МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

Учреждение образования «БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ

ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ»

**Проектирование информационных систем**

**Тема:** «Объектно-ориентированное моделирование. СТРУКТУРНЫЕ ДИАГРАММЫ UML»

**Цель:** Изучение методологии объектно-ориентированного моделирования средствами UML. Ознакомление с основными принципами объектно-ориентированного проектирования программного обеспечения, получение навыков проектирования структуры информационной системы с применением UML.

Выполнил:

Тышкевич Р.А.,

3 курс 7 группа

Проверил:

Якунович А. В.

Минск 2023

**Описание программных средств**

Draw.io — инструмент для создания диаграмм, блок-схем, интеллект-карт, бизнес-макетов, отношений сущностей, программных блоков и другого. Сервис распространяется на бесплатной основе с открытым исходным кодом. Draw.io обладает богатым набором функций для визуализации большинства задач пользователя.

Инструмент работает с Google Диск, Google Workspace и Dropbox, глубоко интегрирован и удобен для работы с продуктами Confluence и Jira от Atlassian. Пользователи также могут работать с диаграммами в автономном режиме и сохранять их локально, используя настольное приложение для персональных компьютеров.

Инструмент позволяет создавать: графики, диаграммы, таблицы, презентации, блок-схемы, планы помещений, воронки продаж, ментальные карты, карты сайтов.

**Постановка задачи**

Задачей данной работы выступает описание структуры взаимодействия классов в приложении, описанной в лабораторной работе #1.

Главными сущностями приложения выступают:

* Engine – Организует всю игру, управляя основными системами, игровыми циклами, временем, вводом/событиями, рендерингом, звуком, физикой и данными
* ResourceManager – Управляет всеми игровыми активами (текстуры, модели, звуки, скрипты), предоставляя к ним доступ и управляя их жизненным циклом.осуществляет обработку запросов, предназначенных предметам. А именно: добавление и удаление предмета.
* Entity – Представляет собой виртуальную среду, в которой происходит действие игры, включая местность, объекты, персонажей и их пространственные отношения.
* Character – Представляет действующих лиц в игре, наследуя от Entity и добавляя такие функции, как анимация, движение, здоровье, искусственный интеллект и возможности взаимодействия.
* GameplayManager – Контролирует общую логику игры, правила, подсчет очков, условия победы и различные игровые состояния.
* UIManager – Управляет элементами пользовательского интерфейса, такими как меню, HUD и уведомления, предоставляя игрокам обратную связь и информацию**.**

**Описание практического задания**

Разработанная диаграмма классов представлена на рисунке 1.1.

