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

ОТЧЕТ

по лабораторной работе №4

по дисциплине «ООП»

Тема: Шаблонные классы, управление

Студент гр. 0383 _____ Козлов Т.В.

Преподаватель _____ Жангиров Т.Р.

Санкт-Петербург

2021

Цель работы.

Необходимо определить набор правил для игры в виде классов (например, какие задачи необходимо выполнить, чтобы он мог выйти с поля; какое кол-во врагов и вещей должно быть на поле, и.т.д.). Затем определить класс игры, которое параметризуется правилами. Класс игры должен быть прослойком между бизнес-логикой и командами управления, то есть непосредственное изменение состояния игрой должно проходить через этот класс.

Требование:

- Созданы шаблонные классы правил игры. В данном случае параметр шаблона должен определить конкретные значения в правилах.
- Создан шаблонный класс игры, который параметризуется конкретными правилами. Класс игры должен проводить управление врагами, передачей хода, передавать информацию куда переместить игрока, и.т.д.

Потенциальные паттерны проектирования, которые можно использовать:

- Компоновщик (Composite) - выстраивание иерархии правил
- Фасад (Facade) - предоставления единого интерфейса игры для команд управления
- Цепочка обязанностей (Chain of Responsibility) - обработка поступающих команд управления
- Состояние (State) - отслеживание состояние хода / передача хода от игрока к врагам
- Посредник (Mediator) - организация взаимодействия элементов бизнес-логики

Ход работы:

Для реализации правил игры был создан класс Rule – от которого будут наследоваться все правила игры.

Для определения количества врагов на поле был создан шаблонный класс EnemySpawnRule() со следующим шаблоном: *template<class ChooseCellClass, class EnemyFirst, int countFirst, class EnemySecond, int countSecond, class EnemyThird, int countThird, class LogRule>,* где *ChooseCellClass* – класс, отвечающий за способ определения клетки, в которой будет происходить спавн противника (для этого в рамках данной лабораторной работы был написан абстрактный класс ChooseCell с чистой виртуальной функцией GetCell() – возвращающей ссылку на клетку. От этого класса наследуется QtRandomChooseCell – класс,

использующий библиотеку Qt для генерирования рандомных чисел (с проверкой на то, что сгенерированные координаты указывают на пустую клетку типа Way))

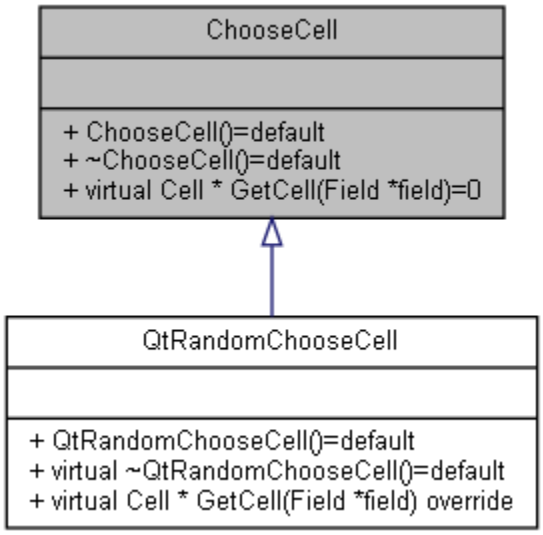

Рис. «Класс ChooseCell и его наследники»

Далее последовательно идет тип врага и количество данного типа на поле (всего 3 пары значений), а потом идет правило для логирования *LogRule* (см. далее).

По функционалу данный класс имеет всего одну переопределенную функцию – Spawn() – которая используя класс-определитель клеток создает экземпляры Врагов в нужном количестве (напоминание: создание Врагов устроено так, что при создании экземпляра врага он автоматически помещается на клетку – и контроль за занимаемой им памятью уже переходит в ответственность поля, на котором тот был создан).

Так же был создан шаблонный класс LoggerRule – задающий способ логирования. Его шаблон содержит две переменные типа bool consolLog и fileLog – отвечающие за логирование в консоль и файл соответственно. При создании экземпляра данного класса, в случае истинности соответствующей переменной создаются нужные экземпляры Логгеров (иначе ссылка остается nullptr), а в деструкторе Логгеры уничтожаются.

По функционалу данный класс содержит всего один метод AddInLogger(Loggable* loggable) – добавляющий логируемый объект в логгер.

