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

ОТЧЕТ

по лабораторной работе №5
по дисциплине «ООП»
Тема: Шаблонные классы, генерация карты

Студентка гр. 1383 Седова Э.А

Преподаватель Жангиров Т.Р.

Санкт-Петербург
2022

Цель работы.

Реализовать шаблонный класс генерирующий игровое поле.

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 правила

Выполнение работы.

Создан шаблонный класс *FieldGenerate*. Его шаблоном является неограниченное количество правил. Класс содержит единственный метод generate(), принимающий на вход пакет параметров – шаблонов правил. Данный метод создаёт поле, применяет к нему классы – правила и возвращает указатель на поле.

DefaultRule – класс-правило, задающий позицию игрока, ключа или выхода. Для определения, какое именно событие нужно расставить в клетки через шаблон передаётся число. За событии *Key()* отвечает 4, за *Level_exit*() – 5, за игрока - 6. Координаты позиции задаются через шаблон. Имеет единственный метод *fill(),* принимающий ссылку на поле.

PoseRule – класс-правило, задающий случайную позицию игрока. В единственном методе *fill(),* принимающем ссылку на поле, совершается обход по полю. Если *rand()%x*, где x – случайное число, задающееся через шаблон, равен определённому числу и на данной клетке нет события *Key()* и Level_exit(), то в неё “ставится” игрок.

KeyRule – класс-правило, задающий случайную позицию ключа. В единственном методе *fill(),* принимающем ссылку на поле, совершается обход по полю. Если *rand()%x*, где x – случайное число, задающееся через шаблон, равен определённому числу и на данной клетке не стоит игрок и нет события *Level_exit(),* то в неё помещается событие *Key().* Если в результате обхода ключ так и не был поставлен на поле, то он помещается в клетку с заранее определёнными координатами.

ExitRule – класс-правило, задающий случайную позицию выхода. В единственном методе *fill(),* принимающем ссылку на поле, совершается обход по полю. Если *rand()%x*, где x – случайное число, задающееся через шаблон, равен определённому числу и на данной клетке не стоит игрок и нет события *Key(),* то в неё помещается событие *Level_exit().* Если в результате обхода

выход так и не был поставлен на поле, то он помещается в клетку с заранее определёнными координатами.

EventRule – класс-правило, задающий случайные позиции событий. Для определения, какое именно событие нужно расставить в клетки через шаблон передаётся число. За событии *Wall()* отвечает 0, за *fire*() – 1, за *Heal()* – 2. В единственном методе *fill(),* принимающем ссылку на поле, реализован цикл *while,* принимающий параметр *counter*, задающийся через шаблон. Данный параметр – это максимальное возможное количество клеток с событием. События записываются в случайные клетки (если в них нет игрока и событий *Key()* и *Level_exit()*) пока цикл не кончится.

DamageRule – класс-правило, определяющий какой урон будет наносить событие *fire()* при срабатывании. В единственном методе *fill(),* принимающем ссылку на поле, совершается обход по полю, во время которого поле клетки *damage*(по умолчанию равное 20) присваивается значение *x,* задающееся через шаблон.

HealRule – класс-правило, определяющий какой урон будет наносить событие *Heal()* при срабатывании. В единственном методе *fill(),* принимающем ссылку на поле, совершается обход по полю, во время которого поле клетки *heal*(по умолчанию равное 30) присваивается значение *x,* задающееся через шаблон.

FieldSizeRule – класс-правило, определяющий размеры поля. В единственном методе *fill(),* принимающем ссылку на поле, устанавливаются ширина и высота поля, задающееся через шаблон. Если новое поле больше старого, то добавляются новые клетки, заполненные событием *Clear_cell().* Иначе, старое поле обрезается. Если пользователь пытается создать поле, длина и/или ширина которого меньше трёх, то сгенерируется поле с размерами по умолчанию, а именно три на три.

Для генерации конкретного уровня в классе *Mediator* реализованы методы: *createField(), createFirstLevel(),createSecondLevel().*

Метода *createField()* в зависимости от выбора пользователя вызывает один из методов: *createFirstLevel()* или *createSecondLevel().*

В методах *createFirstLevel()* и *createSecondLevel()* создаётся объект класса *FieldGenerator,* с нужными шаблонами – правилами. Затем у данного объекта вызывается метод *generate().* После применения всех правил вызывается метод *CheckField* класса *Field*. Данный метод принимает указатель на поле и проверяет стоит ли на поле игрок и есть ли события *Key()* и *Level_exit().* Если эти условия соблюдены, то метод возвращает указатель на поле, если нет, то перед этим вызывает метод *setDefault(),* класса *Field.* В данном методе происходит случайная расстановка событий в клетки стандартного поля, размером 10 на 10. События *Key(), Level_exit()* и позиция игрока устанавливаются в конкретные клетки.

Тестирование программы.

