МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра МО ЭВМ

ОТЧЕТ

по лабораторной работе №5

по дисциплине «Объектно ориентированное программирование»

Тема: Шаблонные классы, генерация карты

Студент гр. 1304 Шаврин А.П.

Преподаватель Жангиров Т.Р.

Санкт-Петербург
2022

Цель работы.

Изучить шаблонные классы с помощью ООП в языке C++. Применить полученные знания в генерации карты в игре. Научиться выстраивать архитектуру проекта с возможностью дальнейшего расширения без изменения исходного кода.

Задание.

Реализовать шаблонный класс генерирующий игровое поле. Данный класс должен параметризироваться правилами генерации (расстановка непроходимых клеток, как и в каком количестве размещаются события, расположение стартовой позиции игрока и выхода, условия победы, и.т.д.). Также реализовать набор шаблонных правил (например, событие встречи с врагом размещается случайно в заданном в шаблоне параметре, отвечающим за количество событий)

Требования:

Реализован шаблонный класс генератор поля. Данный класс должен поддерживать любое количество правил, то есть должен быть variadic template.

Класс генератор создает поле, а не принимает его.

Класс генератор не должен принимать объекты классов правил в каком-либо методе, а должен сам создавать (в зависимости от реализации) объекты правил из шаблона.

Реализовано не менее 6 шаблонных классов правил

Классы правила должны быть независимыми и не иметь общего класса-интерфейса

При запуске программы есть возможность выбрать уровень (не менее 2) из заранее заготовленных шаблонов

Классы правила не должны быть только “хранилищем” для данных.

Так как используются шаблонные классы, то в генераторе не должны быть dynamic_cast

Примечания:

Для задания способа генерации можно использовать стратегию, компоновщик, прототип

Не рекомендуется делать static методы в классах правилах

Выполнение работы.

1. Сначала созданы перечисление *Levels* всех возможных уровней и *Events* всех событий.

2. После реализованы шаблонные правила:

- *SpawnPlayerRule1<int pos_x, int pos_y>*

Данный класс через шаблонные параметры *pos_x* и *pos_y* получает координаты игрока, куда его необходимо установить. Метод *apply* получает указатель на поле и устанавливает игрока по заданным координатам, если на них нет врага или стенки. Если по данным координатам игрока установить нельзя, то выбираются новые координаты.

- *SpawnPlayerRule2<int core>*

Данный класс через шаблонный параметр *core* получает ядро, которое будет передано “рандомайзеру” позиции игрока. Метод *apply* получает указатель на поле и устанавливает игрока по сгенерированным координатам, если на них нет врага или стенки. Если по данным координатам игрока установить нельзя, то генерируются новые координаты. Если входные данные будут плохие, в методе *apply* они преобразуются в допустимые для игры.

- *SpawnWallsRule1<int magic_number>*

Данный класс через шаблонный параметр *magic_number* получает число, влияющее на расстановку непроходимых клеток. Метод *apply* получает указатель на поле и устанавливает непроходимые клетки по заданному правилу. Данное правило расстанавливает стенки вертикальными полосами на поле, по четным или нечетным *x* (в зависимости от параметра *magic_number*). Также в данном методе перед установкой происходит проверка на возможность сделать это. Если входные данные будут плохие, в методе *apply* они преобразуются в допустимые для игры.

- *SpawnWallsRule2<int magic_number>*

Данный класс через шаблонный параметр *magic_number* получает число, влияющее на расстановку непроходимых клеток. Метод *apply* получает указатель на поле и устанавливает непроходимые клетки по заданному правилу. Данное правило расставляет стенки по кругу, через одну клетку (*magic_number* влияет на четность или нечетность). Также в данном методе перед установкой происходит проверка на возможность сделать это. Если входные данные будут плохие, в методе *apply* они преобразуются в допустимые для игры.

- *SpawnEnemiesRule1<int core, int precent>*

Данный класс через шаблонные параметры *core* и *precent* получает ядро, которое будет передано “рандомайзеру” позиции врагов и процент врагов от размера поля. Метод *apply* получает указатель на поле и устанавливает врагов по заданному правилу (используя *random*) и в нужном количестве. Также в данном методе перед установкой происходит проверка на возможность сделать это. Если входные данные будут плохие, в методе *apply* они преобразуются в допустимые для игры.

- *SpawnEnemiesRule2<int magic_number, int count>*

Данный класс через шаблонные параметры *magic_number* и *count* получает число, которое влияет на выбор позиции врага и количество врагов. Метод *apply* получает указатель на поле и устанавливает врагов по заданному правилу (собственный способ выбора псевдослучайной позиции) и в нужном количестве. Также в данном методе перед установкой происходит проверка на возможность сделать это. Если входные данные будут плохие, в методе *apply* они преобразуются в допустимые для игры.

- S*pawnEventsRule1<int core, int precent, Events type>*

Данный класс через шаблонные параметры *core*, *precent* и *type* получает ядро, которое будет передано “рандомайзеру” позиции события, процент событий от размера поля и название генерируемого события из перечисления всех событий *Events*. Метод *apply* получает указатель на поле и устанавливает события по заданному правилу (используя *random*) и в нужном количестве.