Итого получилась следующая UML для данной части (рис. «Класс Rule и его наследники») P.s: В дальнейшем количество правил увеличится):

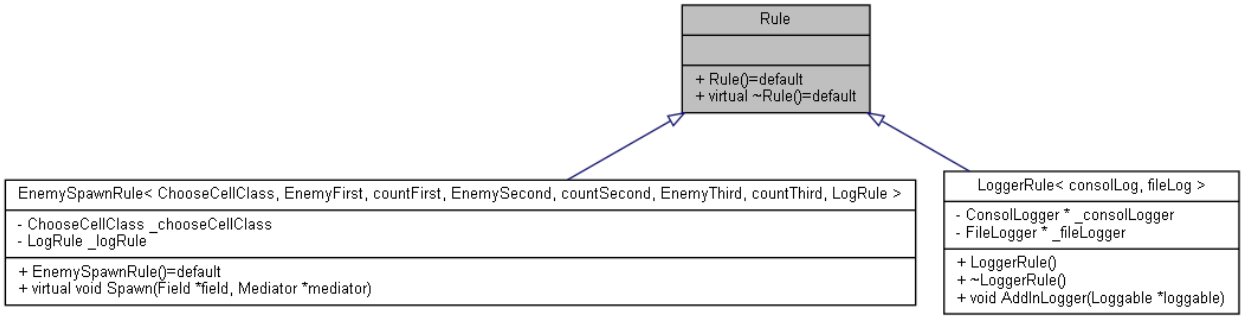

Рис. Класс Rule и его наследники

В процессе создания шаблонного класса игры, был немного переделан класс Mediator, реализующий паттерн «Медиатор» (посредник, контроллер) – через который передавались данные (напр. команда игроку переместится и куда, или обновление графического интерфейса при изменении какого-нибудь класса).

Класс Mediator стал интерфейсом с множеством чистых виртуальных функций, определяющих взаимодействие между модулями программы. От Mediator наследуется шаблонный класс TemplateMediator, шаблон которого задает правила игры (см Рис. «Класс Mediator и его наследники»).

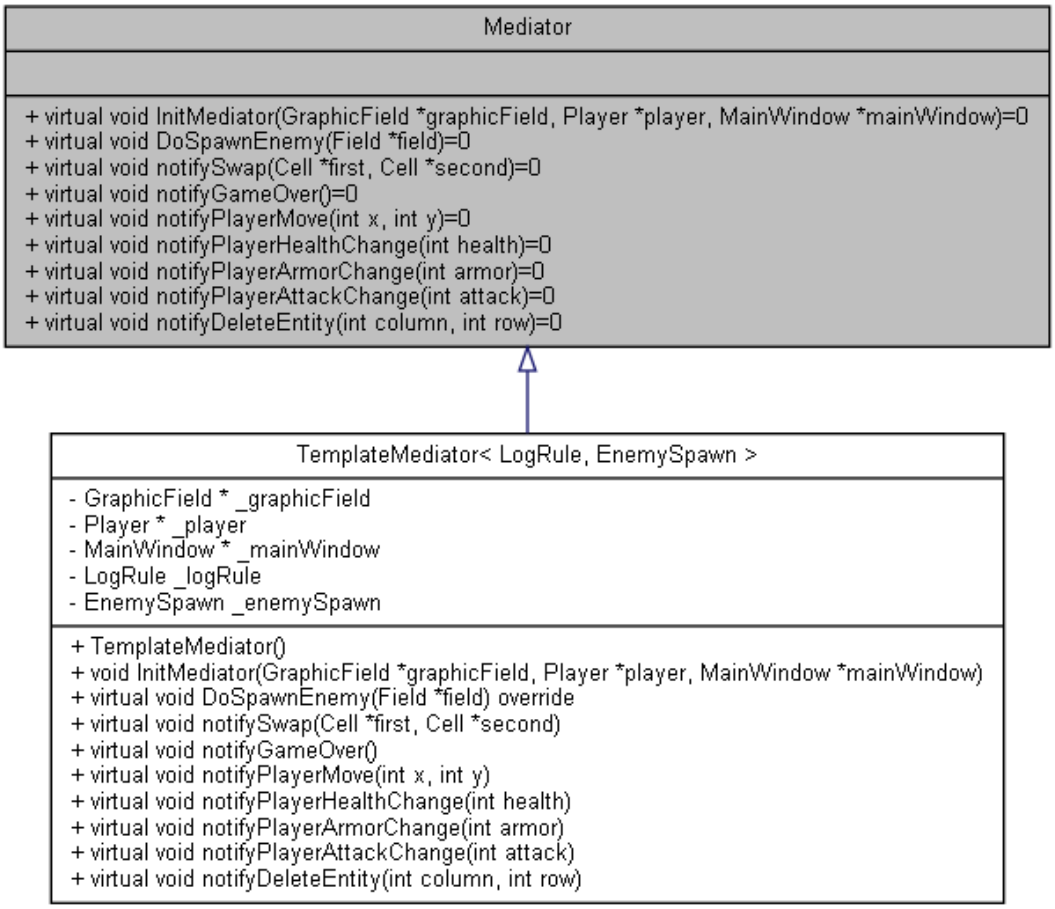

Рис. «Класс Mediator и его наследники»

На момент выполнения лабораторной работы его шаблон принимал два класса: class LogRule и класс EnemySpawn (классы, которые

описывались выше). На основании передаваемых типов данный класс создает приватные переменные _logRule и _enemySpawnRule – у которых вызываются необходимые методы. Сам экземпляр класса TemplateMediator создается в классе Game (классе, который хранит и инициализирует основные модули программы) – см. Рис «Класс Game».

