1. **Архитектура игрового приложения**

**2.1 Исходные данные игрового приложения**

Необходимо разработать приложение, реализующее многопользовательскую игру «Сражение подводных лодок». Каждый игрок управляет подводной лодкой, которая может стрелять различными видами торпед. Каждая лодка характеризуется скоростью плавания, бронёй и жизненной энергией. Перед началом игры каждый игрок на одинаковое количество очков может сконфигурировать свою подводную лодку: выбрать максимальное значение скорости, максимальное значение брони, жизненной энергии, количество торпед, одновременно находящихся на борту. Перед началом игры на каждую подводную лодку может быть загружено определённое число

торпед различных типов. Торпеды отличаются скоростью перемещения, повреждающим эффектом, скорострельностью. Во время игры на экране случайным образом появляются призы, позволяющие пополнить боезапас, броню, жизненную энергию, увеличить/уменьшить на определённое число процентов скорость. Кроме того, периодически должен появляться эсминец и сбрасывать подводные мины, столкновение с которыми наносить повреждения лодкам. Мины уничтожаются торпедами, либо достигая дна. Проигрывает тот игрок, чья подводная лодка погибает. Для реализации использовать

средства языка программирования C# Windows form и шаблоны проектирования: «фабричный метод» – для реализации генерации призов: «декоратор» – для задания характеристик подводных лодок. Для отображения объектов на игровом поле использовать спрайтовую графику и средства OpenGL.

* 1. **Анализ игрового приложения «Сражение подводных лодок»**

Необходимый функционал игрового приложения:

* Удобное меню игры;
* Возможность редактирование характеристик лодки в начале игры;
* Многопользовательское управление;
* Возможность ведения боя между двумя игроками;
* Получение бонусов во время игры;
* Вывод итогов игры.

Для реализации игры будем использовать собственный игровой движок.

Основные задачи игрового движка:

1. Работа со спрайтовой графикой;
2. Работа с 2D анимацией;
3. Физика перемещение и взаимодействие игровых объектов;
4. Реализация и интегрирование в игру паттерна «декоратор»;
5. Реализация и интегрирование в игру паттерна «фабричный метод».

**2.3 Структура игрового приложения**

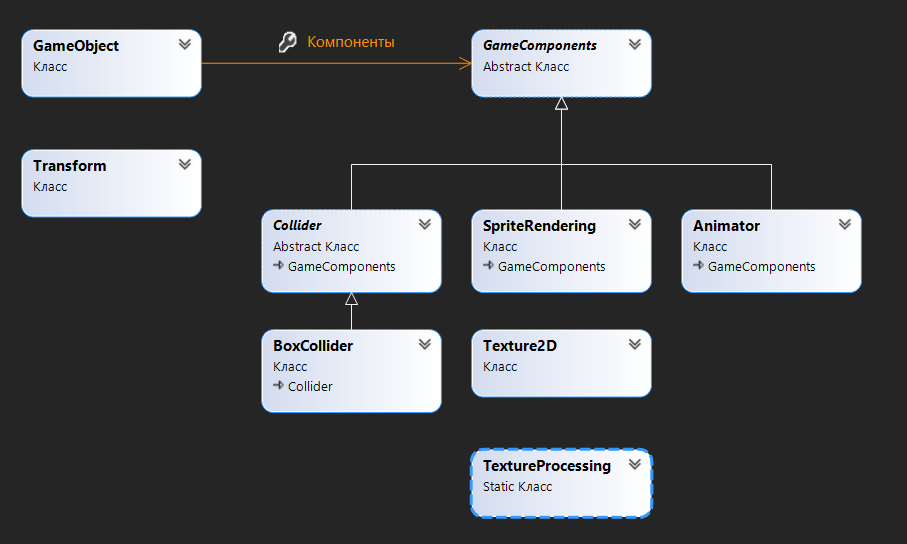
Рассмотрим схему классов нашего игрового движка.

Рисунок 2.1 – UML схема игрового движка

Подробно рассмотрим классы игрового движка.

Одним из главных классов игрового приложения является класс нашей игровой сцены, который наследует класс GameWindow из графической библиотеки OpenTK. В классе нашей сцене мы создаем окно OpenTK и переопределяем самые необходимые методы. Данный класс не указан в игровом движке, так как он является сценой для игровых объектов, но при этом использует средства библиотеки OpenTK.

