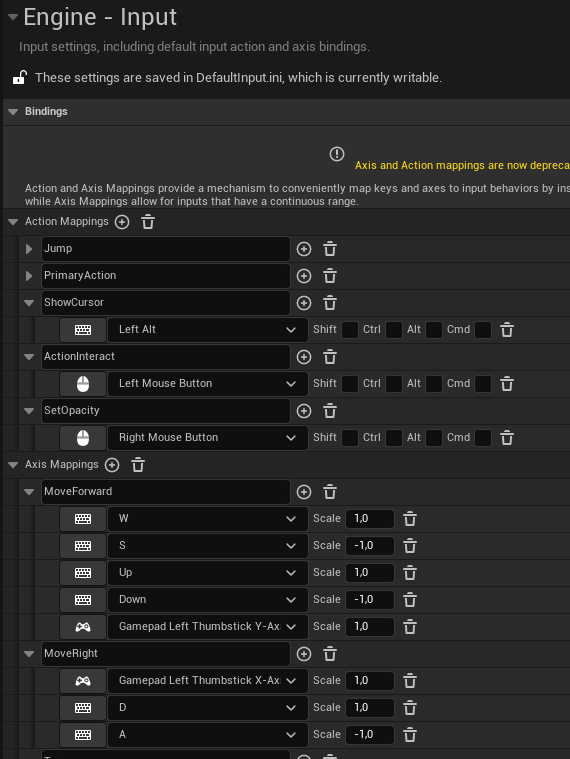
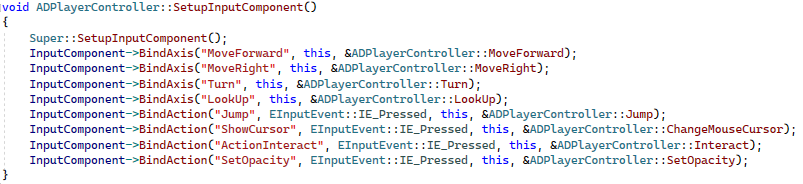


Рисунок 7 – настройка клавиш

Листинг 8 – метод SetupInputComponent



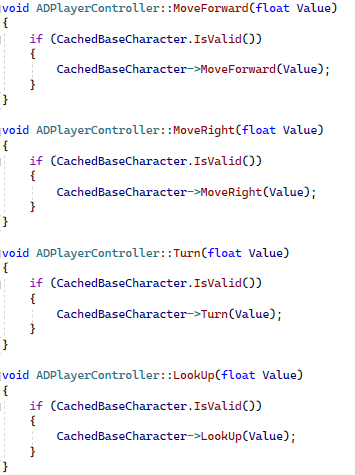
В методе *OnInteractableObjectFound* проверяем на валидность виджет интерфейса, и, если он не валиден делаем ранний выход. Далее следует конструкция, создающая массив и записывающая в него возможные клавиши для взаимодействия с объектом, если такие клавиши присутствуют, то включаем выделение объекта, если таких клавиш нет, то выделение снимается (листинг 9).

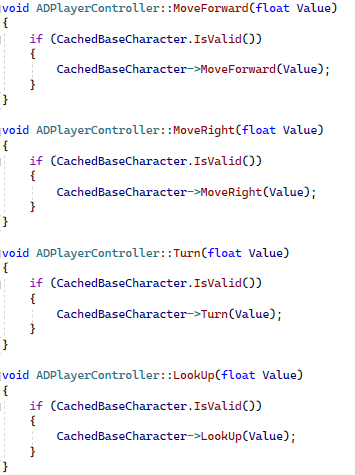
Листинг 9 – метод *OnInteractableObjectFound*



Далее следуют методы *MoveForward*, *MoveRight*, *Turn*, *LookUp*, отвечающие за движение персонажа. В них проверяется валидность персонажа и вызывается соответствующий метод класса персонажа (листинг 10).

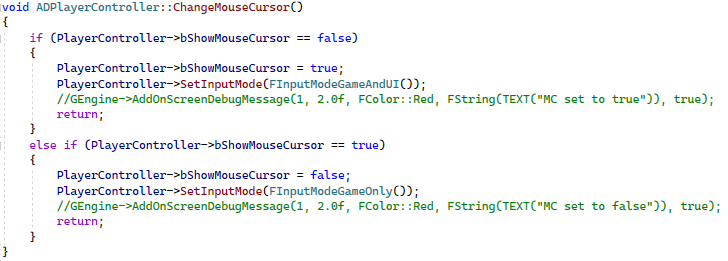
Листинг 10 – методы *MoveForward*, *MoveRight*, *Turn*, *LookUp*





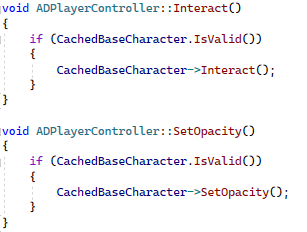
В методе *ShowMouseCursor* в случае если он не отображается, включается его отображение и параметры ввода переключаются в режим для интерфейса, а в случае когда он отображается, его отображение выключается и параметры ввода переходят в игровой режим (листинг 11).

Листинг 11 – метод *ShowMouseCursor*



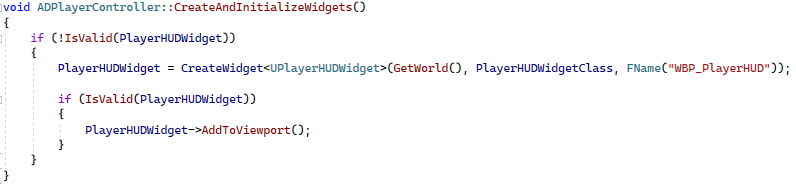
В методах *Interact* и *SetOpacity* происходит проверка на валидность персонажа и вызов метода из класса персонажа (листинг 12).

Листинг 12 – методы *Interact* и *SetOpacity*



В методе *CreateAndInitializeWidgets* происходит проверка валидности виджета интерфейса и если он не валиден, то ему присваивается значение, после чего он добавляется на экран (листинг 13).

Листинг 13 – метод *CreateAndInitializeWidgets*



Полный код файла PlayerController.cpp представлен в приложении А.

Далее необходимо создать класс управляемого персонажа. В заголовочном файле объявляем следующие публичные методы:

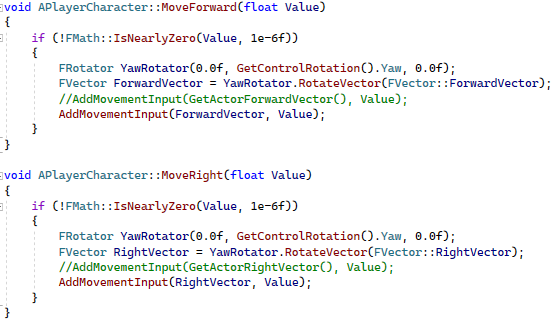
* *virtual void MoveForward(float Value) override, virtual void MoveRight(float Value) override, virtual void Turn(float Value) override, virtual void LookUp(float Value) override* – они будут реализовывать перемещение персонажа.
* *APlayerCharacter()* – конструктор по умолчанию.

Объявляем защищенные свойства класса:

* *UPROPERTY(VisibleAnywhere, BlueprintReadOnly, Category = "Character | Camera") class UCameraComponent\* CameraComponent* – компонент камеры.
* *UPROPERTY(VisibleAnywhere, BlueprintReadOnly, Category = "Character | Camera") class USpringArmComponent\* SpringArmComponent* – компонент «пружины». Нужен для плавности поведения камеры.

