**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

|  |  |  |
| --- | --- | --- |
| Студентка гр. 1303 |  | Королева П.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

## Выполнение работы.

Для реализации этой лабораторной работы был использован паттерн стратегия.

При создании контроллера, вызывается новый метод *init\_map\_with\_level().* В нем программа спрашивает у пользователя, какой уровень он хочет запустить и в соответствии с введенной цифрой, устанавливает в поле context нужный уровень.

В поле context находится объект нового класса **LevelContext**. Этот класс нужен для управления уровнями, через интерфейс *LevelStrategy*. Этот класс при использовании метода *set\_level()* запускает инициализацию поля и через *get\_field()* возвращает его в контроллер.

**LevelStrategy** является интерфейсом, который реализуют *LevelOne* и *LevelTwo.* Имеет всего один виртуальный метод – *create\_field().*

**LevelOne** и **LevelTwo** создают шаблонный класс *FieldGenerator* с нужными шаблонными правилами генерации.

**FieldGenerator** принимает в качестве аргументов шаблона некие правила генерации rules, каждое из которых будет с помощью перегруженного оператора() применяться к полю, которое будет создано в методе create() и определенным образом его модифицировать.

**Rules.**

Реализованные правила генерации включают следующие классы:

**RuleWin** – устанавливает условие победы

**RuleSize** – устанавливает новые размеры поля с помощью нового метода *Field.resize()*

**RuleBone** – устанавливает на поле count косточек

**RuleBarb** – устанавливает на поле count колючек

**RuleLevelUp** – устанавливает на поле count цветочных полянок для увеличения уровня

**RuleMap** – устанавливает на поле count карт с зарытыми косточками

**RuleRock** – устанавливает на поле count камней

**UML-диаграмма лабораторной работы №5.**

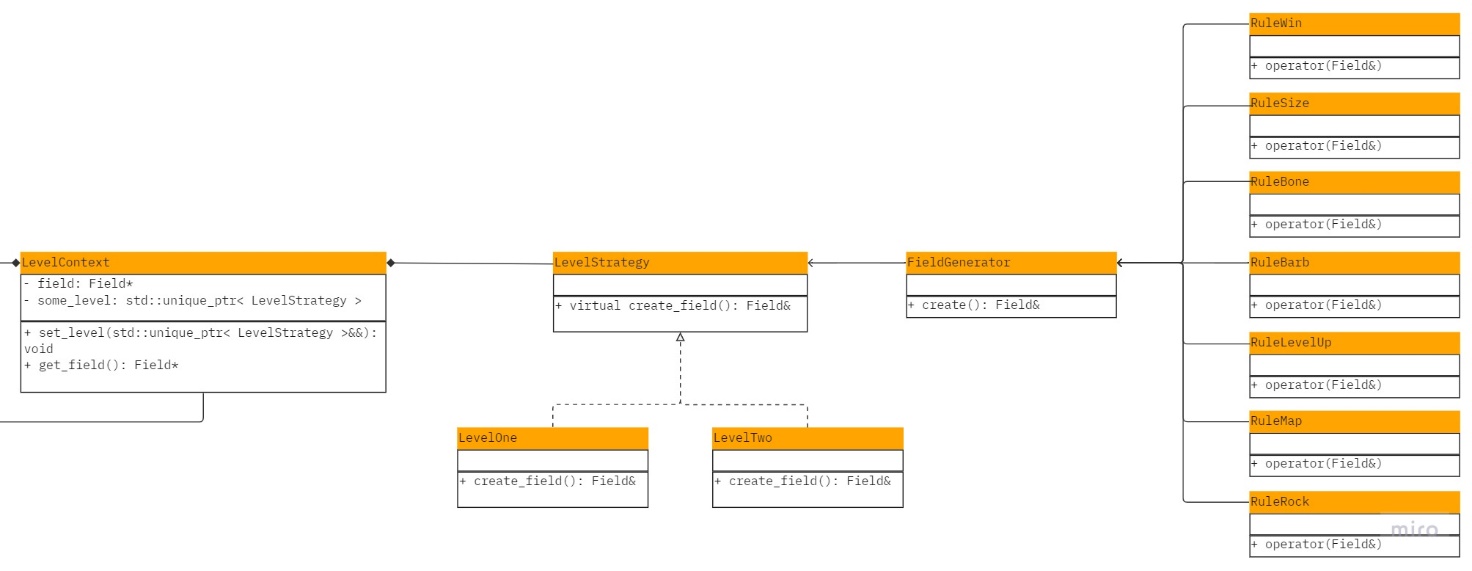
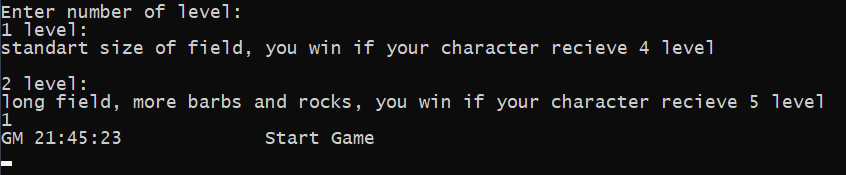
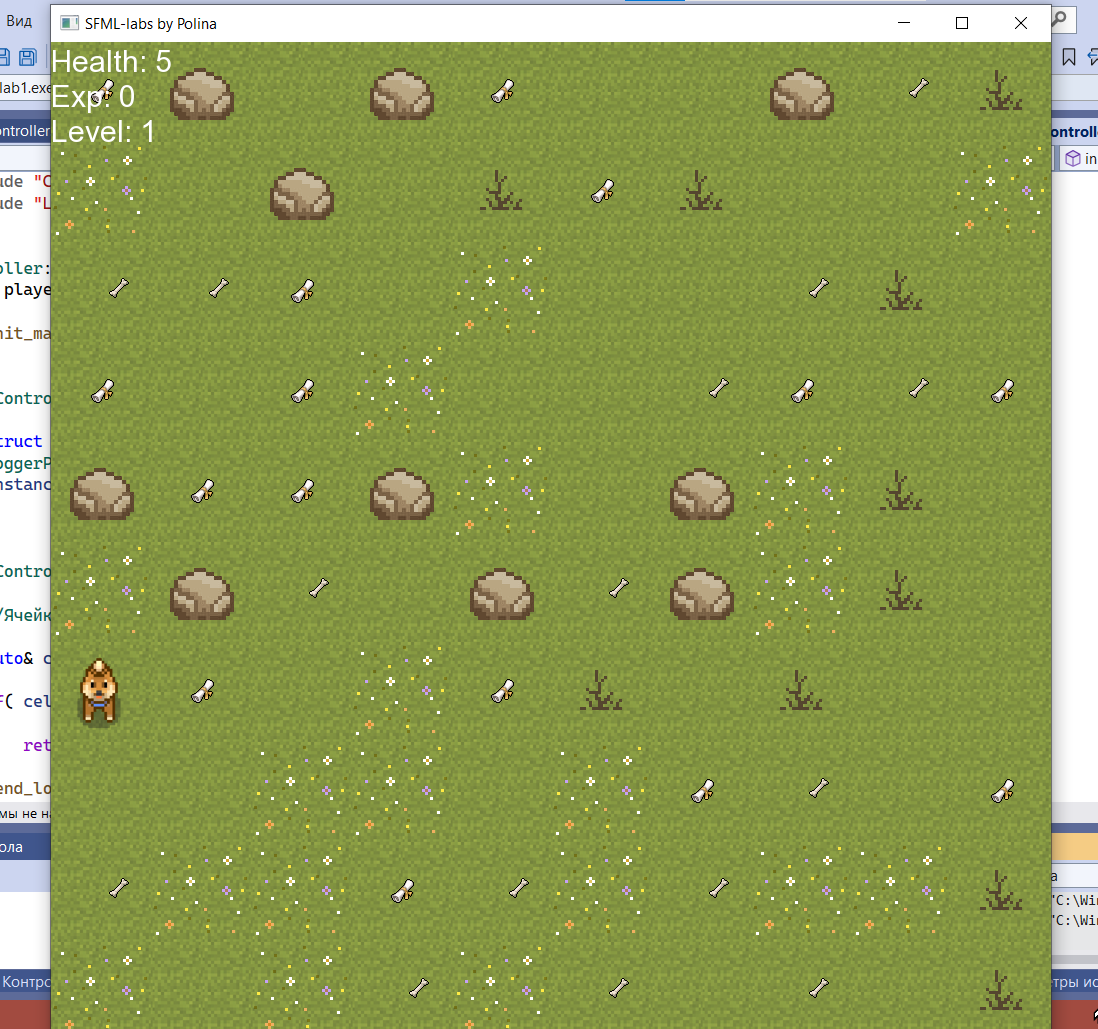
Иерархия классов представлена на рисунке 1.

