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

Студентка гр.1381 _____ Рымарь М.И.

Преподаватель _____ Жангиров Т.Р.

Санкт-Петербург
2022

Цель работы.

Реализовать шаблонный класс, генерирующий игровое поле. Данный класс должен параметризироваться правилами генерации. Также необходимо создать набор 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 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 правилах

Выполнение работы.

В ходе выполнения лабораторной работы создан шаблонный класс, генерирующий игровое поле и набор шаблонных классов-правил, которыми задаётся генерация поля.

В директории поля создан шаблонный класс *fieldGenerator,* его шаблоном является неограниченное количество правил. В шаблонном классе есть только один метод – *createField(),* который создаёт поле, применяя классы-правила.

Созданы классы-правила, являющиеся функторами из-за того, что в них переопределён оператор круглых скобок. Каждый класс правил при вызове оператора () принимает указатель на поле, а при определении оператора изменяет это поле.

Класс-правило *fieldSize* задаёт размеры поля через шаблоны. Сначала удаляется поле, созданное в методе *createField()*, а потом создаётся поле с заданным размером. Это сделано для того, чтобы создавалось после со стандартными размерами в случае, если не выбрано правило. Также реализована проверка параметров, чтобы не создавалось поле с некорректными размерами.

Класс-правило *playerPosition* задаёт позицию игрока. В случае, если клетка является проходимой и на ней нет события, координаты задаются через шаблонные параметры. В обратном случае игрок ставится на клетку с координатами (0;0).

Класс-правило *setWallsColumn* устанавливает стены (непроходимые клетки) в столбец. Добавлена проверка на корректность значений размеров поля. Установлены пределы, чтобы случайно не создалась ситуация, в которой игрок будет заперт.

Класс-правило *setWallsRow* устанавливает непроходимые клетки в ряд. Проверка и ограничения аналогичны предыдущему правилу.

Класс-правило *spawnEventField* отвечает за установку событий, влияющих на поле. В шаблонные параметры передаются: класс-событие, значение типа bool и два значения типа int. В столбец ставятся события, связанные с полем. Второе значение из шаблонных параметров отвечает за установку события поля на клетку даже в том случае, если на клетке уже есть событие. Одно из целых значений отвечает за ограничения по событиям, чтобы они не занимали весь столбец.

Класс-правило *spawnEventPlayer* устанавливает события, влияющие на игрока, в ряд. Правило аналогично предыдущему, только отсутствует значение в параметрах шаблона, отвечающее за установку события даже на занятую клетку.

Для того, чтобы задать определенную генерацию уровня, изначально создан интерфейс levelStrategy, который в классах-наследниках генерирует поле по определенным правилам и возвращает указатель на него. От этого класса наследуются классы firstLevel и secondLevel, в которых определяется метод, где генерируется поле с разными правилами. Также был создан levelContext, в полях которого находится указатель на поле и указатель на уровень, который содержит наследника levelStrategy. В конструкторе принимает уровень, а также предоставляет сеттер для его изменения во время выполнения. Создан геттер для поля. Также реализован метод setLevel(), который у lvl вызывает метод generateLevel(), генерирующий поле по определенным правилам, и записывает это поле в поля класса. Таким образом реализован паттерн стратегия.

Тестирование.

Интерфейс командной строки при генерации поля первого уровня представлен на рисунке 1.

