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

ОТЧЕТ

по лабораторной работе №5
по дисциплине «Объектно-ориентированное программирование»
Тема: Шаблонные классы, генерация карты

Студент гр. 1304 ___ Байков Е.С.

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)

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

Описание архитектурных решений и классов.

Реализовано несколько шаблонных классов, которые осуществляют генерацию игровой карты. Также созданы шаблонные классы для правил, по которым будет строиться будущая карта. Подробное описание классов:

1. *FieldScheme*: представляет собой контейнер, по которому будет строиться будущая карта. Хранит в себе *enum CellType*.
2. *LevelGenerator*: представляет собой шаблонный класс генерирующий поле по заданным правилом, принимая при этом в себя вариативное количество правил. Метод *generate()* создает схему и производит распаковку правил, вызывая при этом метод *applyRule()*, а затем производится обновление схемы по заданному правилу. Затем метод возвращает схему.
3. *IStrategy*: интерфейс для подключения разных уровней, которые может выбрать игрок в начале игры.
4. *FirstLevelInfo, SecondLevelInfo, ThirdLevelInfo*: классы реализующие интерфейс *IStrategy*. Содержат в себе генератор, и правила, по которым он будет создавать схему. Метод *update()* возвращает схему, заполненную по правилам, которые указаны в данном уровне. Генератор может принимать правила в любом порядке.
5. *RuleEnemy, RuleFieldChanger, RuleUnexplored, RuleWalls*: правила определяющие оккупированные, неизведанные, блокированные клетки и клетки с событием изменяющим поле. Могут быть созданы в конкретной клетке, могут быть созданы в каком-то количестве при этом будут создаваться случайно.
6. *RuleExit, RulePlayerPosition*: определяют позицию игрока и позицию выхода для него, принимая два параметра позицию X и Y, в которую будет установлена соответствующая точка (появления персонажа или окончания игры). Поскольку позиции игрока и выхода являются основными для успешного прохождения игры, происходит проверка того что игрок дойдет до выхода в любом случае.

7. *FieldGenerator*: обновлена логика этого класса, который теперь создает поле по схеме, которую хранит в себе. Добавлен метод, который создает поле и возвращает его после заполнения событиями и игроком.
8. *Menu*: обновлена логика данного класса, теперь вместо ввода размеров поля, запрашивает выбор уровня. По умолчанию отправляет игрока на первый уровень.

Демонстрация работы программы и тестирование.

Реализовано три уровня. Первый размером 7x7, второй размером 10x10 и третий размером 15x15. Для каждого уровня заданы свои параметры событий, изменяющих поле, закрытых клеток и клеток со врагом, а также место появления игрока и место выхода из игры. Создание поля для второго уровня на рисунке 1.

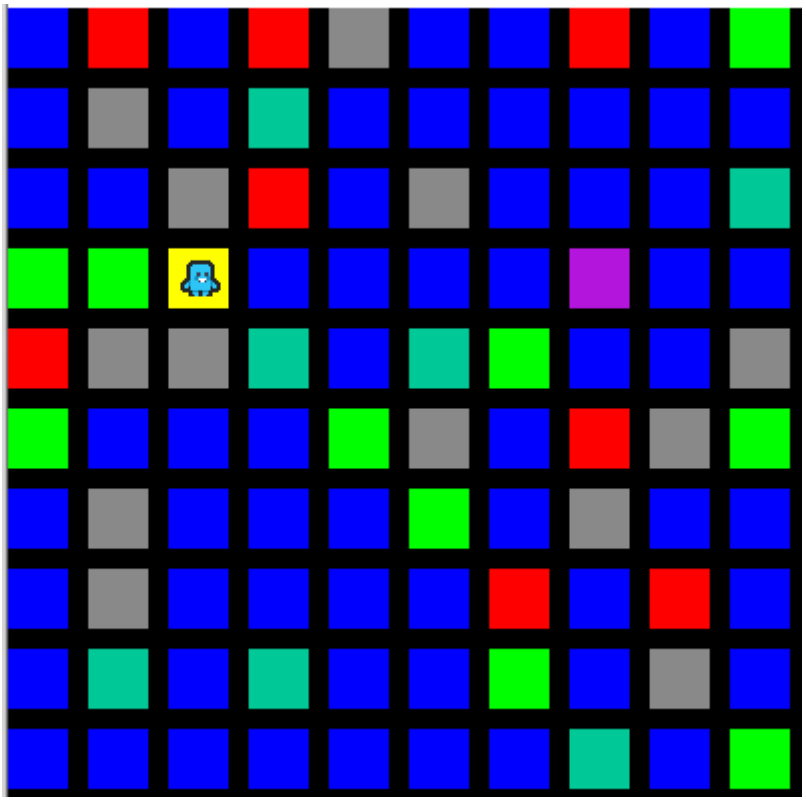

Рисунок 1 — Поле, сгенерированное для второго уровня.

Вывод сообщения об установке стандартного уровня на рисунке 2.

```
 [Info]: Defau
lt level FIRST has been set
```

Рисунок 2 — генерация уровня по умолчанию.

Генерация поля для первого уровня на рисунке 3.