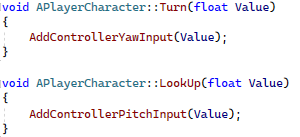
В реализации методов *MoveForward* и *MoveRight* добавляем движение (с учетом погрешности значения, которое в основном применимо для геймпада) по соответствующему вектору (листинг 14).

Листинг 14 – методы *MoveForward* и *MoveRight*



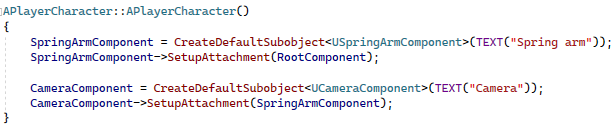
В методах *Turn* и *LookUp* добавляем вращение по осям X и Z соответственно (листинг 15).

Листинг 15 – методы *Turn* и *LookUp*



В конструкторе по умолчанию создаем компоненты камеры и «пружины» в корень объекта персонажа (листинг 16).

Листинг 16 – конструктор по умолчанию



После этого может создать блупринт персонажа, указав в качестве родительского класса PlayerCharacter (рисунок 8).

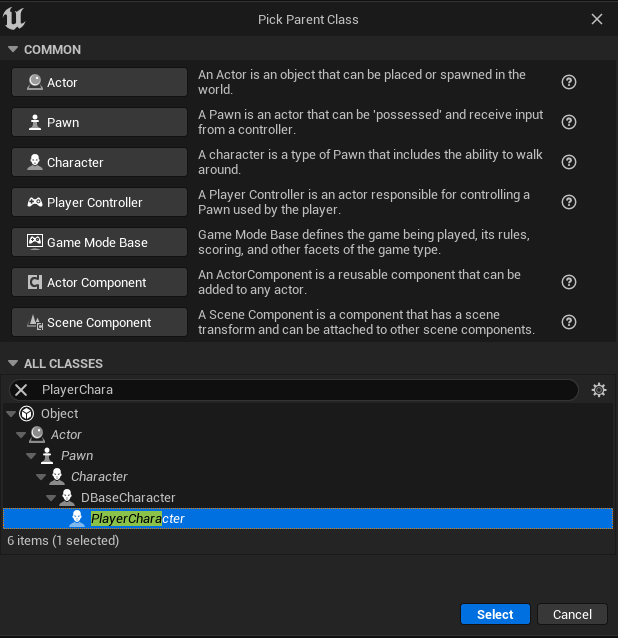


Рисунок 8 – создание блупринта персонажа

Таким же способом создадим блупринт контроллера персонажа.

Далее для указания необходимого персонажа и контроллера нужно создать новый класс *GameMode*.

Создадим новый класс, наследованный от *GameModeBase*. Оставим его без изменений, так как он реализует базовые методы родительского класса.

Наследуясь от него, создадим блупринт и в *World Settings* укажем его в качестве *GameMode* *Override*, после чего укажем блупринт управляемого персонажа и блупринт контроллера (рисунок 9) [6].

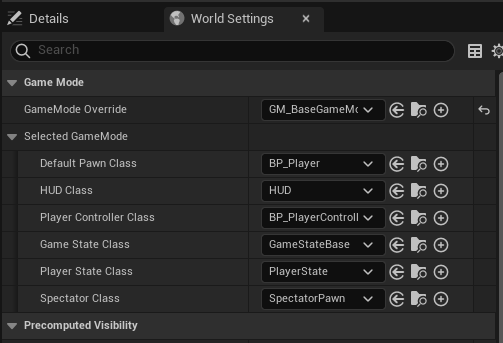


Рисунок 9 – World Settings

## Создание пользовательского интерфейса

Для взаимодействия пользователя с объектами на сцене использован пользовательский интерфейс. В Unreal Engine для его создания используется *Widget Blueprint*.

При старте программы будет отображаться интерфейс меню, в котором присутствуют кнопки «Войти в свободный режим», «Выбрать задание» и «Выход».

Для создания виджета интерфейса необходимо в меню создания выбрать *Widget Blueprint* (рисунок 10) [10].

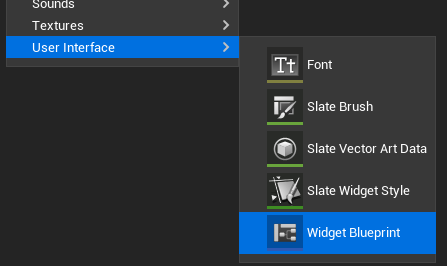


Рисунок 10 – создание виджета интерфейса

В созданном виджете необходимо добавить Canvas, перетащив его из списка элементов управления на окно дизайнера (рисунок 11).

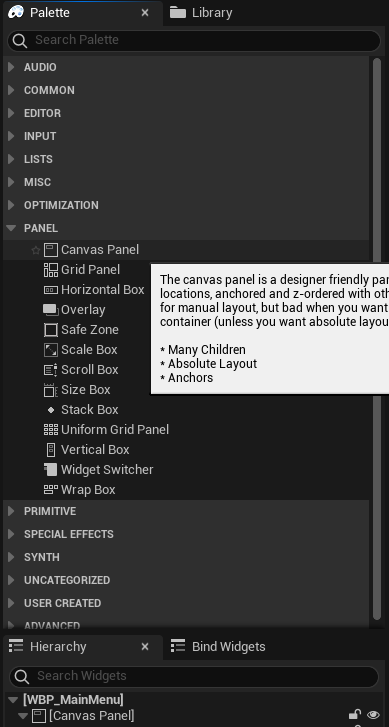


Рисунок 11 – canvas

Аналогичным методом добавляем кнопки и вставляем в них изображение (рисунок 12).

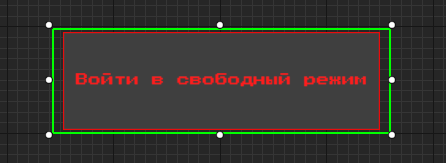


Рисунок 12 – настройка кнопок

Для того, чтобы кнопки могли выполнять какие-либо функции им нужно сделать обработчик *On Clicked* (рисунок 13).



Рисунок 13 – создание обработчика

В открывшемся Event Graph добавляем ноды:

* Set Show Mouse Cursor.
* Get Player Controller.
* Remove All Widgets.
* Create Widget.
* Add to viewport.

Соединяем эти ноды как представлено на рисунке 14.

