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

ОТЧЕТ

по практической работе №3

по дисциплине «Объектно-ориентированное программирование»

Тема: Логическое разделение классов.

Студент гр. 8383 _____ Федоров И.А.

Преподаватель _____ Жангиров Т.Р.

Санкт-Петербург

2020

Цель работы.

Реализовать и разработать набор классов для взаимодействия пользователя с юнитами и базой.

Постановка задачи.

Основные требования:

- Должен быть реализован функционал управления юнитами.
- Должен быть реализован функционал управления базой.
- *Реализован паттерн "Фасад", через который пользователь управляет программой.
- *Объекты между собой взаимодействуют через паттерн "Посредник".
- *Для передачи команд используется паттерн "Команда".
- *Для приема команд от пользователя используется паттерн "Цепочка обязанностей".

Выполнение работы.

Примечание* : файлы относящиеся к данной работа: Mediator.h, Command.h, BaseCommand.h, ClickedCommand.h, AttackCommand.h, UnitInfoCommand.h, GameFacade.h, CreateBaseCommand.h. Были реализованы паттерны "Фасада", "Команда", "Цепочка обязанностей". Паттерн "Посредник" был сделан **только** для одного класса - AttackMediator.h, для остальных взаимосвязей он не реализован.

Был создан класс **`AttackMediator`** (унаследованный от интерфейса **`IMediator`**), который был выполнен по принципу паттерна "Посредник". К данному классу имеют доступ все созданные юниты, т.к. они хранят указатель на него. Атака юнитов производится через данный класс-посредник в следующей последовательности:

1. Юнит в своем методе `attack()` передает указатель на себя, и координаты клетки для атаки.

2. Посредник проверяет дистанцию атаки (хватает ли юниту дальности для атаки), получает от атакующего атрибуты для атаки (характеристики типа урон, негативное воздействие и т.д.)
3. Затем класс-посредник проверяет, кого требуется атаковать (базу или другого юнита), проверяет "отношения" объектов между собой (одной из характеристик юнитов и базы является поле **`relation`** - их принадлежность вражеской или союзной команде, атаковать союзного юнита/базу нельзя).
4. Проверяет умер ли атакуемый объект, и вызывает метод удаления у поля в случаи уничтожения.

Был создан класс **`GameFacade`**, реализованный по принципу паттерна "Фасад". Пользователь взаимодействует с программой через него. Класс-фасад создает нужные команды, формирует для них нужный запрос (через класс **`Event`** из прошлой работы), получает выходной результат и передает сообщение в метод вывода класса **`Game`** (например характеристик базы).

Полная UML диаграмма представлена на рис. 1.

Для передачи команд были созданы классы-команды по принципу паттерна "Команда":

1. **`Command`** - абстрактный класс, от которого наследуются все остальные команды и содержащий единственный метод **`execute`**.
2. **`CreateBaseCommand`** - команда, обрабатывающая запрос "создать базу", передает вызов получателю - **`BaseConstructor`**, указатель на которого хранит в поле.
3. **`AttackCommand`** - команда атаки, передает запрос юниту, указатель на которого хранит.
4. **`MoveCommand`** - аналогично `AttackCommand` передает юниту координаты назначения для перемещения.

5. **`BaseCommand`** - команда, обрабатывающая запросы для базы, в зависимости от запроса вызывает у базу либо метод `getInfo` для получения информации, либо метод `createUnit`.
6. **`FieldCommand`** - использует метода класс `GameField` (хранится в поле), создает нужные команды и передает им указатель на объект в случае необходимости. В случаи ошибочного запроса сам отправляет ошибку.
7. **`ClickedCommand`** - команда обрабатывающая клик пользователя по полю и передающая запрос классу `FieldCommand`.