На рисунках 1 - 3 представлено тестирование программы

Рисунок 1 – Создаём поле первого уровня, но указываем неправильные правила

Рисунок 2 – Создаём поле второго уровня

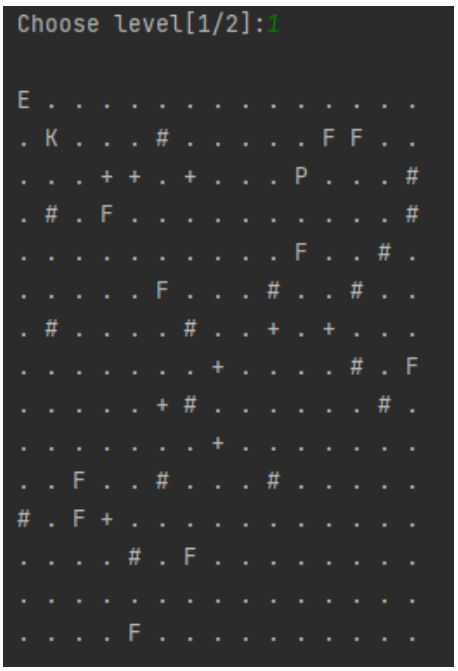

Рисунок 3 – Создаём поле первого уровня, указывая верные правила

UML-диаграмма межклассовых отношений.

На рисунке 4 изображена UML Диаграмма классов

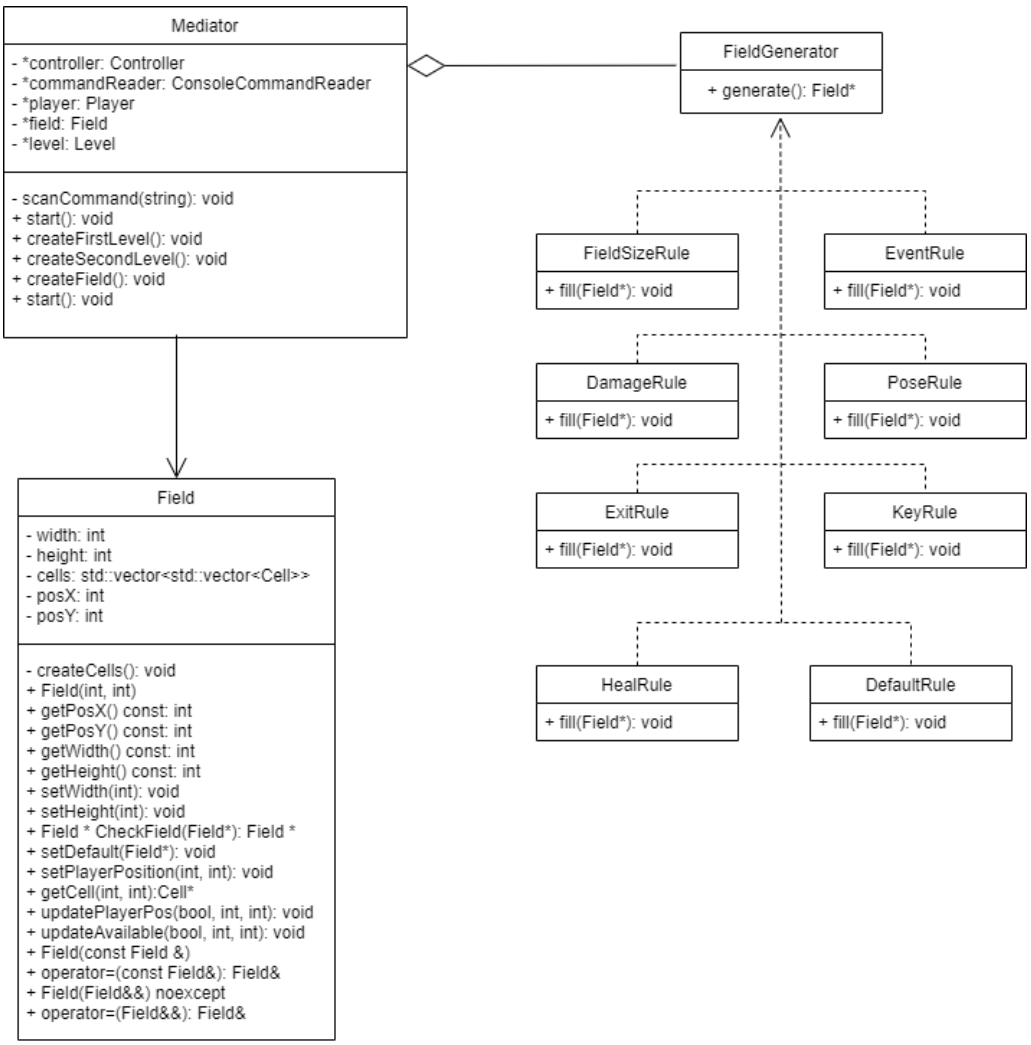

Рисунок 4 – UML Диаграмма классов

Выводы.

В ходе выполнения работы реализован шаблонный класс генерирующий игровое поле.