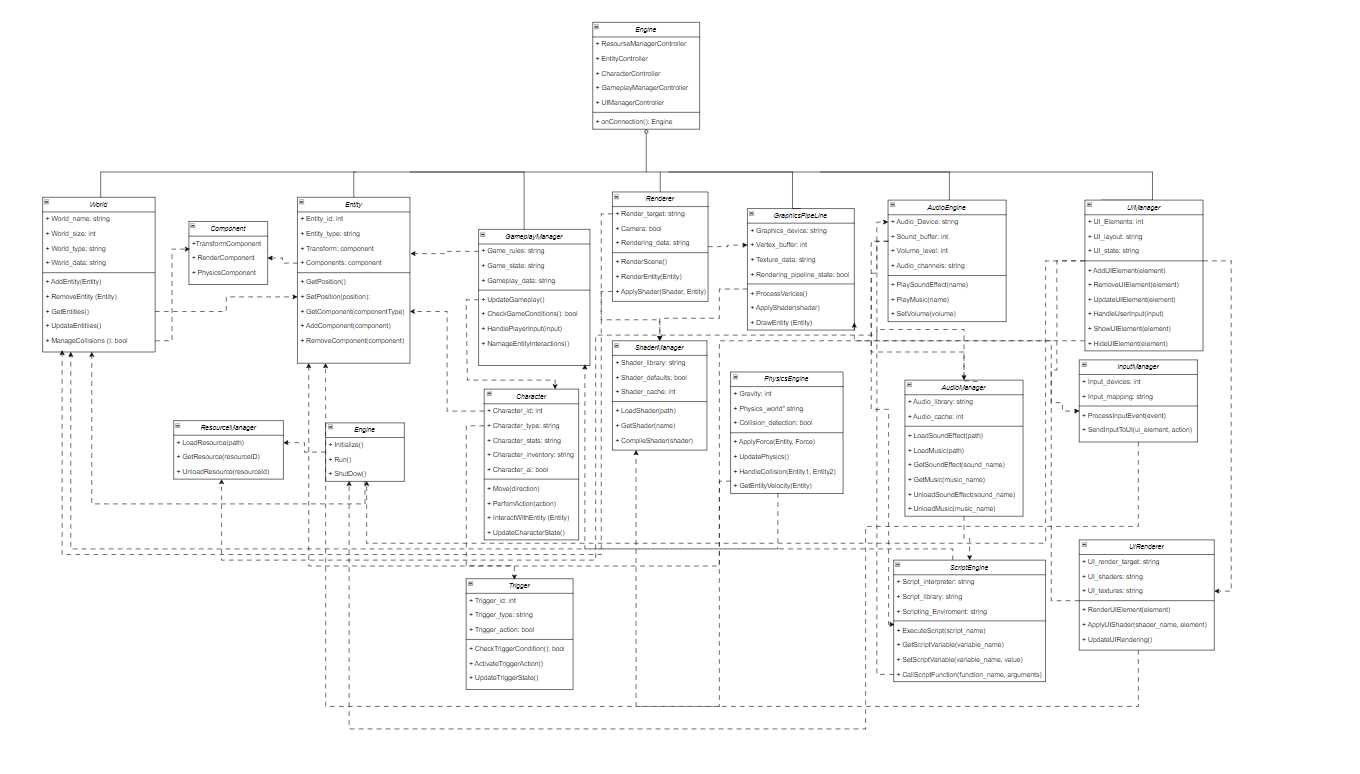
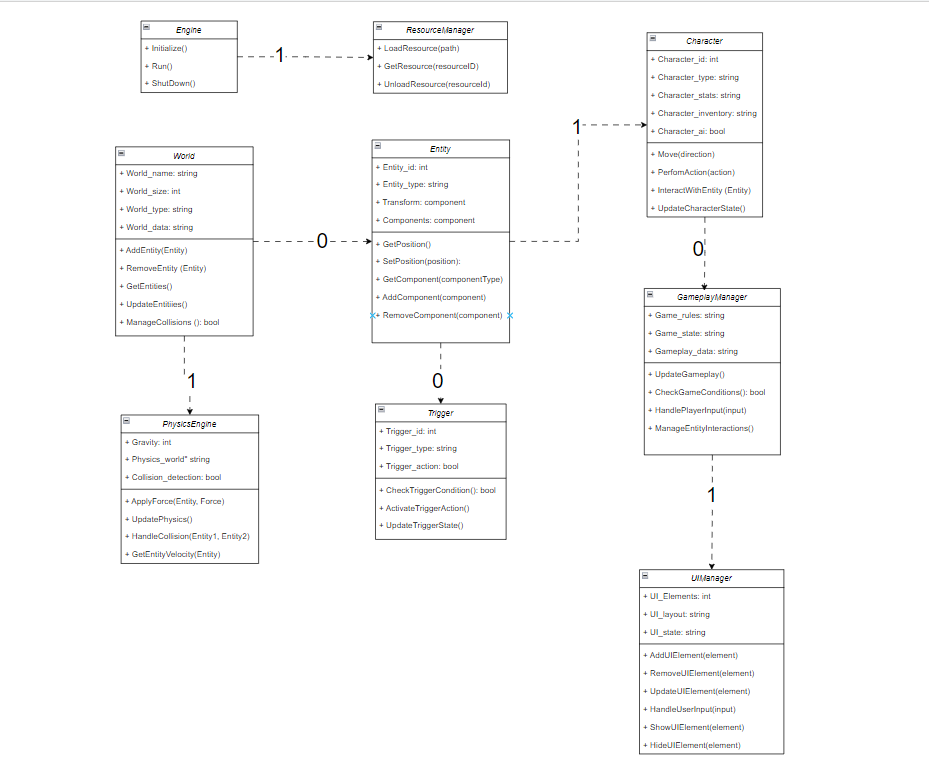