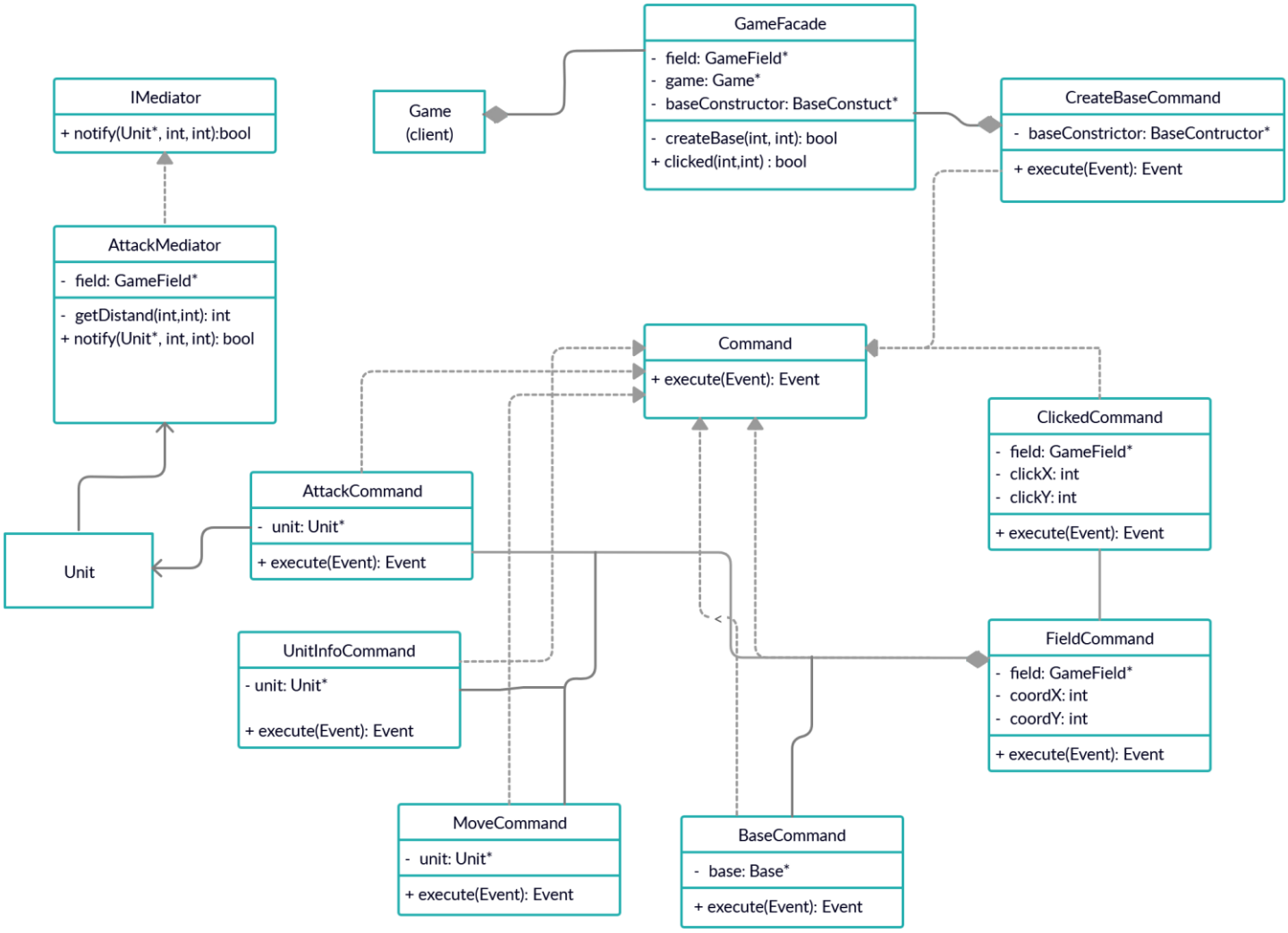

Рисунок 1 - UML диаграмма.

Прием команд от пользователя реализован по принципу паттерн "Цепочка обязанностей", основные цепочки:

1) Атака юнита: `GameFacade->ClickedCommand->FieldCommand->AttackCommand`. Команда-поле передает команде-атаки указатель на атакующего юнита (координаты которого получила от команды-клика) и координаты атакуемой клетки.

2) Передвижение юнита: `GameFacade->ClickedCommand->FieldCommand->MoveCommand`. Команда-поле получает от команды-клика координаты направления и передает указатель на юнита и координаты клетки, в которую нужно переместиться.

3) Информация о базе: `GameFacade->ClickedCommand->FieldCommand->BaseCommand`. Аналогично предыдущем, поле-команда по полученным координатам передает указатель на базу.

4) Создание юнита: `GameFacade->ClickedCommand->FieldCommand->BaseCommand`.

5) Создание базы: `GameFacade->CreateBaseCommand`. Клиент (в данном случае фасад) может отправлять запрос любому из объектов цепочки, не обязательно первому, в данном случае команда создания базы вызывается сразу.

На рисунках 2-4 приведены некоторые примеры работы..

Рисунок 2 - По щелчку на базу появляется информация и кнопки.