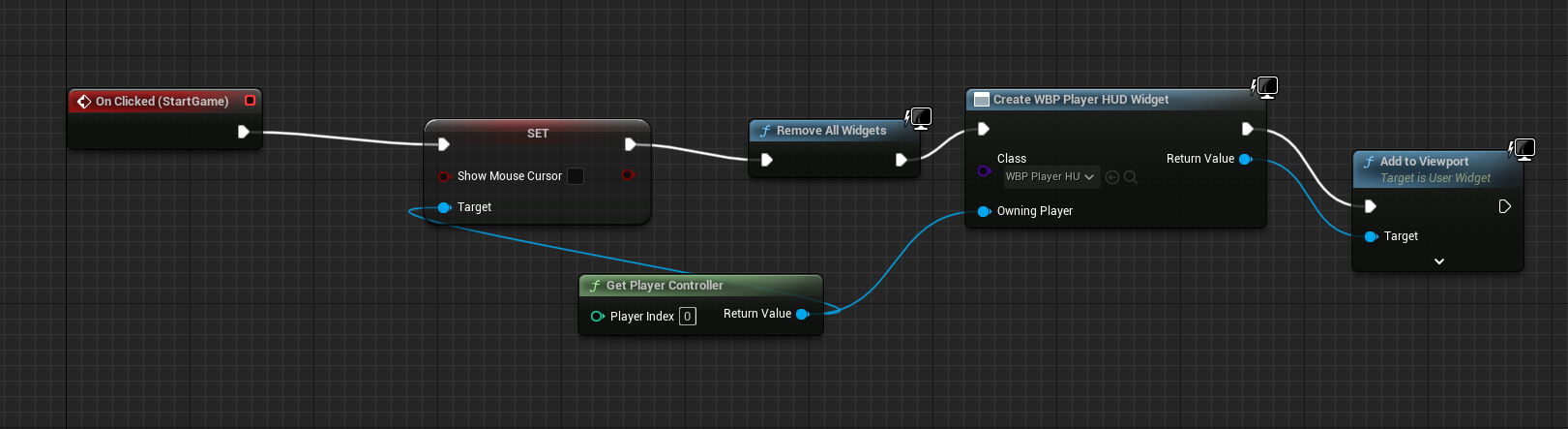


Рисунок 14 – блупринт для кнопки Start Game

Аналогично делаем для кнопки Quest.

Для кнопки Exit реализуем следующий блупринт (рисунок 15).

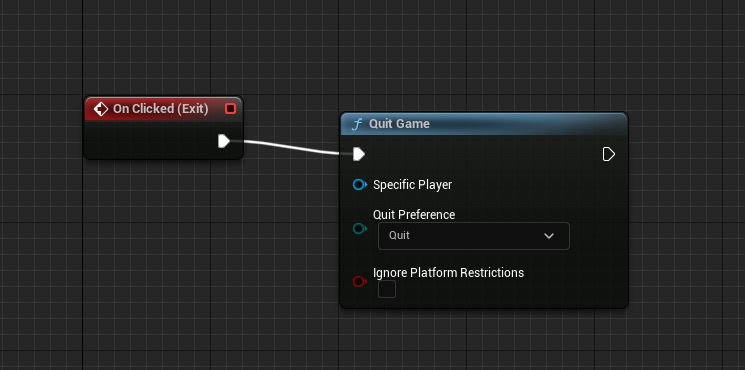
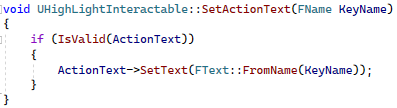


Рисунок 15 – блупринт для кнопки Exit

Далее создадим вспомогательный виджет интерфейса *HighlightInteractable*. Для его создания нужен родительский класс. Создадим класс, наследованный от *UserWidget*. В заголовочном файле указываем публичный метод *SetActionText*, который будет показывать описание, и защищенное свойство *UPROPERTY(meta = (BindWidget)) UTextBlock\* ActionText* – сам текст для отображения.

В исполняемом файле создаем реализацию метода, в котором присваиваем переменной *ActionText* значение (листинг 17).

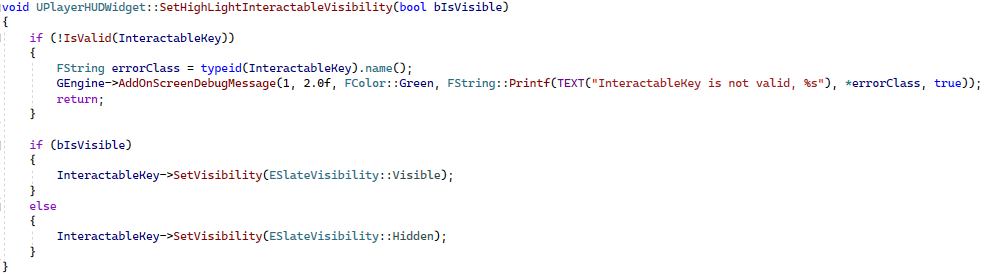
Листинг 17 – метод *SetActionText*



В виджете добавляем canvas и текст, который называем *ActionText* для связи с переменной класса.

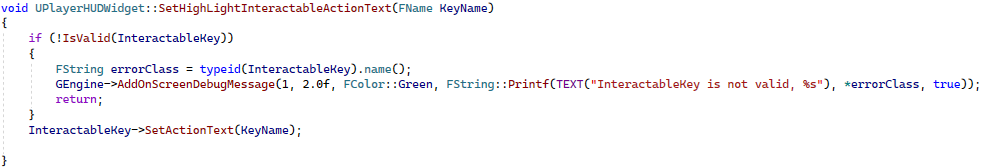
Далее создадим основной виджет интерфейса, при котором будем управлять персонажем. Для него так же необходим родительский класс. В классе объявляем два публичных метода – *SetHighLightInteractableVisibility* и *SetHighLightInteractableActionText*, которые отвечают за отображение и наполнение текста соответственно, так же объявляем защищенное свойство *UPROPERTY(meta = (BindWidget)) class UHighLightInteractable\* InteractableKey = nullptr* – ссылка на ранее созданный объект виджета интерфейса. В реализации метода *SetHighLightInteractableVisibility* устанавливаем значение видимости текста (листинг 18).

Листинг 18 – метод *SetHighLightInteractableVisibility*



В реализации метода *SetHighLightInteractableActionText* выставляем необходимый текст (листинг 19).

Листинг 19 – метод *SetHighLightInteractableActionText*



Изменим родительский класс виджета на необходимый (рисунок 16).

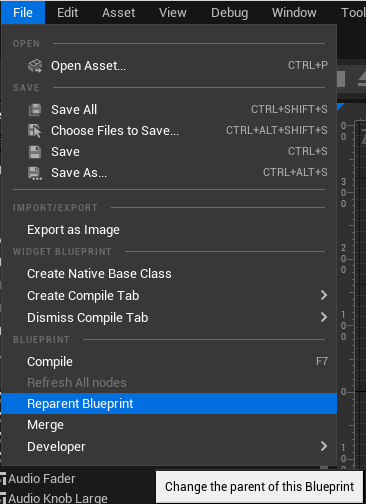


Рисунок 16 – изменение родительского класса

В него добавляем canvas, кнопку и созданный ранее виджет *HighlightInteractable* (рисунок 17).

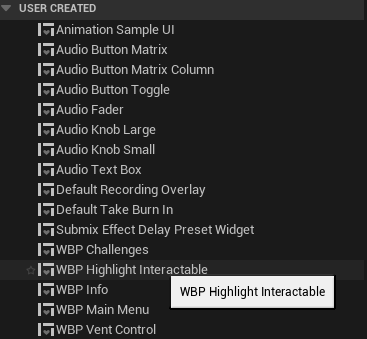


Рисунок 17 – добавление созданного виджета

Так же необходимо настроить кнопку открытия пульта управления. Для этого создадим следующий блупринт (рисунок 18).