Рисунок 1.1 – Диаграмма классов приложения игрового движка

1. Между классами «Engine» (Движок) и «ResourceManager» (Менеджер ресурсов):
   * Кратность: 1\* (Один движок может иметь только один ResourceManager, а один ResourceManager может обслуживать только один движок)
2. Между классами «World» (Мир) и «Entity» (Сущности):
   * Кратность: 0\* (Один мир может содержать множество сущностей, и каждая сущность может существовать в одном мире)
3. Между классами «Entity» (Сущности) и «Character» (Персонаж):
   * Кратность: 1 (Одна сущность может быть персонажем или наследовать от него черты, но каждый персонаж является сущностью.)
4. Между классами «Character» (Персонаж) и «GameplayManager» (Геймплей Менеджер):
   * Кратность: 0\* (Один персонаж может взаимодействовать с GameplayManager, а GameplayManager может управлять многими персонажами.)
5. Между классами «GameplayManager» (Геймплей Менеджер) и «UIManager» (Менеджер UI):
   * Кратность: 1\* (один GameplayManager предоставляет информацию одному UIManager.)
6. Между классами «Entity» (Сущности) и «TriggerController» (Триггер-Контроллер):
   * Кратность: 0\* (дна сущность может вызывать события, обрабатываемые TriggerController, а TriggerController может управлять событиями для многих сущностей.)
7. Между классами «World» (Мир) и «PhysicsEngine» (Физ. Движок):
   * Кратность: 1\* (один мир взаимодействует с одним PhysicsEngine, и один PhysicsEngine моделирует физические правила для одного мира.)



Далее более подробно будут описаны каждый из модулей диаграммы.

1. ***Модуль «Engine Core»:***

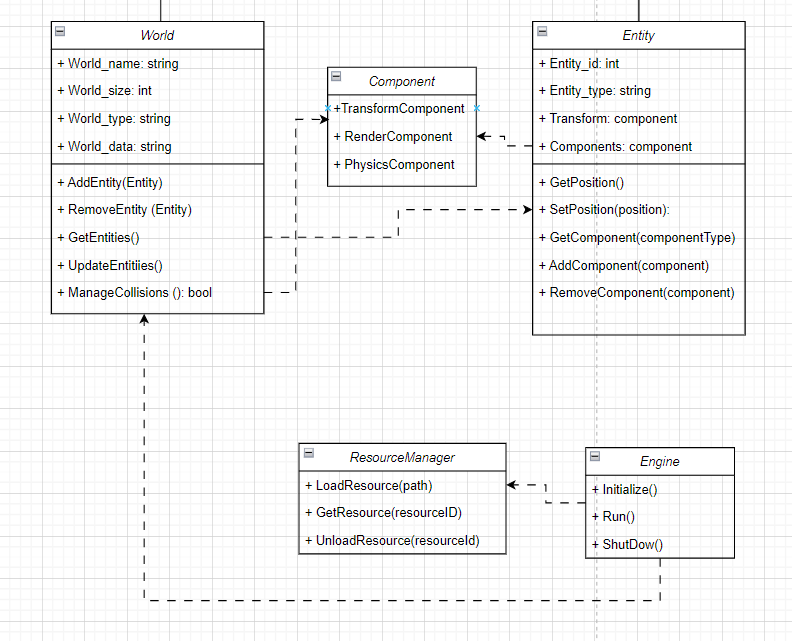


Рисунок 1.2 – Модуль «Engine Core»

World – управляет игровым миром и сущностями.

