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.

Студент гр. 1384 _____ Тапеха В. А.

Преподаватель _____ Жангиров Т. Р.

Санкт-Петербург
2022

Цель работы.

Необходимо реализовать шаблонный класс генерирующий игровое поле. Этот класс должен задаваться правилами генерации. Также необходимо реализовать 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.

Для выполнения лабораторной работы были созданы шаблонный класс, генерирующий игровое поле и набор шаблонных классов-правил, которыми задается генерация поля.

Новые классы:

Был реализован шаблонный класс FieldGenerator, шаблоном которого является какое-то неопределенное количество правил. В нем есть только один метод Field* fill(), который создает поле, задает его, применяя классы-правила к нему и используя распаковку шаблона, и возвращает указатель на него.

Были созданы классы-правила. В них переопределен оператор (), то есть они являются функторами. Каждый класс правил при вызове оператора () принимает указатель на поле Field* field и при определении оператора () изменяет это поле.

Классы-правила:

1) RuleFieldSize — задает размеры поля. Размеры поля задаются через шаблоны. Удаляет созданное поле в методе fill и создает новое поле уже с заданным размером, а не стандартным(10, 10). Так сделано, чтобы, если не было выбрано правило, которое задает размеры поля, было создано поле со стандартным размером.

2) RuleSpawnEventField — ставит одно событие, которые как-то влияет на поле. Какое конкретно событие нужно поставить задается через шаблон. На позицию этого события влияет magic_number, который задается в шаблоне. С помощью несложных арифметических действий над индексами и magic_number, которые задают положение клетки, на которую будет ставиться

событие, определяются координаты клетки. Если соблюдены условия у этой клетки, то на клетку ставится событие и она становится проходимой.

3) RuleSpawnEventPlayer — ставит определенное количество событий, которые как-то влияет на игрока. Количество событий задается через шаблон max_count. Правило пытается поставить максимальное количество событий, но, если мест не будет хватать, то оно прекратит ставить события. Какое конкретно событие нужно поставить задается через шаблон. На позицию этого события влияет magic_number, который задается в шаблоне. С помощью несложных арифметических действий над индексами и magic_number, которые задают положение клетки, на которую будет ставиться событие, определяются координаты клетки. Если соблюдены условия у этой клетки, то на клетку ставится событие и она становится проходимой.

4) RuleSpawnKeys — ставит 2 события, которое ставит ключ. Координаты ключа задаются через шаблонный параметр magic_number, который проверяется на отрицательность и на выход за границы размеров поля. Первое событие-ключ ставится по координатам (ширина — 1; magic_number). Второе событие-ключ ставится по координатам (magic_number; высота — 1). Эти две клетки с событиями становятся проходимыми.

5) RuleSpawnPlayer — устанавливает позицию игрока. Координаты задаются через шаблонные параметры. Проверяет, что клетка не занята событием и является проходимой. Иначе игрок ставится на позицию (0; 0).

6) RuleSpawnTrap — ставит определенное количество событий-ловушек, которые отнимают здоровье у игрока. Количество этих событий задается через шаблон max_count. Урон, который наносит клетка с ловушкой тоже задается шаблонно через параметр damage. Правило пытается поставить максимальное количество событий, но, если мест не будет хватать, то оно прекратит ставить события. На позицию этого события влияет magic_number, который задается в шаблоне. С помощью несложных арифметических действий над индексами и magic_number, которые задают положение клетки, на которую будет ставиться

событие, определяются координаты клетки. Если соблюдены условия у этой клетки, то на клетку ставится событие и она становится проходимой.

7) RuleSpawnWalls — ставятся непроходимые клетки, которые задаются так же, как и в прошлых правилах через magic_number.

Для того, чтобы задать определенную генерацию уровня, изначально был создан интерфейс LevelStrategy с чистым виртуальным методом Field* generate_level(), 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 Field* generate_level(), где генерируется поле с разными правилами.

Также был создан LevelContext, в полях которого находится указатель на поле и unique_ptr<LevelStrategy> level, который содержит наследника LevelStrategy. В конструкторе принимает уровень через конструктор, а также предоставляет сеттер для его изменения во время выполнения. Был создан геттер для поля. Также был реализован метод set_level(), который у level вызывает вызывает метод generate_level(), генерирущий поле по определенным правилам, и записывает это поле в поля класса.