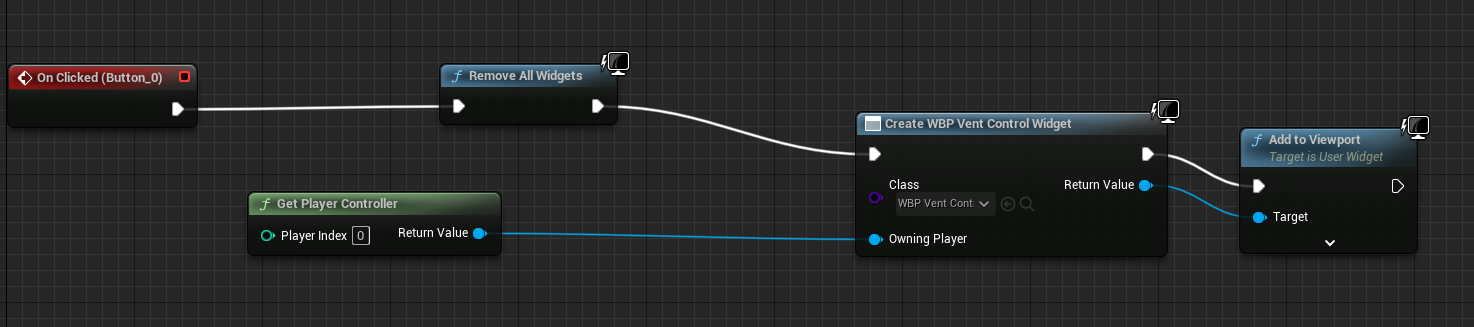


Рисунок 18 – блупринт для кнопки вызова пульта управления

Конечный результат основного виджета представлен на рисунке 19.

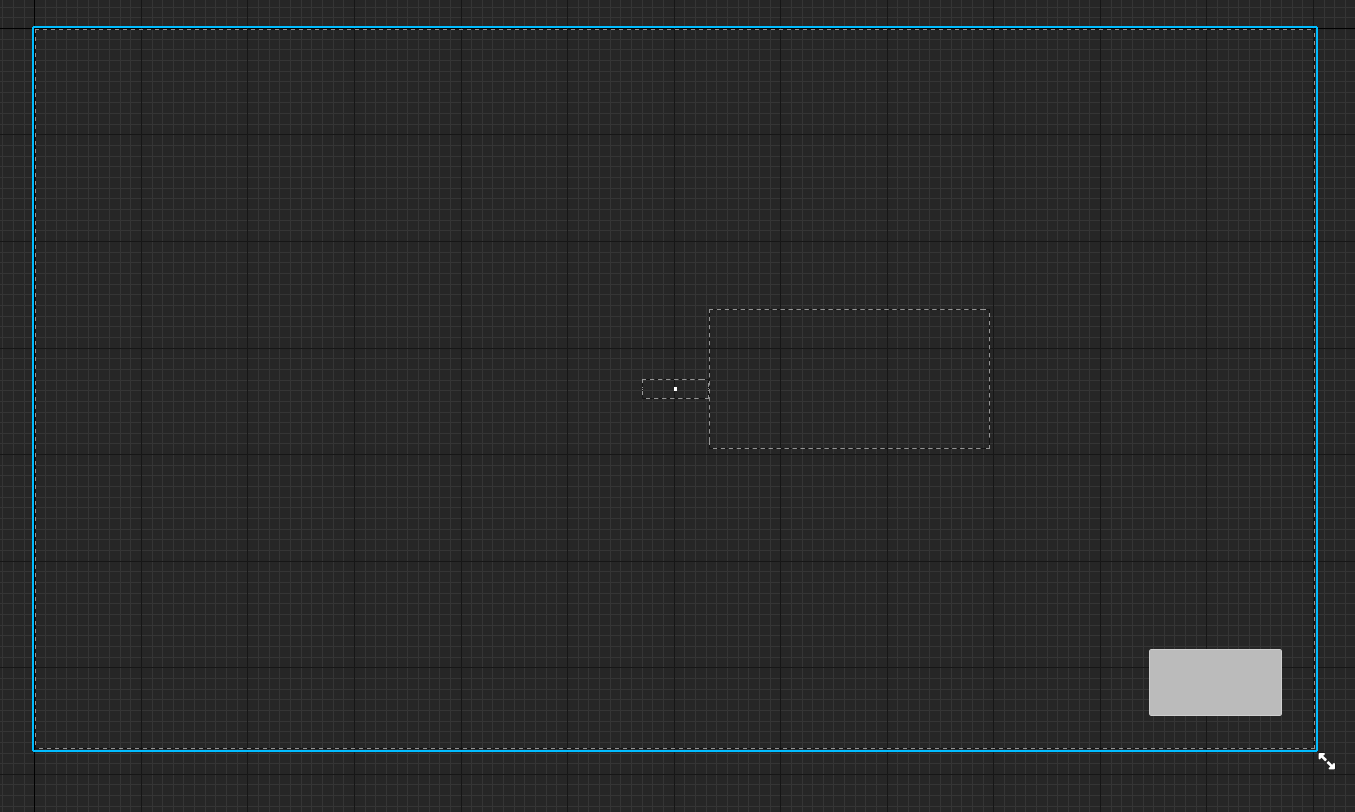


Рисунок 19 – основной виджет

Далее нужно создать класс для интерфейса пульта управления. В заголовочном файле объявим следующие публичные методы:

* *UFUNCTION(BlueprintCallable) void PlusTemp()* – метод прибавления температуры.
* *UFUNCTION(BlueprintCallable) void SubTemp()* – метод убавления температуры.

Объявим защищенное свойство *UPROPERTY(EditAnywhere, BlueprintReadWrite, Category = "MaterialPipe") AComponentOfMechanism\* TeploPipe* – ссылка на объект, с которым будет работать пульт управления.

В реализации методов вызываем необходимые методы из класса объекта взаимодействия (листинг 20).

Листинг 20 – методы PlusTemp и SubTemp





Создаем блупринт интерфейса, наследованный от этого класса, добавляем в него canvas, кнопки управления, текст и изображение, расположим в нужном месте (рисунок 20).

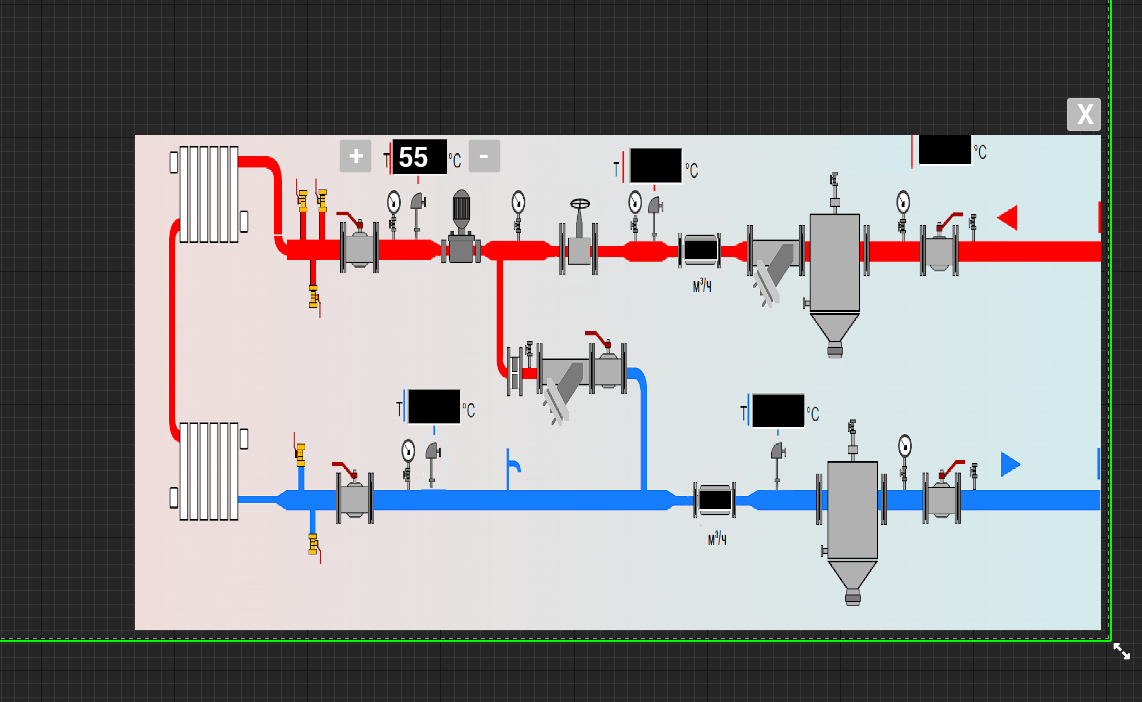


Рисунок 20 – виджет пульта управления

Настраиваем кнопки в Event Graph (рисунок 21).