*Свойства:*

* + World\_name - Название мира
  + World\_size - Размер мира
  + World\_type - Тип мира
  + World\_data - Дополнительные данные, относящиеся к миру

*Методы:*

* + AddEntity - Добавляет сущность в мир.
  + RemoveEntity - Удаляет сущность из мира.
  + GetEntities - Получает список всех сущностей в мире.
  + UpdateEntities - Обновляет состояние всех сущностей в мире.
  + ManageCollisions - Обнаруживает и обрабатывает столкновения между сущностями.

Component – определяет поведение и функциональность сущностей.

*Методы:*

* + TransformComponent - Хранит и управляет положением, вращением и масштабом сущностей.
  + RenderComponent - Определяет визуальный вид сущности.
  + PhysicsComponent - Управляет физическим поведением сущности.

Entity – представляет актеров и объекты в мире.

*Свойства:*

* + Entity\_id - Уникальный идентификатор сущности.
  + Entuty\_type - Тип сущности
  + Transform - Компонент, содержащий положение, поворот и масштаб сущности.
  + Components - Список компонентов, присоединенных к сущности.

*Методы:*

* + GetPostion - Получает позицию сущности в мире.
  + SetPosition - Устанавливает позицию сущности в мире.
  + GetComponent - Получает компонент, прикрепленный к сущности.
  + AddComponent - Прикрепляет компонент к сущности.
  + RemoveComponent - Удаляет компонент из сущности.

ResourceManager – загружает и управляет игровыми активами.

*Методы:*

* + LoadResource - Загружает ресурс по указанному пути.
  + GetResource - Получает ресурс по его идентификатору.
  + UnloadResource - Выгружает ресурс из памяти.

Engine – инициализирует и управляет игровым циклом.

*Методы:*

* + Initialize - Инициализирует движок и все основные системы.
  + Run - Запускает основной цикл игры и обрабатывает обновления игры.
  + ShutDown - Очищает ресурсы и завершает работу движка.

1. ***Модуль «GamePlay & Entity»:***

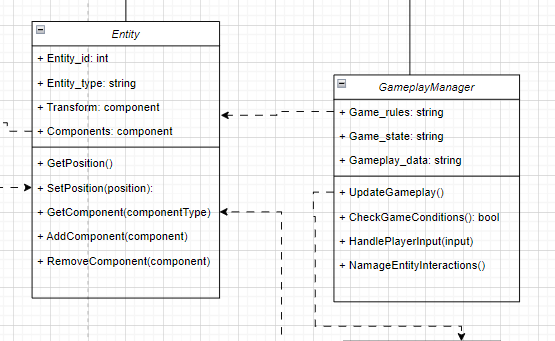


Рисунок 1.3 – Модуль «Gameplay & Entity»

Entity – представляет актеров и объекты в мире.

*Свойства:*

* + Entity\_id - Уникальный идентификатор сущности.
  + Entuty\_type - Тип сущности
  + Transform - Компонент, содержащий положение, поворот и масштаб сущности.
  + Components - Список компонентов, присоединенных к сущности.

*Методы:*

* + GetPostion - Получает позицию сущности в мире.
  + SetPosition - Устанавливает позицию сущности в мире.
  + GetComponent - Получает компонент, прикрепленный к сущности.
  + AddComponent - Прикрепляет компонент к сущности.
  + RemoveComponent - Удаляет компонент из сущности.

GameplayManager – контролирует игровую логику и правила.

*Свойства:*

* + Game\_rules - Определяет основные правила и условия игры.
  + Game\_state - Отслеживает текущее состояние игры
  + Gameplay\_data - Дополнительные данные, относящиеся к игре.