Измененные классы:

В классе Control теперь больше не хранится поле. Вместо него хранится LevelContext, который имеет доступ к полю и может изменять поле. И теперь игрок не может задавать размеры поля самостоятельно. Теперь игра у него спрашивает номер уровня. Поэтому был добавлен метод void set_level(char), который принимает символ и сравнивает его с единицей. Если была введена единица, то у поля устанавливается первый уровень. Иначе ставится второй.

UML-диаграмма межклассовых отношений.

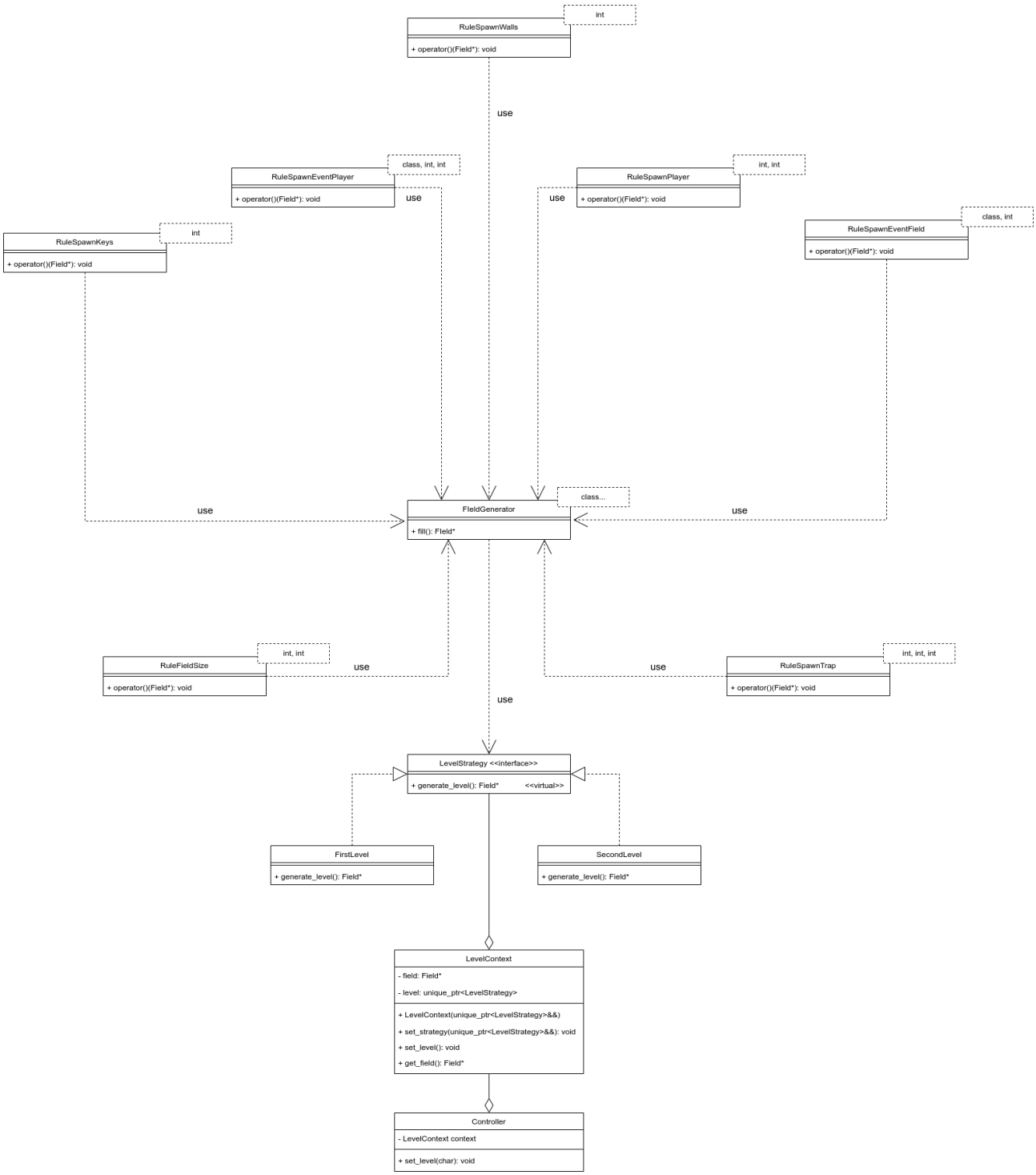

Рис. 1 UML-диаграмма.

Выводы.

Был реализован шаблонный класс, генерирующий игровое поле. Данный класс параметризируется правилами генерации. Также был реализован набор шаблонных правил.