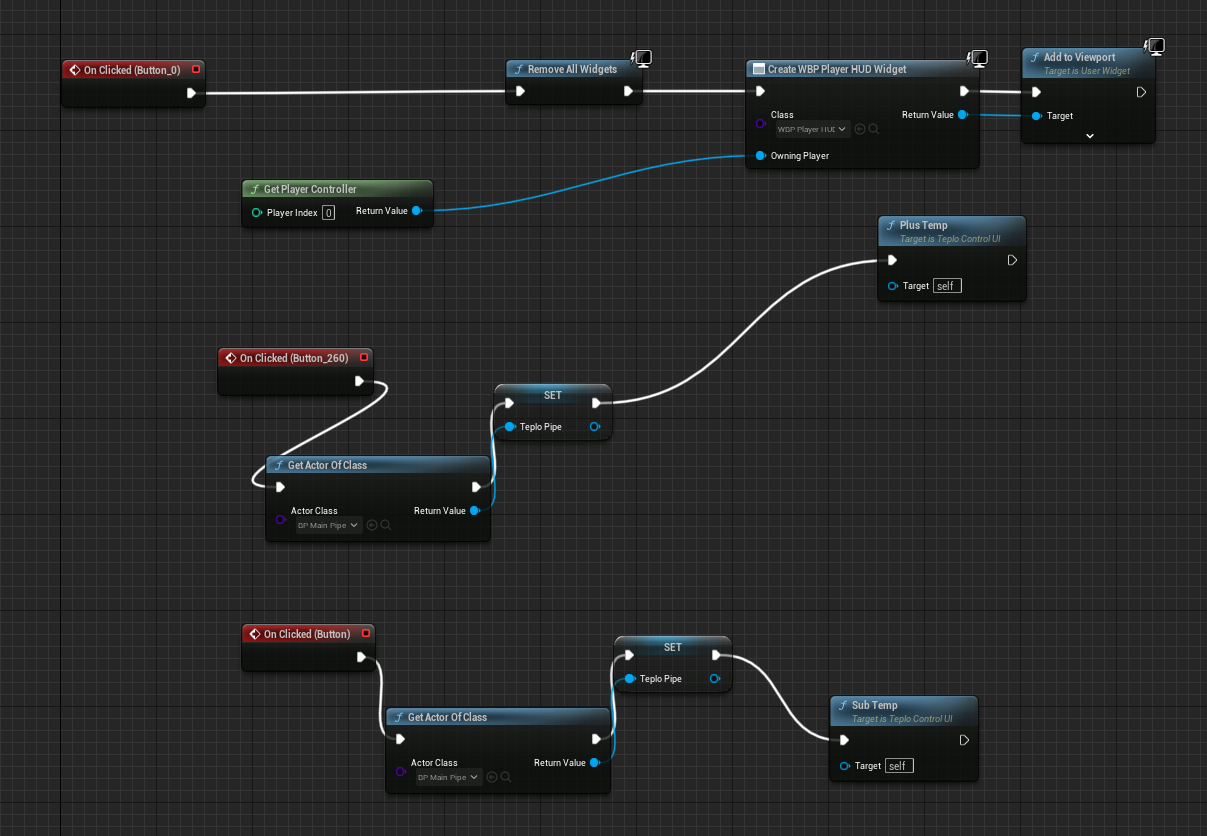


Рисунок 21 – блупринт пульта управления

Аналогичным способом создаем виджет для меню заданий (рисунок 22), в котором по нажатию на кнопку будет открываться свободный режим (рисунок 23).

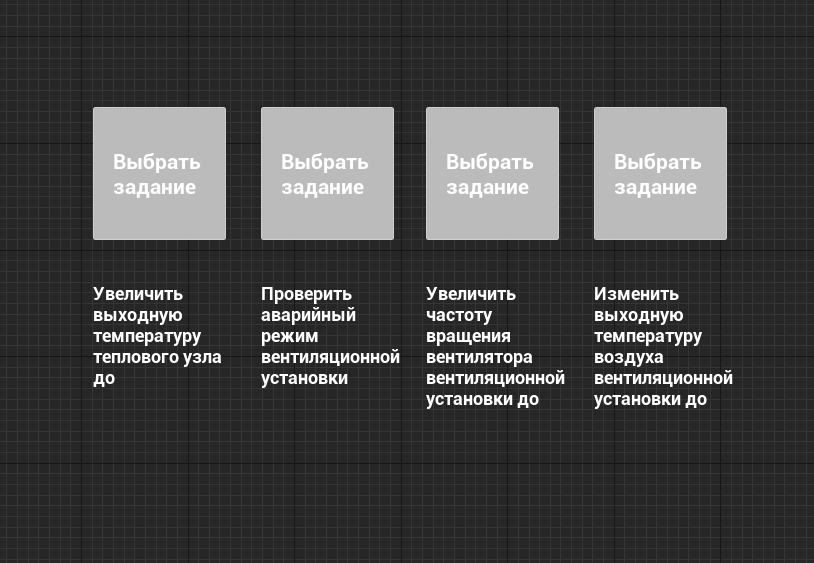


Рисунок 22 – виджет меню заданий

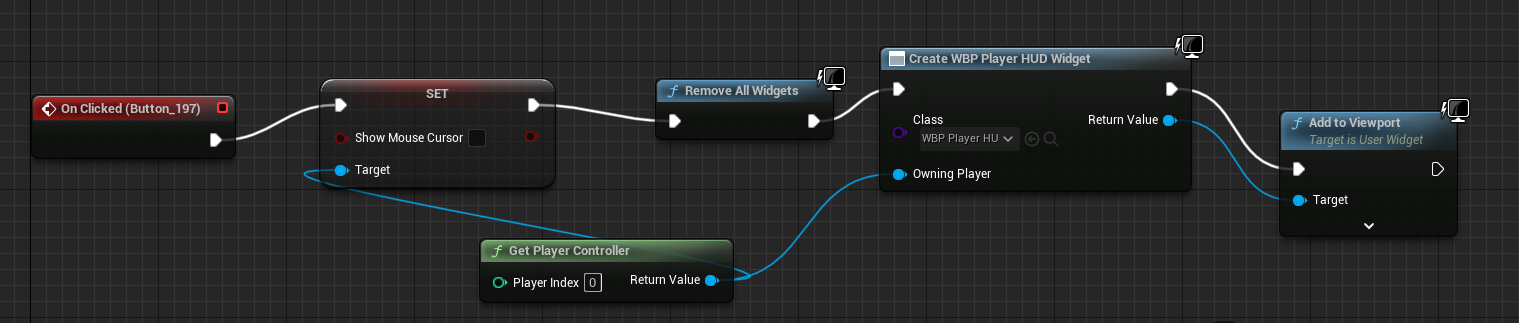


Рисунок 23 – блупринт меню заданий

# ОПИСАНИЕ ПРОГРАММЫ

## Структура программы

В состав программы входят следующие подсистемы:

* Подсистема 3D-модели установки
* Подсистема интерфейса
* Подсистема контроллера

**Подсистема 3D-модели** предназначена для отображения модели установки на сцене [11].

**Подсистема интерфейса** предназначена для обеспечения взаимодействия пользователя с 3D-моделью установки путем изменения свойств ее элементов.