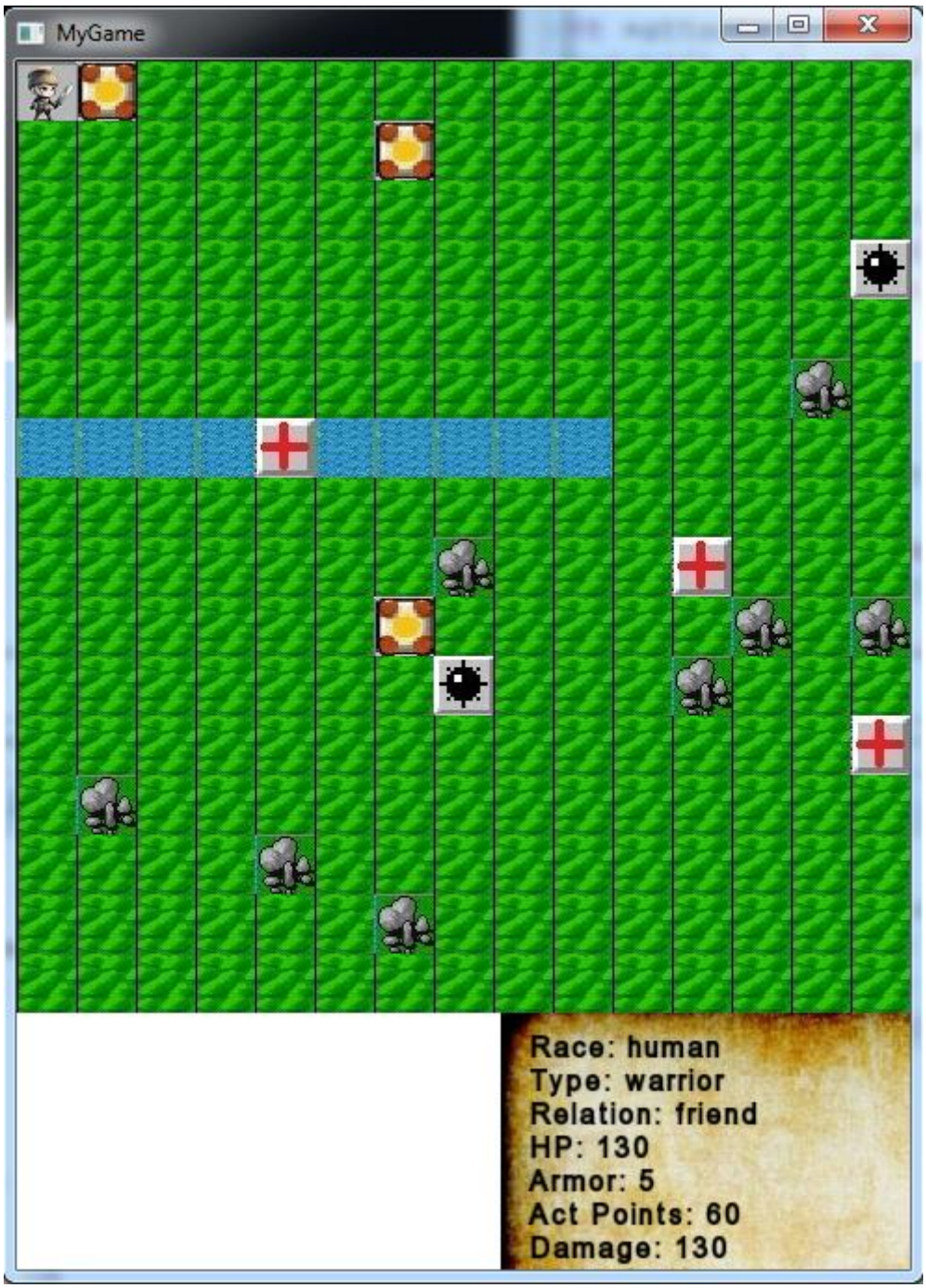

Рисунок 3 - По щелчку по юниту появляются его данные, после следующего щелчка юнит перемещается/атакует (т.к. теперь он активный)

Рисунок 4 - У вражеской базы нет кнопок.

Выводы.

В ходе выполнения лабораторной работы были реализованы наборы классов для взаимодействия пользователя с юнитами и базой. Были реализованы паттер "Фасад", "Команды", "Цепочка обязанностей". Был применен паттерн "Посредник" в единственном случае.