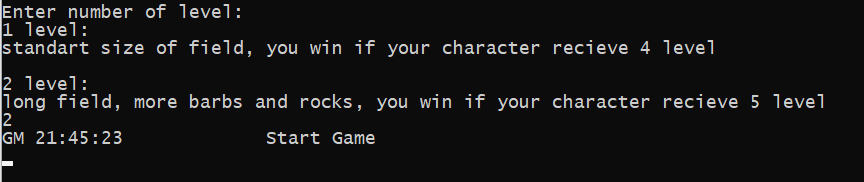
Рис.1 – Диаграмма иерархии классов инициализации поля

## Тестирование.





## Рис.2 – Запуск программы, с первым уровнем



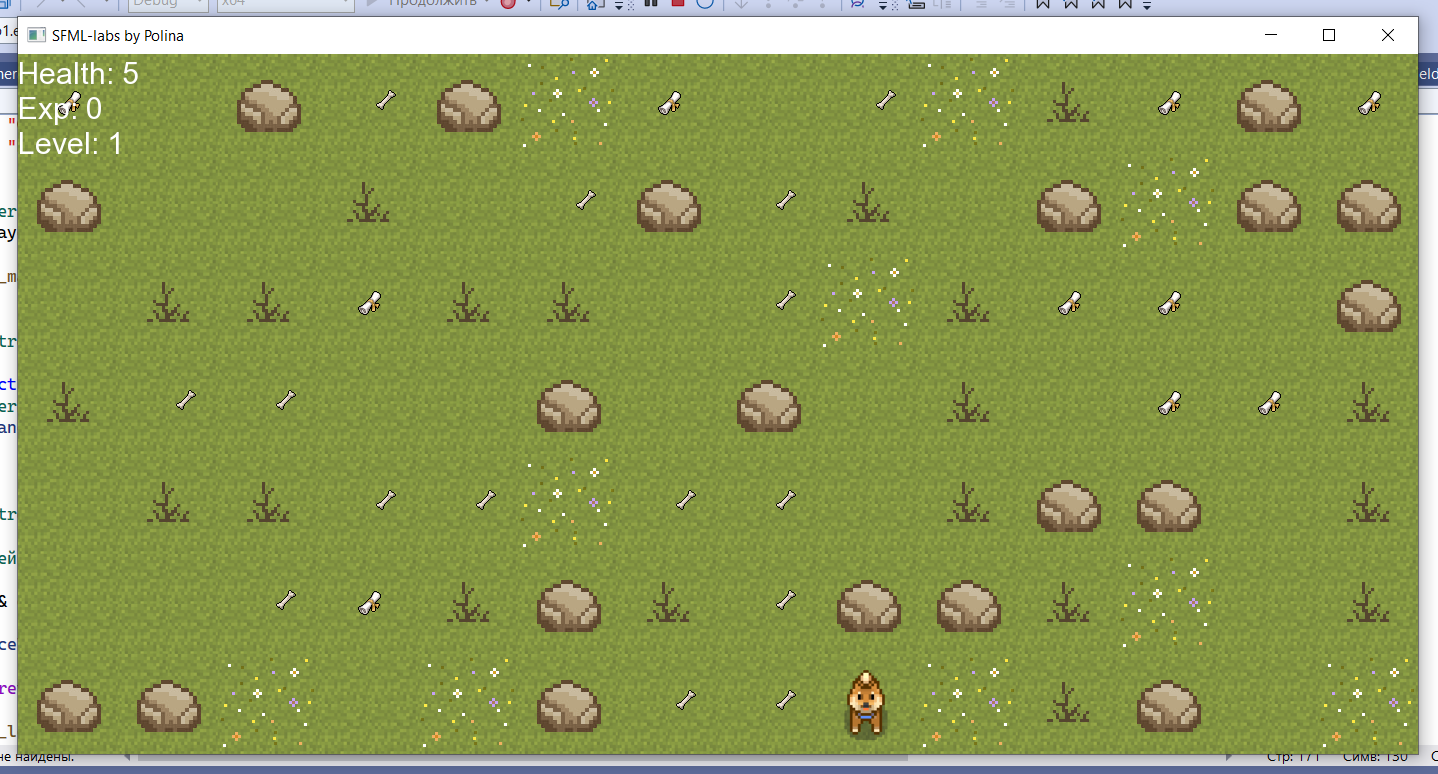


Рис.3 - Запуск программы, со вторым уровнем

## Выводы.

## Был реализован шаблонный класс генерирующий игровое поле.