Основные методы класса сцены:

1. OnRenderFrame – данный метод вызывается при рендеринге кадра и позволяет производить какие-либо графические изменения над нашими игровыми объектами.
2. OnUpdateFrame – данный метод вызывается при обновлении кадра и используется для реализации игровой логики и изменение игровых объектов.
3. OnLoad – данный метод вызывается после установления контекста OpenGL, но до входа в основной цикл игрового приложения.

Классы игрового движка:

1. GameObject является абстрактным базовым типом для всех игровых объектов. GameObject содержит в себе список компонентов игрового объекта (GameComponent).
2. GameComponent является базовым типом для всех компонентов игры, он позволяет добавлять игровым объектам различные свойства и функционал.

Возможные компоненты:

* Класс SpriteRenderer является производным от GameComponent и отвечает за отображение объекта на сцене;
* Класс Collider является производным от GameComponent и содержит в себе коллайдер игрового объекта
* Класс Animator является производным от GameComponent и отвечает за анимацию игровых объектов.

1. Класс Transform содержится в игровом объекте GameObject как обязательный параметр, так как все объекты должны иметь положение, вращение и масштаб.

Transform содержит такие поля как:

* Position – положение игрового объекта в координатах x, y;
* Rotation – вращение игрового объекта вокруг осей x, y;
* Scale – масштаб игрового объекта.

1. Класс Collider является абстрактным базовым типом для коллайдеров.
2. Класс BoxCollider является наследником класса Collider и описывает коллайдеры, которые имеют прямоугольную форму.
3. Класс Texture2D хранит в себе номер в буфере и размеры добавленной текстуры.
4. Статический класс TextuteProcessing создаёт текстуру на основы добавленного изображения и выводит класс Texture2D со всеми её свойствами.

Теперь рассмотрим основные объекты нашей игры.

Игровые объекты:

1. Подводная лодка, которой будут управлять игроки.
2. Различного вида ракеты, которыми будет стрелять подводная лодка. В зависимости от вида наносит различные повреждения.
3. Различного вида призы, каждый приз будет увеличивать различные свойства подводной лодки.
4. Миноносец который будет периодически появляться и сбрасывать на игроков мины.

Для реализации классов танков и ракет используется наследование. В случае с подводной лодкой (Submarine) мы не используем наследование, так как подводные лодки не отличаются характеристиками, по желанию только спрайтами. Для ракет мы используем наследование, так как ракеты отличаются наносимыми повреждениями в зависимости от подтипа ракеты.

На рисунке 2.2 отображена иерархия игровых объектов.

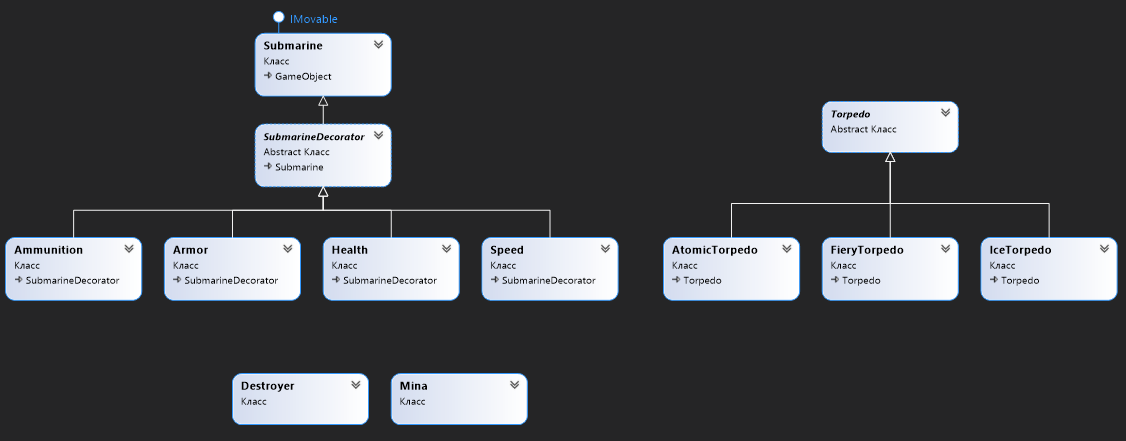


Рисунок 2.2 – Иерархия игровых объектов

Рассмотри паттерны программирования, которые будут использоваться в нашем игровом приложении.