**Подсистема контроллера** предназначена для управления персонажем. Обрабатывает все действия, выполненные персонажем.

Структурная схема системы представлена на рисунке 24.

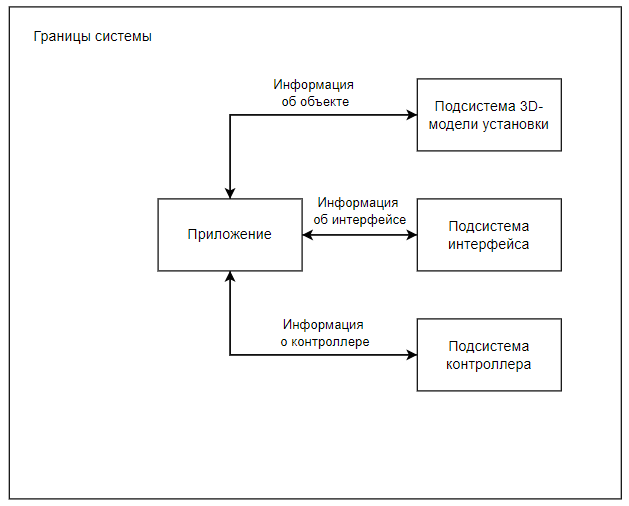


Рисунок 24 – структурная схема системы

## Сведения об операционной системе

Для работы с приложением необходима операционная система Windows 10 64-разрядная.

При выборе операционной системы учитывались такие характеристики, как:

* Активное обновление операционной системы;
* широкое использование;
* большое количество специалистов поддержки;
* стабильная работа средства разработки;
* стабильная работа приложения.

Microsoft Windows 10 удовлетворяет все эти критерии, поэтому выбрана в качестве используемой операционной системы.

# ЗАКЛЮЧЕНИЕ

В результате проделанной работы был создан пользовательский интерфейс для взаимодействия пользователя с программой и установками теплового узла, реализована подсистема контроллера, отвечающая за входную информацию и вызывающая методы ее обработки, а также настройка персонажа и связь с подсистемами контроллера и интерфейса.

Практическая ценность полученных результатов заключается в разработке программного продукта, позволяющего пользователю управлять персонажем, взаимодействовать с пользовательским интерфейсом и ознакомится с устройством теплового узла.

# БИБЛИОГРАФИЧЕСКИЙ СПИСОК

1. *Документация по Unreal Engine 5.0* / [Электронный ресурс] // Unreal Engine : [сайт]. — URL: https://docs.unrealengine.com/5.0/ (дата обращения: 20.11.2022).
2. *Qwent* : инженерная компания : сайт. / ООО «СТРОЙИНЖПРОЕКТ» — Москва, 2011—2023. — URL: https://ceds.ru (дата обращения: 23.01.2023).
3. *Программирование и скриптинг в Unreal Engine 5.0* / [электронный ресурс] // Unreal Engine: сайт. — США, 2004—2023. — URL: https://docs.unrealengine.com/5.0/en-US/unreal-engine-programming-and-scripting/ (дата обращения: 23.01.2023).
4. *Справочник по API в Unreal Engine 5.0* / [электронный ресурс] // Unreal Engine: сайт. — США, 2004—2023. — URL: https://docs.unrealengine.com/5.0/en-US/API/ (дата обращения: 23.01.2023).
5. *Материалы в Unreal Engine 5.0* / [электронный ресурс] // Unreal Engine: сайт. — США, 2004—2023. — URL: https://docs.unrealengine.com/5.0/en-US/unreal-engine-materials/ (дата обращения: 23.01.2023).
6. *Blueprints в Unreal Engine 5.0* / [электронный ресурс] // Unreal Engine: сайт. — США, 2004—2023. — URL: https://docs.unrealengine.com/5.0/en-US/overview-of-blueprints-visual-scripting-in-unreal-engine/ (дата обращения: 23.01.2023).
7. *Программные и технические средства для Unreal Engine 5.0* / [электронный ресурс] // Unreal Engine: сайт. — США, 2004—2023. — URL: https://docs.unrealengine.com/5.0/en-US/hardware-and-software-specifications-for-unreal-engine/ (дата обращения: 23.01.2023).
8. *StroyDay.ru* : ведущий строительный портал России: сайт. / ООО «Интернет-Конверсия». — Кемерово, 2014—2023. — URL: https://stroyday.ru (дата обращения: 23.01.2023).
9. *Каскад-ГРУП* : офиц. сайт. / НПО «Каскад-ГРУП». — Чебоксары, 2007—2023. — URL: https://kaskad-asu.com (дата обращения: 23.01.2023).
10. *How to builds menus and UI in UE5 Beginner tutorial* : [видео] // Virtus Learning Hub : [канал пользователя YouTube]. — URL : https://www.youtube.com/watch?v=198AMGtdo-E (дата обращения 23.01.2023) – опубликовано: 8 июля 2022.
11. *Смирнова, М.В.* Разработка сцены интерактивного тренажера «Приточная вентиляционная установка» с использованием Unreal Engine 5 и Blender: курсовая работа / Смирнова М.В. — Москва, 2022. — 72 с.
12. *Арам К.* Разработка игр на Unreal Engine 4 : пер. с англ. / К. Арам, Д. Райан, К. Клинтон ; под ред. Р. Фасхутдинова — Москва : БОМБОРА : Эксмо , 2019 — 523с. — ISBN 978-5-04-103162-6