*Методы:*

* + UpdateGameplay - Обновляет состояние игры на основе действий игрока и взаимодействия с сущностями.
  + CheckGameConditions - Оценивает состояние игры и определяет условия победы/проигрыша.
  + HandlePlayerInput - Обрабатывает вводимые игроком данные и применяет их влияние на состояние игры.
  + ManageEntityInteractions - Обрабатывает взаимодействия между сущностями в соответствии с правилами игры.

1. ***Модуль «Character & Gameplay»:***

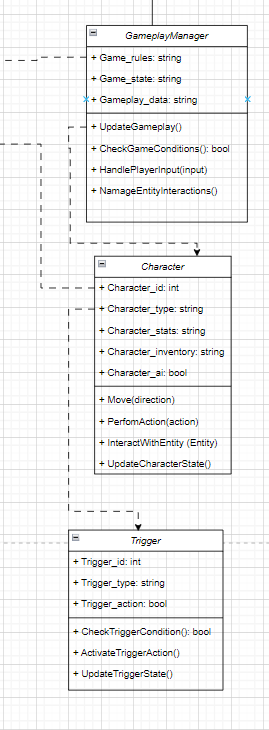


Рисунок 1.4 – Модуль «Character & Gameplay»

GameplayManager – контролирует игровую логику и правила.

*Свойства:*

* + Game\_rules - Определяет основные правила и условия игры.
  + Game\_state - Отслеживает текущее состояние игры
  + Gameplay\_data - Дополнительные данные, относящиеся к игре.

*Методы:*

* + UpdateGameplay - Обновляет состояние игры на основе действий игрока и взаимодействия с сущностями.
  + CheckGameConditions - Оценивает состояние игры и определяет условия победы/проигрыша.
  + HandlePlayerInput - Обрабатывает вводимые игроком данные и применяет их влияние на состояние игры.
  + ManageEntityInteractions - Обрабатывает взаимодействия между сущностями в соответствии с правилами игры.

Character – представляет игрока и NPC в игре.

*Свойства:*

* + Character\_id - Уникальный идентификатор персонажа.
  + Character\_type - Тип персонажа
  + Character\_stats - Атрибуты и статистика, определяющие возможности персонажа
  + Character\_inventory - Инвентарь предметов, которыми владеет персонаж.
  + Character\_ai - Сценарий или логика, управляющая поведением персонажа.

*Методы:*

* + Move - Перемещает персонажа в указанном направлении.
  + PerfomAction - Выполняет определенное действие, основанное на возможностях персонажа
  + InteractWithEntity - Инициирует взаимодействие с другой сущностью в мире.
  + UpdateCharacter - Обновляет состояние персонажа на основе игровых событий и действий.

Trigger – инициирует действия, основанные на событиях.

*Свойства:*

* + Trigger\_id - Уникальный идентификатор триггера.
  + Trigger\_type - Тип события триггера
  + Trigger\_action - Сценарий или логика, выполняемая при наступлении события триггера.

*Методы:*

* + CheckTrigger - Оценивает, выполнены ли условия триггера.
  + ActivateTrigger - Выполняет связанное действие при срабатывании триггера.
  + UpdateTrigger - Обновляет состояние триггера на основе игрового мира и взаимодействий сущностей.