Паттерн Декоратор (Decorator) – представляет структурный шаблон проектирования, который позволяет динамически подключать к объекту дополнительную функциональность.

Данный паттерн следует использовать, когда объекту нужно добавлять новый функционал. Отличительной особенностью данного паттерна от обычного наследования является возможность убрать у объекта добавленный функционал, прямо во время работы программы. Т.е. декоратор позволяет снимать декорации, но снятие декораций должно происходить в порядке, обратном наложению.

На рисунке 2.3 представлена UML схема реализации паттерна «декоратор» для поставленной задачи.

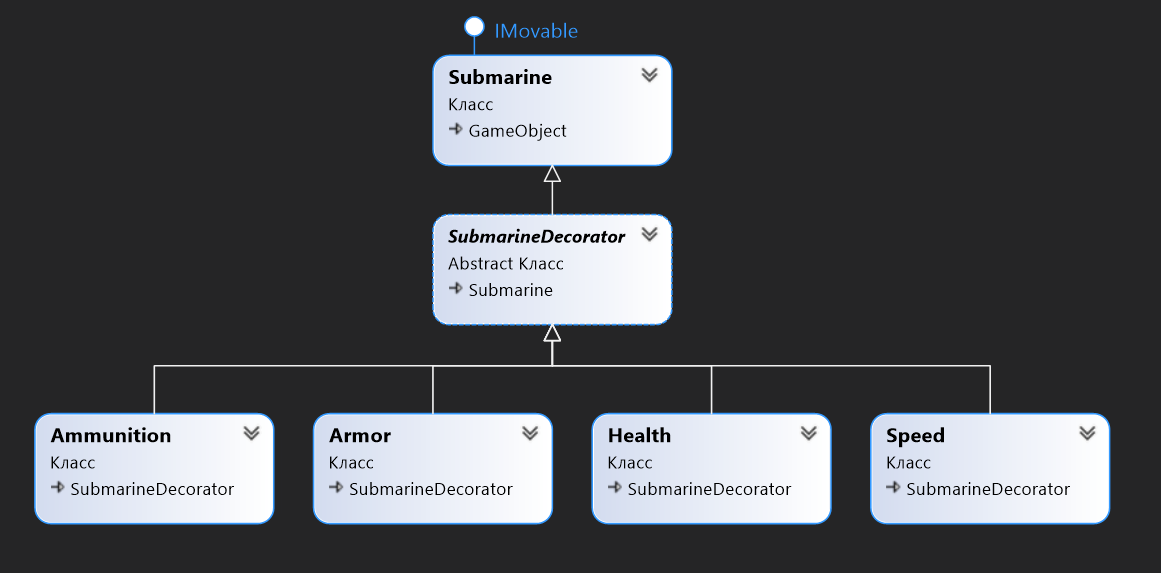


Рисунок 2.3 – UML схема паттерна «Декоратор»

Описание классов участвующих в реализации паттерна:

1. Submarine: данный класс представляет подводную лодку. Часто классы которые определяют интерфейс для наследуемых объектов, являются абстрактными. Применение декораций возможно, так как Submarine содержит виртуальные методы декорируемых свойств. В данном случае это не так, так как подводные лодки не отличаются характеристиками, по желанию только спрайтами.
2. SubmarineDecorator: сам декоратор, реализуется посредством абстрактного класса и имеет тот же базовый класс, что и декорируемые объекты. Наследники данного класса представляют дополнительные функциональности, которыми должен быть расширен объект Submarine.
3. Ammunition: дополнительный функционал для класса Submarine в виде пополнения боеприпасов.
4. Armor: дополнительный функционал для класса Submarine в виде увеличения брони.
5. Health: дополнительный функционал для класса Submarine в виде пополнения здоровья.
6. Speed: дополнительный функционал для класса Submarine в виде увеличения скорости.

Фабричный метод (Factory method) – это паттерн, который определяет интерфейс для создания объектов некоторого класса, но непосредственное решение о том, объект какого класса создавать происходит в подклассах. То есть паттерн предполагает, что базовый класс делегирует создание объектов классам-наследникам.

