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. 1381 _____ Мамин Р.А.

Преподаватель _____ Жангиров Т.Р.

Санкт-Петербург
2022

Цель работы.

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)

Задание.

Реализовать набор классов отвечающих за считывание команд пользователя, обрабатывающих их и изменяющих состояния программы (начать новую игру, завершить игру, сохраниться, управление игроком, и.т.д.). Команды/клавиши определяющие управление должны считываться из файла.

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.
- 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

Выполнение работы. Ход решения:

Используется стандартная библиотека C++.

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*:

- *Field *fill(LogOutInfo *info, Player *player, int width, int height)*– метод, создающий, генерирующий с помощью переданных в шаблон классов-правил и возвращающий указатель на игровое поле.

2. Определяется шаблонный класс правила *CoinSetRule,* расставляющий события GetCoin по игровому полю. Шаблон класса принимает объект перечисления *COMPLEXITY*, означающий уровень сложности игры.

Реализуются методы класса с модификатором доступа *public*:

- *void operator()(Field *field)* – перегрузка оператора *(),* расставляющего соответствеющее событие по полю. С помощью цикла метод алгоритм проходится по всем клеткам поля и при выполнени определённого условия (кратности абсолютных значений координат заданным числам) выставляет указатель на событие *Coin* в клетку.

3. Определяется класс правила *CollapseSetRule,* расставляющий события GetCoin по игровому полю.

Реализуются методы класса с модификатором доступа *public* аналогично классу *CoinSetRule*.

4. Определяется класс правила *EnemySpawnRule,* расставляющий события *EnemySpawn* по игровому полю.

Реализуются методы класса с модификатором доступа *public* аналогично классу *CoinSetRule.*

5. Определяется класс правила *FieldCreaseRule,* расставляющий события

Increase и *Decrease* по игровому полю.

Реализуются методы класса с модификатором доступа *public* аналогично классу *CoinSetRule.*

6. Определяется класс правила *HealSetRule*, расставляющий события *Heal* по игровому полю.

Реализуются методы класса с модификатором доступа *public* аналогично классу *CoinSetRule.*

7. Определяется класс правила *PlayerSpawnSetRule,* задающий координаты появления игрока на игровом поле.

Реализуются методы класса с модификатором доступа *public* аналогично классу *CoinSetRule.*

8. Определяется класс правила *WallSetRule,* расставляющий непроходимые клетки по игровому полю.

Реализуются методы класса с модификатором доступа *public* аналогично классу *CoinSetRule*.

Архитектура программы.

В методе создания поля *create_field()* класса *Controller* с помощью метода *fill()* класса *FieldGenerator* создаётся и заполняется игровое поле. Каждое отдельное событие/группа событий и игрок создаются по определённым координатам, которые определяются согласно остаткам от деления на различные числа в условии соответствующего правила. Для определения уровня в шаблон передаётся элемент перечисления *COMPLEXITY*, изменяющий своим значением частоту расстановки соответствующего события или игрока.

Результат работы программы:

Результаты работы программы представлены на рисунке 1.

Рис. 1 – демонстрация работы программы в терминале Ubuntu.

Генерация игрового поля.

UML-диаграмма межклассовых отношений:

UML-диаграмма представлена на рисунке 2.

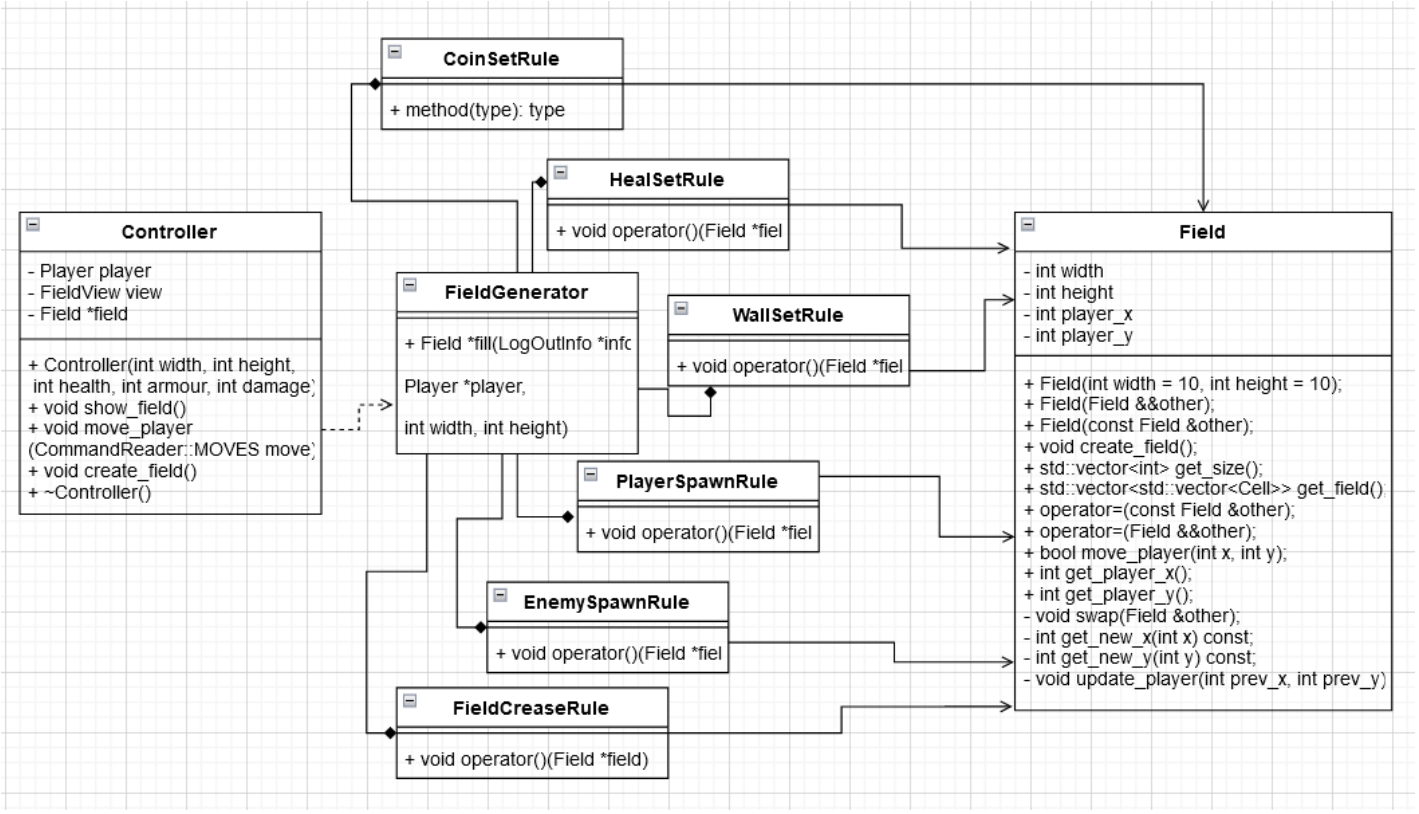

Рис 2. – UML-диаграмма.

Вывод: Реализован шаблонный класс генерации игрового поля а также шаблонные классы правил для конкретных уровней. Была изучена работа с классами на языке C++, паттерны проектирования, основы составления UML-диаграмм.