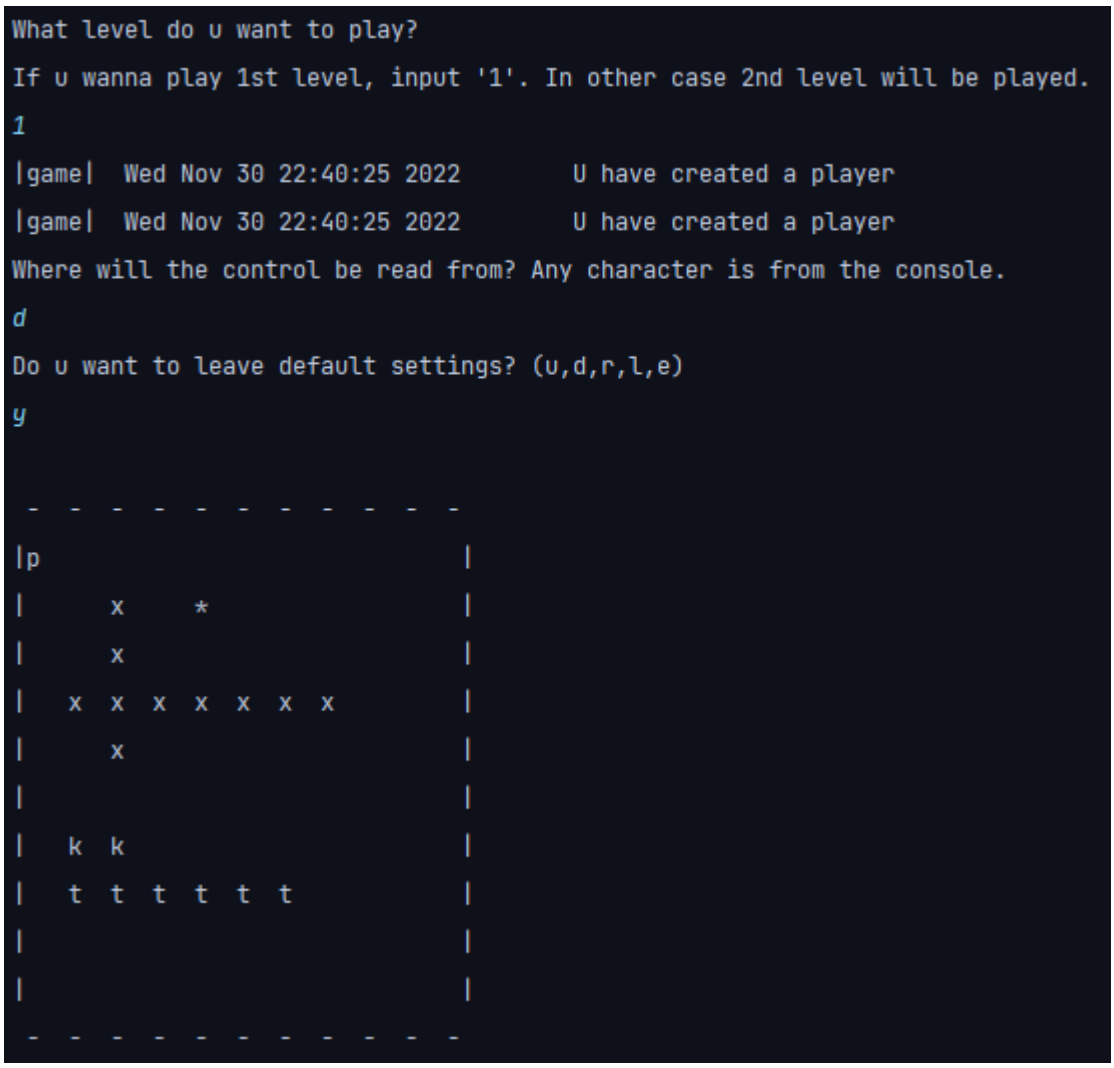

Рисунок 1 – Генерация поля первого уровня

Работа программы при генерации поля второго уровня представлена на рисунке 2.

Рисунок 2 – Генерация поля второго уровня

UML-диаграмма межклассовых отношений.

На рисунке 3 представлена UML-диаграмма межклассовых отношений.