В каких случаях необходимо применять данный паттерн:

* когда система должна быть независимой от процесса создания новых объектов и расширяемой: в нее можно легко вводить новые классы, объекты которых система должна создавать;
* когда заранее неизвестно, объекты каких типов необходимо создавать;
* когда создание новых объектов необходимо делегировать из базового класса классам наследникам.

На рисунке 2.4 представлена UML схема реализации паттерна «Фабричный метод» для поставленной задачи.

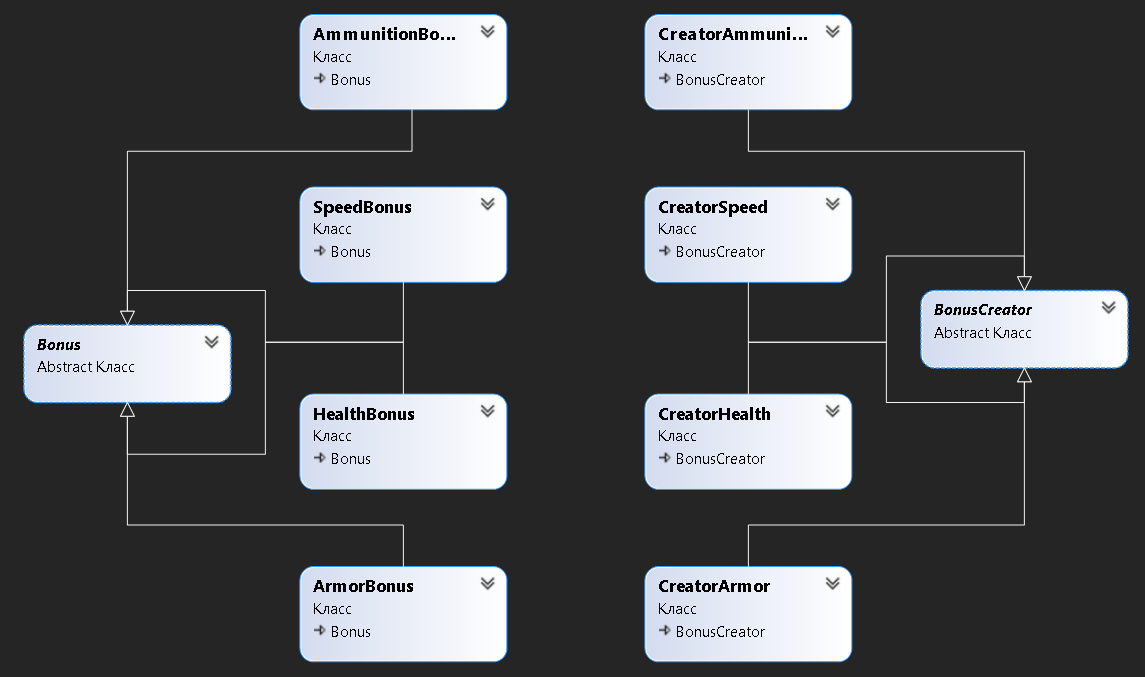


Рисунок 2.4 – UML схема паттерна «Фабричный метод»

Описание классов участвующих в реализации паттерна:

1. Bonus: абстрактный класс определяющий интерфейс класса, объекты которого необходимо генерировать;
2. AmminitionBonus: конкретный класс представляющий реализацию класса Bonus. Класс реализует пополнение боеприпасов;
3. SpeedBonus: конкретный класс представляющий реализацию класса Bonus. Класс реализует увеличение скорости;
4. HealthBonus: конкретный класс представляющий реализацию класса Bonus. Класс реализует пополнение здоровья;
5. ArmorBonus: конкретный класс представляющий реализацию класса Bonus. Класс реализует пополнение брони;
6. BonusCreator: абстрактный класс определяющий абстрактный фабричный метод CreateBonus, который возвращает объект Bonus. Наследники данного класса определяют свою реализацию метода CreatorBonus. Метод CreateBonus каждого отдельного класса-создателя возвращает определенный конкретный тип бонуса.
7. CreatorSpeed: возвращает класс с бонусом скорости.
8. CreatorHealth: возвращает класс с бонусом здоровья.
9. CreatorArmor: возвращает класс с бонусом брони.
10. CreatorAmmunition: возвращает класс с бонусом боеприпасов.