1. ***Модуль «Rendering & Physics»:***

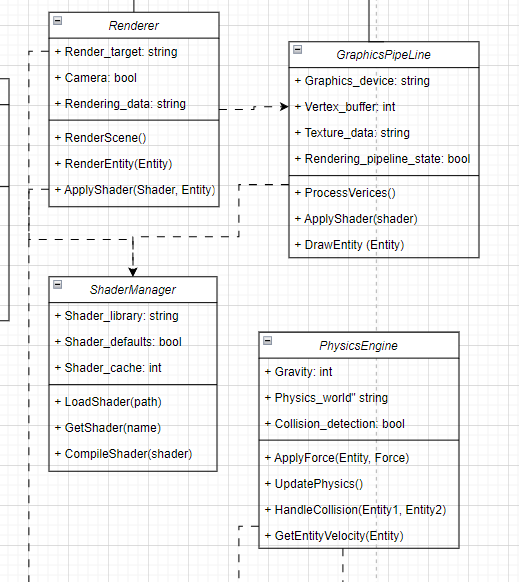


Рисунок 1.5 – Модуль «Rendering & Physics»

Renderer – рендерит визуальные эффекты игрового мира.

*Свойства:*

* + Render\_target - Поверхность, на которой будет отображаться конечное изображение.
  + Camera - Определяет перспективу просмотра и проекцию.
  + Rendering\_data - ополнительные данные, относящиеся к процессу рендеринга.

*Методы:*

* + RenderScene - Рендерит всю сцену, основываясь на данных о мире и настройках камеры.
  + RenderEntity - Рендерит конкретную сущность на основе ее визуальных компонентов.
  + ApplyShader - Применяет определенный шейдер к процессу рендеринга сущности.

ShaderManager – правляет шейдерами для визуальных эффектов.

*Свойства:*

* + Shader\_library - Коллекция шейдеров для различных материалов и эффектов.
  + Shader\_default - Шейдеры по умолчанию, используемые для общих задач рендеринга.
  + Shader\_cache - Хранит скомпилированные программы шейдеров для ускорения рендеринга.

*Методы:*

* + LoadShader - Загружает программу шейдера из указанного пути.
  + GetShader - Получает шейдерную программу по ее имени.
  + CompileShader - Компилирует шейдерную программу из исходного кода.

GraphicPipeLine – обрабатывает и отображает визуальные эффекты.

*Свойства:*

* + Graphics\_device - Интерфейс к базовому графическому оборудованию.
  + Vertex\_buffer - Хранит вершинные данные для сущностей.
  + Texture\_data - Хранит информацию о текстурах для материалов и объектов.
  + Rendering\_pipeline - Определяет последовательность операций, используемых для рендеринга.

*Методы:*

* + ProcessVerices - Преобразовывает и подготавливает вершинные данные для рендеринга.
  + ApplyShader - Использует определенную программу шейдера для рендеринга.
  + DrawEntity - Рендерит определенную сущность, используя ее вершинные и текстурные данные.

PhysicsEngine - моделирует поведение физики.

*Свойства:*

* + Gravity - Определяет гравитационную силу, воздействующую на объекты в мире.
  + Physics\_world - Внутренняя структура данных, представляющая физическую симуляцию.
  + Collision - Представляет данные о событиях столкновения между двумя объектами.

*Методы:*

* + ApplyForce - Применяет физическую силу к сущности
  + UpdatePhysics - Моделирует физическое поведение всех сущностей в мире.
  + HandleCollision - Обрабатывает столкновение между двумя сущностями.
  + GetEntityVelocity - Получает текущую скорость сущности.

1. ***Модуль «Audio & Scriptings»:***

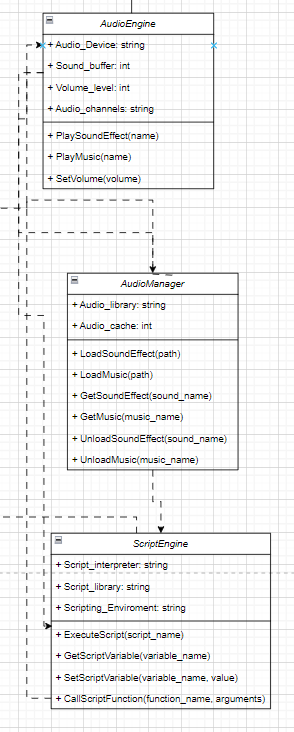


Рисунок 1.6 – Модуль «Audio & Scriptings»

AudioEngine - воспроизводит и микширует звуковые дорожки.