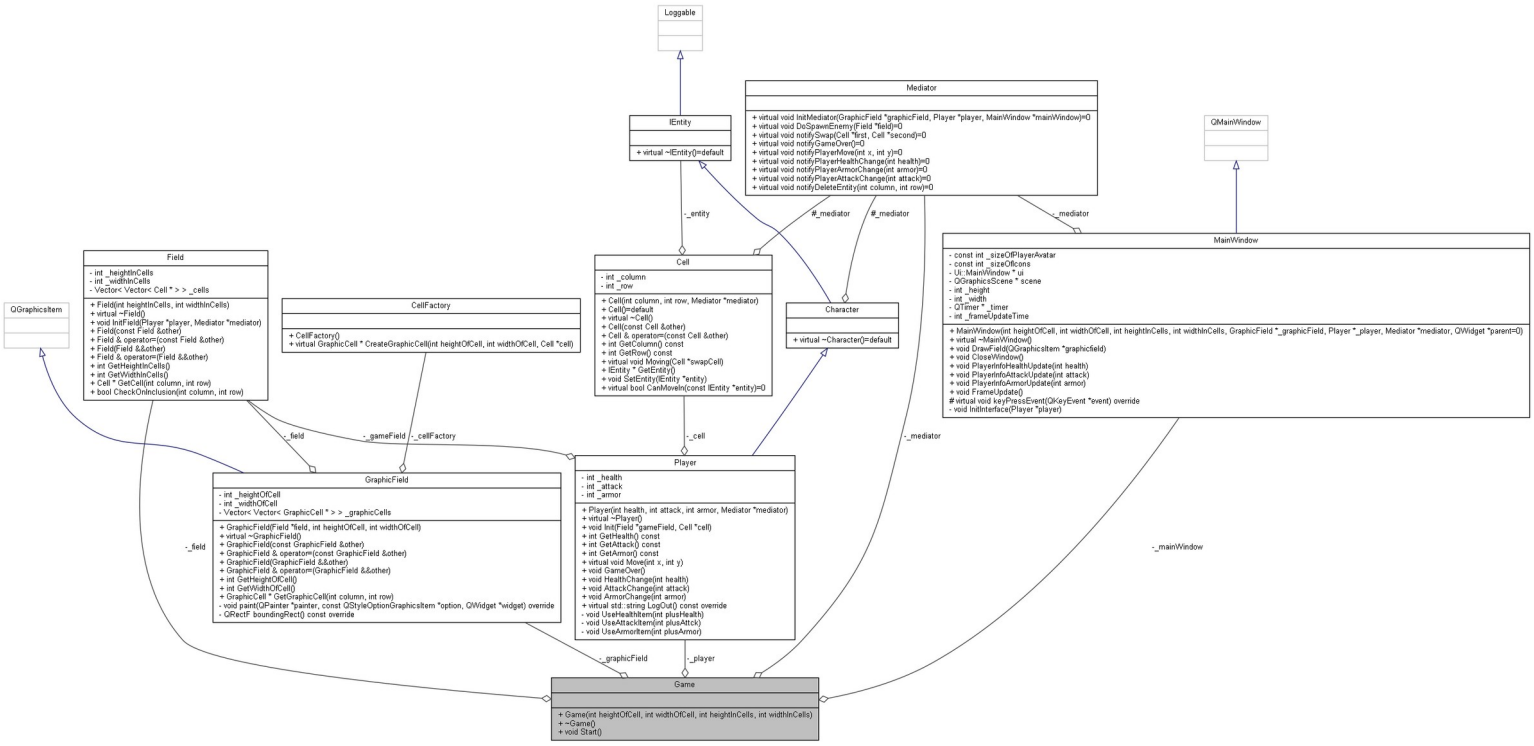

Рис. «Класс Game»

Выводы:

В ходе выполнения лабораторной работы были изучены и применены шаблонные классы. Написано несколько шаблонных классов для игры и немного переделан класс Медиатора, добавлен новый шаблонный класс Медиатора, который в своем шаблоне принимает классы-правила. Созданы UML-диаграммы по проделанной работе.