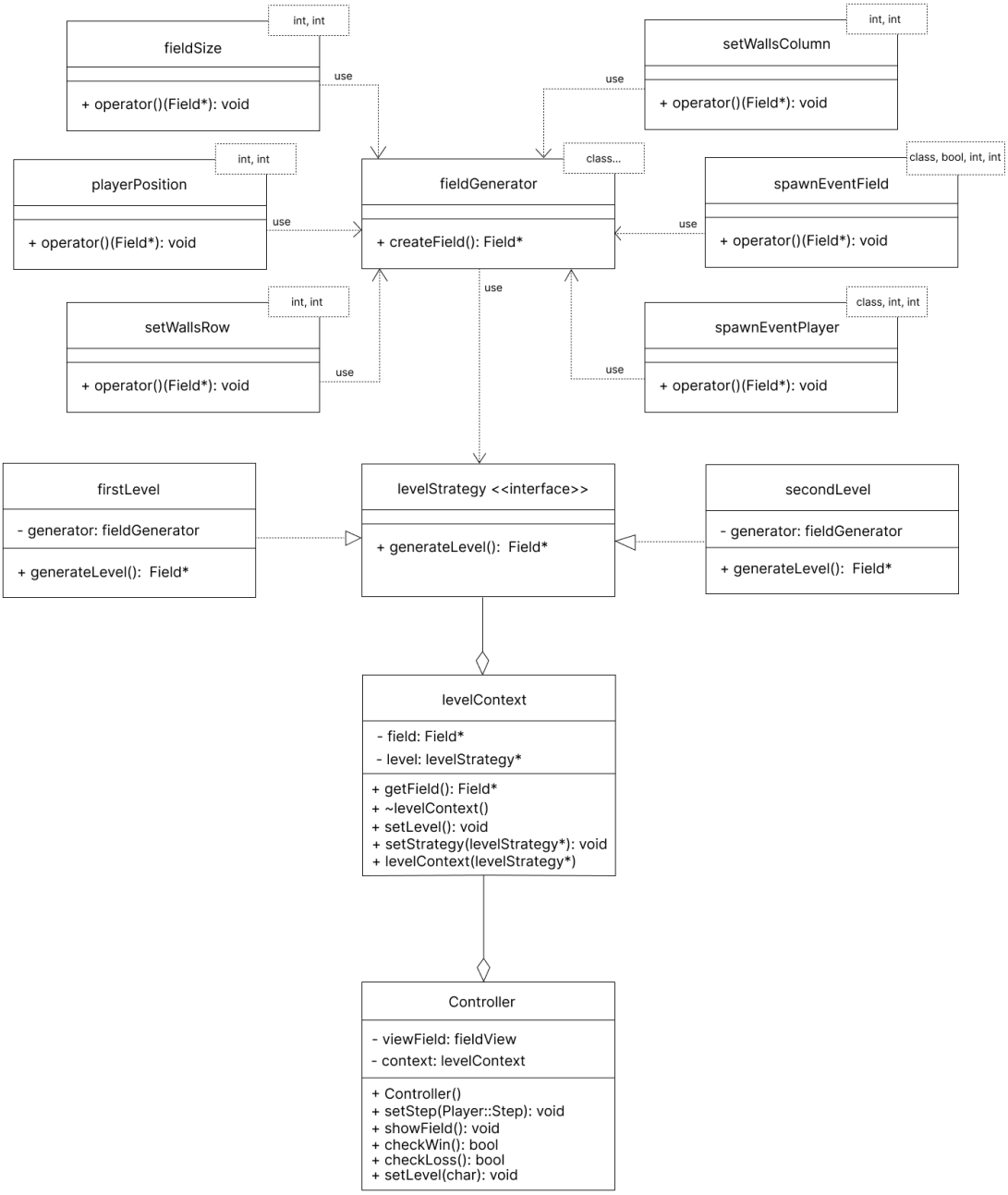

Рисунок 3 – UML-диаграмма

Выводы.

В ходе выполнения лабораторной работы реализован шаблонный класс, генерирующий игровое поле. Этот класс параметризируется правилами генерации. Создан набор шаблонных правил.