Также в данном методе перед установкой происходит проверка на возможность сделать это. Если входные данные будут плохие, в методе *apply* они преобразуются в допустимые для игры.

- *SpawnEventsRule2< int magic_number, int count, Events type>*

Данный класс через шаблонные параметры *magic_number*, *count* и *type* получает число, которое влияет на выбор позиции события, количество событий и название генерируемого события из перечисления всех событий *Events*. Метод *apply* получает указатель на поле и устанавливает события по заданному правилу (собственный способ выбора псевдослучайной позиции) и в нужном количестве. Также в данном методе перед установкой происходит проверка на возможность сделать это. Если входные данные будут плохие, в методе *apply* они преобразуются в допустимые для игры.

3. После того, как реализованы все 8 правил, написан шаблонный класс *LevelGenerator*, который параметризуется правилами генерации поля. Шаблонный метод *applyer*, параметризуется конкретным правилом, создает обьект этого правила и вызывает у него метод *apply*, передавая в него указатель на карту. Метод *generate* для каждого правила вызывает шаблонный метод *applyer*. Метод *createMap* принимает необходимые для создания карты аргументы и создает карту.

4. Потом реализован класс-интерфейс *LevelStrategy*, имеющий чисто виртуальный метод *generateLevel*, возвращающий указатель на карту и виртуальный деструктор (паттерн стратегия).

5. После реализованы 2 конкретные стратегии, унаследованные от *LevelStrategy*:

- *LevelOne*
- *LevelTwo*

Обе стратегии имеют приватные поля *Map** и *LevelGenerator* (отличается шаблонными параметрами), в конструкторе принимают указатель на *GameController*, *Player*, *Mediator<Log*>*, а также *int map_height* и *int map_width*. В конструкторе создается поле при помощи *level_generator*, с учетом переданных параметров. В методе *generateLevel* оба класса вызывают метод собственного *level_generator.generate* и возвращают указатель на сгенерированную, с учетом правил, карту (паттерн стратегия).

6. Затем написан класс *LevelContext*, имеющий единственное приватное поле *LevelStrategy**. Метод setStrategy устанавливает стратегию, а метод *createLevel* генерирует карту по заданной стратегии и возвращает ее (паттерн стратегия).

7. После написан класс *StartLevelDialog*, который имеет 3 приватных поля: *int map_height, int map_width, Levels level*. В методе *userDialog* происходит диалог с пользователем и выбор уровня. Исходя из выбранного уровня устанавливаются размеры поля и *level*.

Зависимости классов приведены на UML диаграмме ниже:

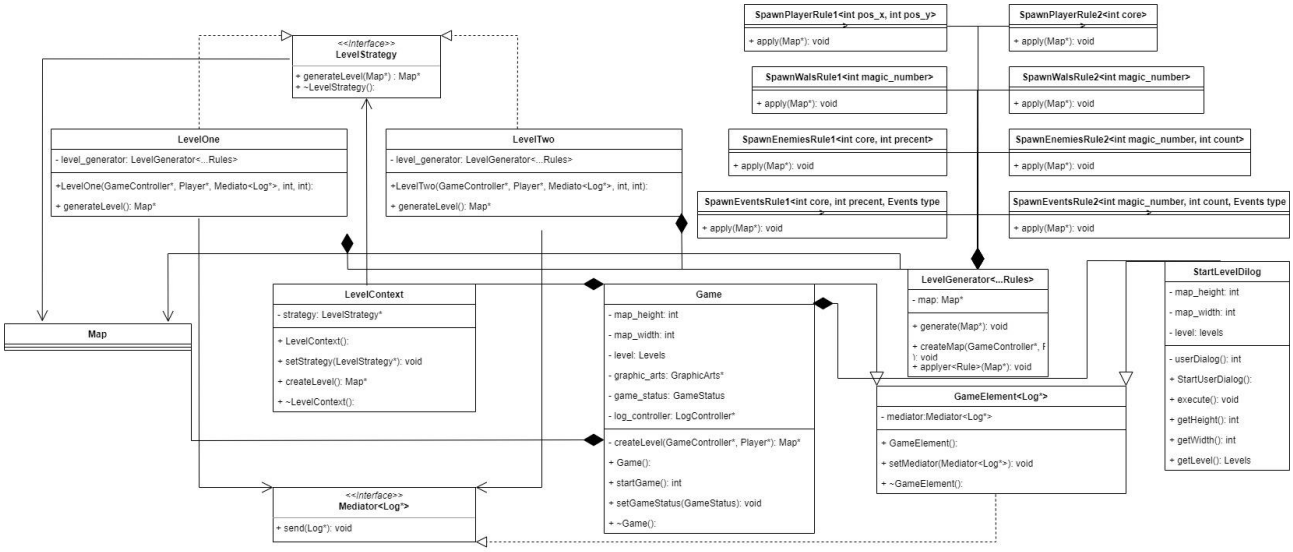

Рисунок 1 - UML диаграмма классов.

Выводы.

Изучены шаблонные классы с помощью ООП в языке C++. Применены полученные знания в генерации карты в игре. Научился выстраивать архитектуру проекта с возможностью дальнейшего расширения без изменения исходного кода. А также изучен и применен на практике поведенческий паттерн стратегия.