*Свойства:*

* + Audio\_Device - Интерфейс к базовому аудиооборудованию.
  + Sound\_buffer - Хранит аудиоданные для различных звуковых эффектов и музыкальных треков.
  + Volume\_level - Управляет общей громкостью звука.
  + Audio\_channels - Управляет несколькими аудиоканалами для одновременного воспроизведения.

*Методы:*

* + PlaySoundEffect - Воспроизводит определенный звуковой эффект.
  + PlayMusic - Воспроизводит фоновую музыкальную дорожку.
  + SetVolume - Устанавливает общую громкость звука.

AudioManager - управляет аудиоактивами.

*Свойства:*

* + Audio\_library - Коллекция загруженных звуковых эффектов и музыкальных дорожек.
  + Audio\_cache - Хранит аудиоданные в памяти для более быстрого доступа.

*Методы:*

* + LoadSoundEffect - Загружает звуковой эффект из указанного пути.
  + LoadMusic - Загружает музыкальную дорожку по указанному пути.
  + GetSoundEffect - Получает загруженный звуковой эффект по его имени.
  + GetMusic - Получает загруженную музыкальную дорожку по ее названию.
  + UnloadSoundEffect - Выгружает звуковой эффект из памяти.
  + UnloadMusic - Выгружает музыкальную дорожку из памяти.

ScriptEngine - выполняет скрипты для динамического поведения.

*Свойства:*

* + Script\_interpreter - Интерпретирует и выполняет код сценария.
  + Script\_library - Коллекция доступных скриптов и ресурсов.
  + Scripting\_Enviroment - Предоставляет переменные и функции, доступные в скриптах.

*Методы:*

* + ExecuteScript - Выполняет определенный скрипт с указанными аргументами.
  + GetScriptVariable - Получает значение переменной в текущем сценарии.
  + SetScriptVariable - Устанавливает значение переменной в текущем сценарии.
  + CallScriptFunction - Вызывает функцию, определенную в сценарии.

1. ***Модуль «User Interface»:***

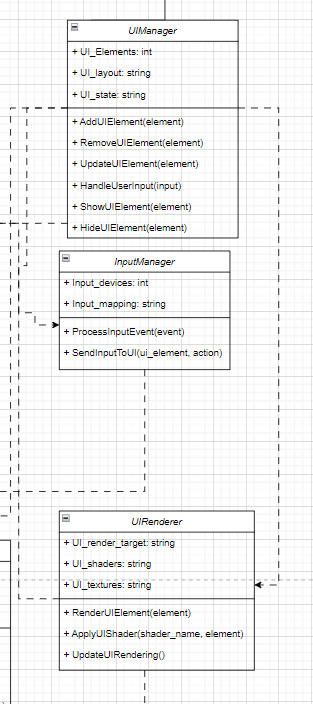


Рисунок 1.7 – Модуль «User Interface»

UIManager - управляет элементами пользовательского интерфейса.

*Свойства:*

* + UI\_Elements - Коллекция всех активных в данный момент элементов пользовательского интерфейса.
  + UI\_layout - Определяет расположение и организацию элементов пользовательского интерфейса.
  + UI\_state - Отслеживает текущее состояние и взаимодействие элементов пользовательского интерфейса.

*Методы:*

* + AddUIElement - Добавляет новый элемент пользовательского интерфейса в интерфейс.
  + RemoveUIElement - Удаляет элемент пользовательского интерфейса из интерфейса.
  + UpdateUIElement - Обновляет состояние и внешний вид элемента пользовательского интерфейса.
  + HandleUIElement - Обрабатывает пользовательский ввод и применяет его к соответствующим элементам пользовательского интерфейса.
  + ShowUIElement - Делает определенный элемент пользовательского интерфейса видимым.
  + HideUIElemet - Скрывает определенный элемент пользовательского интерфейса.