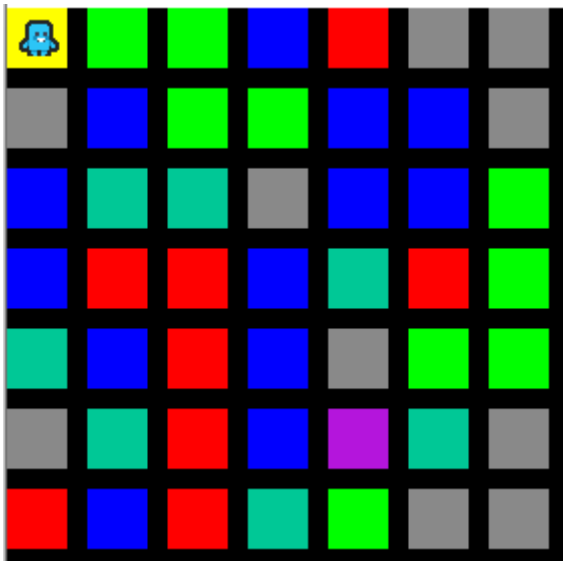

Рисунок 3 — Поле, сгенерированное для первого уровня.

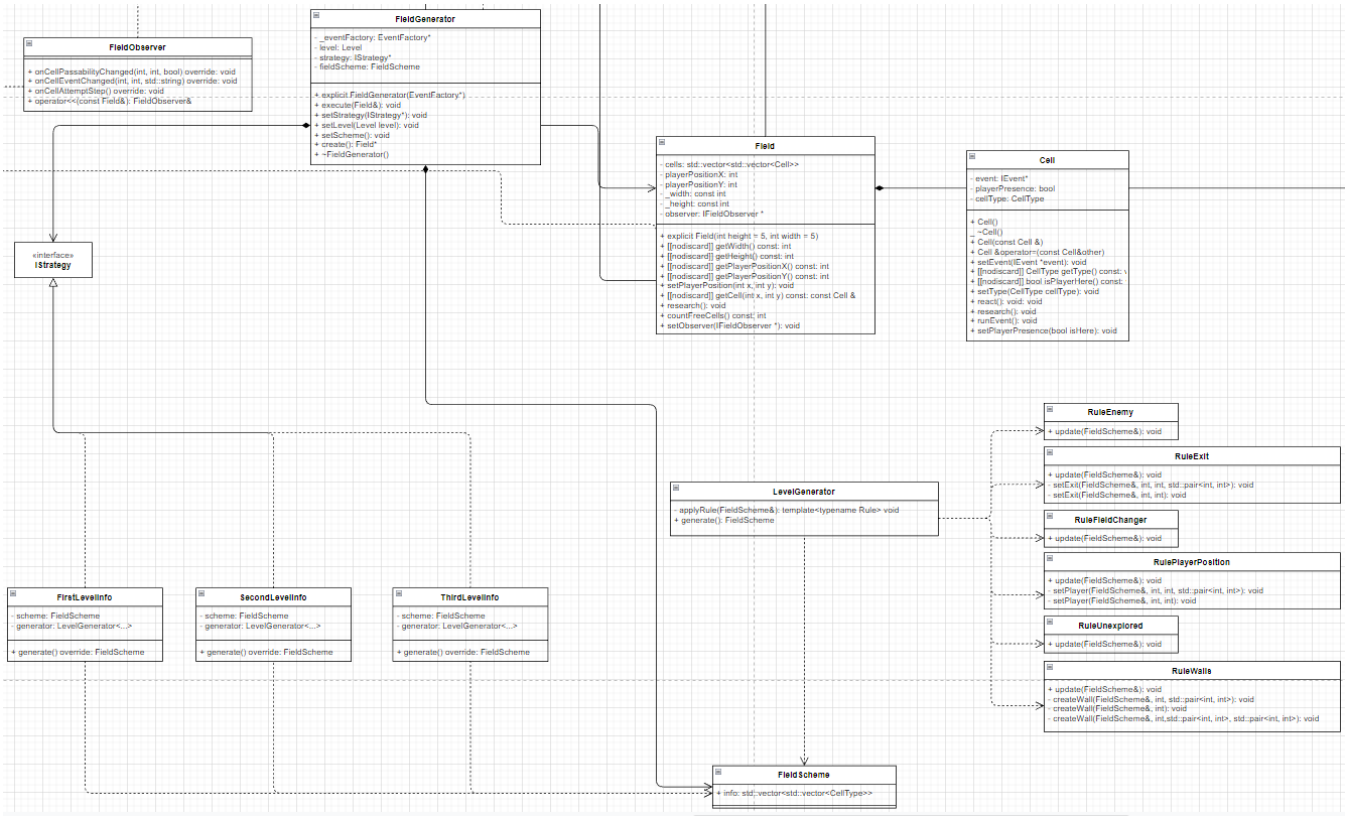

Рисунок 4 — UML-диаграмма классов.

Вывод.

Реализованы шаблонные классы, представляющие собой правила создания карты для игры, а также класс, принимающий в себя вариативное количество параметров (правил) и создающий по этим параметром поле определенного типа. Также добавлена возможность выбора определенного уровня игроком.