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: шаблонная класса, генерация карты

Студент гр. 1381 Возмитель В. Е

Преподаватель Жангиров Т.Р.

Санкт-Петербург

2022

Цель работы.

Реализация шаблонного класса, генерирующего игровое поле. Данный класс должен параметризироваться правилами генерации. Также реализация набора шаблонных правил.

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 создаёт поле, а не принимает его
- Класс генератор не должен принимать объекты классов правил в каком-либо методе, а должен сам создавать объекты правил из шаблонов
- Реализовано не менее 6 шаблонных классов правил
- Классы правила должны быть независимы и не иметь общего класса-интерфейса
- При запуске программы есть возможность выбрать уровень 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*

Выполнение работы.

1) Определяется шаблонный класс генератор поля FieldGenerator, создающий игровое поле в соответствии с классами-правилами, переданными в шаблон.

Реализуются методы класса с модификатором доступа *public*:

- *PrintField* fill(int height, int width, InfoLog* log_out_info, Player* player)* – *метод* генерирующий с помощью переданных в шаблон классов-правил и возвращающий указатель на игровое поле

2) Определяется шаблонный класс правила Enemy, расставляющий события Event_Enemy по игровому полю. Шаблон класса принимает объект перечисления *DIF*, означающий уровень сложности игры.

Реализуются методы класса с модификатором доступа *public*:

- *void operator()(CreateEvent& builder)* - перегрузка оператора (), расставляющего соответствующее событие по полю. С помощью цикла метода алгоритм проходится по всем клеткам поля и случайным образом определенным клеткам присваивает указатель на событие *Event_Enemy* в клетку

3) Определяется шаблонный класс правила *Hp*, расставляющий события *Event_Hp* по игровому полю. Шаблон класса принимает объект перечисления *DIF*, означающий уровень сложности игры.

Реализуются методы класса с модификатором доступа *public*:

- *void operator()(CreateEvent& builder)* - перегрузка оператора (), расставляющего соответствующее событие по полю. С помощью цикла метода алгоритм проходится по всем клеткам поля и случайным образом определенным клеткам присваивает указатель на событие *Event_Enemy* в клетку

4) Определяется класс правила *Arm*, расставляющий события

Event_Arm по игровому полю. Реализуются методы класса с модификатором доступа *public* аналогично классу Enemy.

5) Определяется класс правила *Trap*, расставляющий события *Event_Damage* по игровому полю. Реализуются методы класса с модификатором доступа *public* аналогично классу Enemy.

6) Определяется класс правила *Wall*, расставляющий непроходимые клетки по игровому полю. Реализуются методы класса с модификатором доступа *public* аналогично классу Enemy

7) Определяется класс правила *Win*, расставляющий события *Event_Win* по игровому полю, определяются условия победы в игре в зависимости от выбранной сложности игры. Реализуются методы класса с модификатором доступа public аналогично классу Enemy

Архитектура программы.

В методе создания поля *create_field*() класса *Controller* с помощью метода *fill()* класса *FieldGenerator* создаётся и заполняется игровое поле. Каждое отдельное событие создаётся случайным образом с помощью библиотеки *random*. Сложность игры влияет на диапазон выбора случайного значения, чем выше уровень сложности, тем меньше диапазон.

Для определения уровня в шаблон передаётся элемент перечисления DIF, изменяющий своим значением диапазон.

UML-диаграмма представлена на рис. №1.

UML диаграмма.

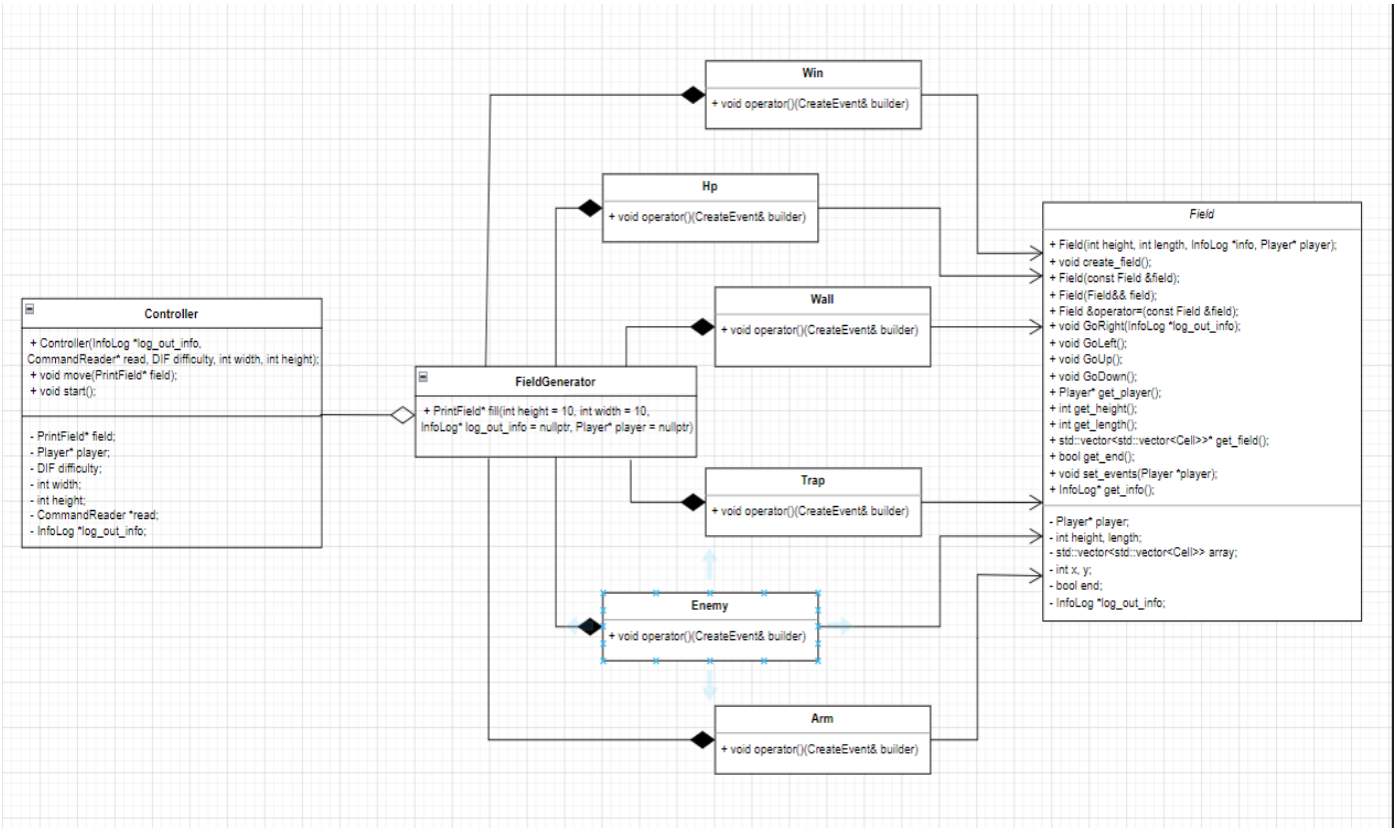

Рис 1 – UML-диаграмма.

Выводы.

Реализован шаблонный класс генерации игрового поля, а также шаблонные классы правил.