InputManager - обрабатывает пользовательский ввод для взаимодействия с пользовательским интерфейсом.

*Свойства:*

* + Input\_devices - Список активных устройств ввода
  + Input\_mapping - Определяет отображение между событиями ввода и действиями пользовательского интерфейса.

*Методы:*

* + ProcessInputEvent - Анализирует событие ввода и определяет соответствующее действие пользовательского интерфейса.
  + SendInputToUI - Передает действие ввода в соответствующий элемент пользовательского интерфейса.

UIRenderer - рендеринг пользовательского интерфейса.

*Свойства:*

* + UI\_render\_target - Определяет поверхность, на которой будут отрисовываться элементы пользовательского интерфейса.
  + UI\_shaders - Коллекция шейдеров, используемых для рендеринга различных элементов пользовательского интерфейса.
  + UI\_textures - Коллекция текстур, используемых для графических элементов пользовательского интерфейса.

*Методы:*

* + RenderUIElement - Рендерит определенный элемент пользовательского интерфейса на экране.
  + ApplyUIShader - Применяет определенный шейдер к элементу пользовательского интерфейса.
  + UpdateUIRendering - Обновляет состояние отображения всех элементов пользовательского интерфейса на основе их последних данных.

Далее представлены пакеты, в которые были сгруппированы классы.

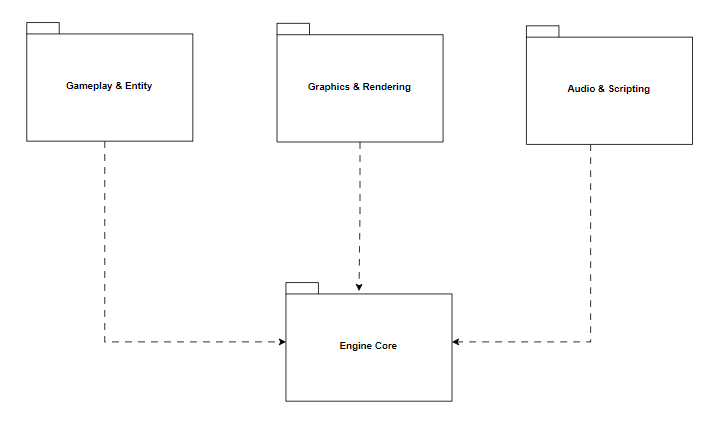


Рисунок 1.8 – Пакеты классов

Engine Core эти классы составляют основу игрового движка и необходимы для его базового функционирования. Они не зависят от конкретного игрового контента и служат основой для создания дополнительных возможностей. К ним относятся: Engine, ResourceManager, World, Entity.

Gameplay & Entity эти классы сосредоточены на определении игровой механики и поведения сущностей в игре. Они тесно взаимодействуют с ядром движка и друг с другом, создавая основной геймплей игры. К этим классам относятся: GameplayManager, Character, Trigger.

Graphics & Rendering эти классы отвечают за создание визуального представления игрового мира и сущностей. Они взаимодействуют с Engine Core и Gameplay & Entity для преобразования игрового состояния и действий в визуальные образы. К этим классам относятся: Renderer, ShaderManager, GraphicsPipeLine.

Audio & Scripting эти классы добавляют дополнительные уровни взаимодействия и погружения в игру. Они взаимодействуют с ядром движка и другими пакетами для улучшения звука, скриптовых возможностей и пользовательского интерфейса игры. К данным классам относится AudioEngine, AudioManager, ScriptEngine, UIManager, InputManager, UIRenderer.

**Вывод:** Я изучил методологию объектно-ориентированного моделирования средствами UML. Ознакомился с основными принципами объектно-ориентированного проектирования программного обеспечения, получил навыки проектирования функциональности информационной системы с применением UML.