# СПИСОК СОКРАЩЕНИЙ

ЛКМ – левая кнопка мыши

ПКМ – правая кнопка мыши

ПК – персональный компьютер

# ПРИЛОЖЕНИЯ

## *Приложение А*

**Листинг кода программы**

|  |
| --- |
| #include "DPlayerController.h"  #include "../DBaseCharacter.h"  #include "../../UI/PlayerHUDWidget.h"  #include "Actors/Interactable/MechamismComponents/ComponentOfMechanism.h"  void ADPlayerController::SetPawn(APawn\* InPawn)  {  Super::SetPawn(InPawn);  CachedBaseCharacter = Cast<ADBaseCharacter>(InPawn);  PlayerController = UGameplayStatics::GetPlayerController(GetWorld(), 0);  if (IsLocalController() && CachedBaseCharacter.IsValid())  {  CreateAndInitializeWidgets();  CachedBaseCharacter->OnIntractableObjectFound.BindUObject(this, &ADPlayerController::OnInteractableObjectFound);  }  PlayerHUDWidget->SetHighLightInteractableVisibility(false);  }  void ADPlayerController::SetupInputComponent()  {  Super::SetupInputComponent();  InputComponent->BindAxis("MoveForward", this, &ADPlayerController::MoveForward);  InputComponent->BindAxis("MoveRight", this, &ADPlayerController::MoveRight);  InputComponent->BindAxis("Turn", this, &ADPlayerController::Turn);  InputComponent->BindAxis("LookUp", this, &ADPlayerController::LookUp);  InputComponent->BindAction("Jump", EInputEvent::IE\_Pressed, this, &ADPlayerController::Jump);  InputComponent->BindAction("ShowCursor", EInputEvent::IE\_Pressed, this, &ADPlayerController::ChangeMouseCursor);  InputComponent->BindAction("ActionInteract", EInputEvent::IE\_Pressed, this, &ADPlayerController::Interact);  InputComponent->BindAction("SetOpacity", EInputEvent::IE\_Pressed, this, &ADPlayerController::SetOpacity);  }  void ADPlayerController::OnInteractableObjectFound(FName ActionName)  {  if (!IsValid(PlayerHUDWidget))  {  GEngine->AddOnScreenDebugMessage(1, 2.0f, FColor::Red, FString(TEXT("PlayerHUD is not valid")), true);  return;  }  if (IsValid(PlayerHUDWidget) && CachedBaseCharacter.IsValid())  {  TArray<FInputActionKeyMapping> ActionKeys = PlayerInput->GetKeysForAction(ActionName);  const bool HasAnyKeys = ActionKeys.Num() != 0;  PlayerHUDWidget->SetHighLightInteractableVisibility(HasAnyKeys);  if (HasAnyKeys)  {  PlayerHUDWidget->SetHighLightInteractableActionText(FName(" "));  CachedBaseCharacter->SetHighlightObject();  }  else  {  PlayerHUDWidget->SetHighLightInteractableVisibility(false);  CachedBaseCharacter->RemoveHighlightObject();  }  }  }  void ADPlayerController::MoveForward(float Value)  {  if (CachedBaseCharacter.IsValid())  {  CachedBaseCharacter->MoveForward(Value);  }  }  void ADPlayerController::MoveRight(float Value)  {  if (CachedBaseCharacter.IsValid())  {  CachedBaseCharacter->MoveRight(Value);  }  }  void ADPlayerController::Turn(float Value)  {  if (CachedBaseCharacter.IsValid())  {  CachedBaseCharacter->Turn(Value);  }  }  void ADPlayerController::LookUp(float Value)  {  if (CachedBaseCharacter.IsValid())  {  CachedBaseCharacter->LookUp(Value);  }  }  void ADPlayerController::Jump()  {  if (CachedBaseCharacter.IsValid())  {  CachedBaseCharacter->Jump();  }  }  void ADPlayerController::ChangeMouseCursor()  {  if (PlayerController->bShowMouseCursor == false)  {  PlayerController->bShowMouseCursor = true;  PlayerController->SetInputMode(FInputModeGameAndUI());  //GEngine->AddOnScreenDebugMessage(1, 2.0f, FColor::Red, FString(TEXT("MC set to true")), true);  return;  }  else if (PlayerController->bShowMouseCursor == true)  {  PlayerController->bShowMouseCursor = false;  PlayerController->SetInputMode(FInputModeGameOnly());  //GEngine->AddOnScreenDebugMessage(1, 2.0f, FColor::Red, FString(TEXT("MC set to false")), true);  return;  }  }  void ADPlayerController::Interact()  {  if (CachedBaseCharacter.IsValid())  {  CachedBaseCharacter->Interact();  }  }  void ADPlayerController::SetOpacity()  {  if (CachedBaseCharacter.IsValid())  {  CachedBaseCharacter->SetOpacity();  }  }  void ADPlayerController::CreateAndInitializeWidgets()  {  if (!IsValid(PlayerHUDWidget))  {  PlayerHUDWidget = CreateWidget<UPlayerHUDWidget>(GetWorld(), PlayerHUDWidgetClass, FName("WBP\_PlayerHUD"));  if (IsValid(PlayerHUDWidget))  {  PlayerHUDWidget->AddToViewport();  }  }  }  #include "CoreMinimal.h"  #include "GameFramework/PlayerController.h"  #include "Kismet/GameplayStatics.h"  #include "../../UI/PlayerHUDWidget.h"  #include "DPlayerController.generated.h"  /\*\*  \*  \*/  UCLASS()  class DIPLOM\_API ADPlayerController : public APlayerController  {  GENERATED\_BODY()    public:  virtual void SetPawn(APawn\* InPawn) override;  //UPlayerHUDWidget\* GetPlayerHUDWidget();  UPlayerHUDWidget\* PlayerHUDWidget = nullptr;  protected:  virtual void SetupInputComponent() override;  UPROPERTY(EditDefaultsOnly, BlueprintReadOnly, Category = "Widgets")  TSubclassOf<class UPlayerHUDWidget> PlayerHUDWidgetClass;  UPROPERTY(EditDefaultsOnly, BlueprintReadOnly, Category = "Widgets")  TSubclassOf<class AComponentOfMechanism> CopmonentOfMechanismClass;  private:  void OnInteractableObjectFound(FName ActionName);  void MoveForward(float Value);  void MoveRight(float Value);  void Turn(float Value);  void LookUp(float Value);  void Jump();  void ChangeMouseCursor();  void Interact();  void SetOpacity();  void CreateAndInitializeWidgets();  APlayerController\* PlayerController;  AComponentOfMechanism\* ComponentsOfMechanism;  TSoftObjectPtr<class ADBaseCharacter> CachedBaseCharacter;  };  #include "DBaseCharacter.h"  #include "../Actors/Interactable/Interactive.h"  void ADBaseCharacter::EndPlay(const EEndPlayReason::Type Reason)  {  if (OnIntractableObjectFound.IsBound())  {  OnIntractableObjectFound.Unbind();  }  Super::EndPlay(Reason);  }  void ADBaseCharacter::Tick(float DeltaSeconds)  {  Super::Tick(DeltaSeconds);  TraceLineOfSight();  }  void ADBaseCharacter::Interact()  {  if (LineOfSightObject.GetInterface())  {  LineOfSightObject->Interact(this);  }  }  void ADBaseCharacter::SetOpacity()  {  if (LineOfSightObject.GetInterface())  {  LineOfSightObject->SetOpacity(this);  }  }  void ADBaseCharacter::SetHighlightObject()  {  if (LineOfSightObject.GetInterface())  {  LineOfSightObject->SetHighlightObject(this);  OldLineOfSightObject = LineOfSightObject;  }  }  void ADBaseCharacter::RemoveHighlightObject()  {  if (OldLineOfSightObject.GetInterface())  {  //GEngine->AddOnScreenDebugMessage(1, 2.0f, FColor::Red, FString(TEXT("RemoveEmmission")), false);  OldLineOfSightObject->RemoveHighlightObject(this);  OldLineOfSightObject = nullptr;  }  else  {  GEngine->AddOnScreenDebugMessage(1, 2.0f, FColor::Red, FString(TEXT("Old is not")), false);  }  }  #pragma once  #include "CoreMinimal.h"  #include "GameFramework/Character.h"  #include <UObject/ScriptInterface.h>  #include "DBaseCharacter.generated.h"  DECLARE\_DELEGATE\_OneParam(FOnIntractableObjectFound, FName);  class IInteractable;  UCLASS(Abstract, NotBlueprintable)  class DIPLOM\_API ADBaseCharacter : public ACharacter  {  GENERATED\_BODY()  public:  virtual void EndPlay(const EEndPlayReason::Type Reason) override;  virtual void Tick(float DeltaSeconds);  virtual void MoveForward(float Value) {};  virtual void MoveRight(float Value) {};  virtual void Turn(float Value) {};  virtual void LookUp(float Value) {};  virtual void ChangeMouseCursor() {};  virtual void Interact();  virtual void SetOpacity();  virtual void SetHighlightObject();  virtual void RemoveHighlightObject();  FOnIntractableObjectFound OnIntractableObjectFound;  protected:  UPROPERTY(EditDefaultsOnly, BlueprintReadOnly, Category = "Character | LineTrace")  float LineOfSightDistance = 300.0f;  void TraceLineOfSight();  UPROPERTY()  TScriptInterface<IInteractable> LineOfSightObject;  TScriptInterface<IInteractable> OldLineOfSightObject;  };  #include "PlayerCharacter.h"  #include "GameFramework/SpringArmComponent.h"  #include "Camera/CameraComponent.h"  #include "GameFramework/PlayerController.h"  APlayerCharacter::APlayerCharacter()  {  SpringArmComponent = CreateDefaultSubobject<USpringArmComponent>(TEXT("Spring arm"));  SpringArmComponent->SetupAttachment(RootComponent);  CameraComponent = CreateDefaultSubobject<UCameraComponent>(TEXT("Camera"));  CameraComponent->SetupAttachment(SpringArmComponent);  }  void APlayerCharacter::MoveForward(float Value)  {  if (!FMath::IsNearlyZero(Value, 1e-6f))  {  FRotator YawRotator(0.0f, GetControlRotation().Yaw, 0.0f);  FVector ForwardVector = YawRotator.RotateVector(FVector::ForwardVector);  //AddMovementInput(GetActorForwardVector(), Value);  AddMovementInput(ForwardVector, Value);  }  }  void APlayerCharacter::MoveRight(float Value)  {  if (!FMath::IsNearlyZero(Value, 1e-6f))  {  FRotator YawRotator(0.0f, GetControlRotation().Yaw, 0.0f);  FVector RightVector = YawRotator.RotateVector(FVector::RightVector);  //AddMovementInput(GetActorRightVector(), Value);  AddMovementInput(RightVector, Value);  }  }  void APlayerCharacter::Turn(float Value)  {  AddControllerYawInput(Value);  }  void APlayerCharacter::LookUp(float Value)  {  AddControllerPitchInput(Value);  }  #pragma once  #include "CoreMinimal.h"  #include "DBaseCharacter.h"  #include "GameFramework/Controller.h"  #include "PlayerCharacter.generated.h"  /\*\*  \*  \*/  UCLASS(Blueprintable)  class DIPLOM\_API APlayerCharacter : public ADBaseCharacter  {  GENERATED\_BODY()  public:  APlayerCharacter();  virtual void MoveForward(float Value) override;  virtual void MoveRight(float Value) override;  virtual void Turn(float Value) override;  virtual void LookUp(float Value) override;  protected:  UPROPERTY(VisibleAnywhere, BlueprintReadOnly, Category = "Character | Camera")  class UCameraComponent\* CameraComponent;  UPROPERTY(VisibleAnywhere, BlueprintReadOnly, Category = "Character | Camera")  class USpringArmComponent\* SpringArmComponent;    };  #include "UI/HighLightInteractable.h"  #include <Components/TextBlock.h>  void UHighLightInteractable::SetActionText(FName KeyName)  {  if (IsValid(ActionText))  {  ActionText->SetText(FText::FromName(KeyName));  }  }  #pragma once  #include "CoreMinimal.h"  #include "Blueprint/UserWidget.h"  #include <Components/TextBlock.h>  #include "HighLightInteractable.generated.h"  class UTextBlock;  UCLASS()  class DIPLOM\_API UHighLightInteractable : public UUserWidget  {  GENERATED\_BODY()    public:  void SetActionText(FName KeyName);  protected:  UPROPERTY(meta = (BindWidget))  UTextBlock\* ActionText;  };  #include "UI/PlayerHUDWidget.h"  #include "HighLightInteractable.h"  #include "Components/TextBlock.h"  void UPlayerHUDWidget::SetHighLightInteractableVisibility(bool bIsVisible)  {  if (!IsValid(InteractableKey))  {  FString errorClass = typeid(InteractableKey).name();  GEngine->AddOnScreenDebugMessage(1, 2.0f, FColor::Green, FString::Printf(TEXT("InteractableKey is not valid, %s"), \*errorClass, true));  return;  }  if (bIsVisible)  {  InteractableKey->SetVisibility(ESlateVisibility::Visible);  }  else  {  InteractableKey->SetVisibility(ESlateVisibility::Hidden);  }  }  void UPlayerHUDWidget::SetHighLightInteractableActionText(FName KeyName)  {  if (!IsValid(InteractableKey))  {  FString errorClass = typeid(InteractableKey).name();  GEngine->AddOnScreenDebugMessage(1, 2.0f, FColor::Green, FString::Printf(TEXT("InteractableKey is not valid, %s"), \*errorClass, true));  return;  }  InteractableKey->SetActionText(KeyName);  }  #pragma once  #include "CoreMinimal.h"  #include "Blueprint/UserWidget.h"  #include "Components/TextBlock.h"  #include "PlayerHUDWidget.generated.h"  UCLASS()  class DIPLOM\_API UPlayerHUDWidget : public UUserWidget  {  GENERATED\_BODY()  public:  void SetHighLightInteractableVisibility(bool bIsVisible);  void SetHighLightInteractableActionText(FName KeyName);  protected:  UPROPERTY(meta = (BindWidget))  class UHighLightInteractable\* InteractableKey = nullptr;  };  #include "UI/TeploControlUI.h"  #include "../Actors/Interactable/MechamismComponents/ComponentOfMechanism.h"  void UTeploControlUI::PlusTemp()  {  AComponentOfMechanism\* TeploPipePtr = Cast<AComponentOfMechanism>(TeploPipe);  if (IsValid(TeploPipePtr))  {  TeploPipePtr->PlusTemp();  }  else  {  GEngine->AddOnScreenDebugMessage(-1, 5.f, FColor::Red, TEXT("Cast failed"));  }  }  void UTeploControlUI::SubTemp()  {  AComponentOfMechanism\* TeploPipePtr = Cast<AComponentOfMechanism>(TeploPipe);  if (IsValid(TeploPipePtr))  {  TeploPipePtr->SubTemp();  }  else  {  GEngine->AddOnScreenDebugMessage(-1, 5.f, FColor::Red, TEXT("Cast failed"));  }  }  #pragma once  #include "CoreMinimal.h"  #include "Blueprint/UserWidget.h"  #include "../Actors/Interactable/MechamismComponents/ComponentOfMechanism.h"  #include "TeploControlUI.generated.h"  /\*\*  \*  \*/  UCLASS()  class DIPLOM\_API UTeploControlUI : public UUserWidget  {  GENERATED\_BODY()    public:  UFUNCTION(BlueprintCallable)  void PlusTemp();  UFUNCTION(BlueprintCallable)  void SubTemp();  protected:  UPROPERTY(EditAnywhere, BlueprintReadWrite, Category = "MaterialPipe")  AComponentOfMechanism\* TeploPipe;  }; |