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

ОТЧЕТ

по лабораторной работе №2
по дисциплине «Объектно-ориентированное программирование»
Тема: Интерфейсы, динамический полиморфизм

Студент гр. 1304 ___ Байков Е.С.

Преподаватель ___ Жангиров Т.Р.

Санкт-Петербург
2022

Цель работы.

Реализовать систему событий. Событие - сущность, которая срабатывает при взаимодействии с игроком. Должен быть разработан класс интерфейс общий для всех событий, поддерживающий взаимодействие с игроком. Необходимо создать несколько групп разных событий реализуя унаследованные от интерфейса события (например, враг, который проверяет условие, будет ли воздействовать на игрока или нет; ловушка, которая безусловно воздействует на игрока; событие, которое меняет карту; и.т.д.). Для каждой группы реализовать конкретные события, которые по-разному воздействуют на игрока (например, какое-то событие заставляет передвинуться игрока в определенную сторону, а другое меняет характеристики игрока). Также, необходимо предусмотреть событие “Победа/Выход”, которое срабатывает при соблюдении определенного набора условий.

Реализовать ситуацию проигрыша (например, потери всего здоровья игрока) и выигрыша игрока (добрался и активировал событие “Победа/Выход”)

Требования.

- Разработан интерфейс события с необходимым описанием методов
- Реализовано минимум 2 группы событий (2 абстрактных класса наследников события)
- Для каждой группы реализовано минимум 2 конкретных события (наследники от группы события)
- Реализовано минимум одно условное и безусловное событие (условное - проверяет выполнение условий, безусловное - не проверяет).
- Реализовано минимум одно событие, которое меняет карту (меняет события на клетках или открывает расположение выхода или делает какие-то клетки проходимыми (на них необходимо добавить события) или не непроходимыми
- Игрок в гарантированно имеет возможность дойти до выхода

Описание архитектурных решений и классов.

Был создан интерфейс *IEvent*, содержащий в себе три виртуальных метода и виртуальный деструктор. Метод *invoke()* – метод для запуска события, *clone()* – определяет, каким образом будет происходить копирование события например при копировании клеток, и метод *isActive()* – проверяющий является ли метод активным. Для реализации интерфейса были созданы классы *PlayerEvent* и *FieldEvent*, которые содержат *protected* поля *player* и *_fieldChanger* соответственно.

Рассмотрим наследников класса *PlayerEvent*:

- *HealthEvent*: класс хранит в себе поле *_isActive* которое по умолчанию *true*, с помощью указателя на игрока в *protected* поле своего родителя устанавливает новое значение здоровья игроку, используя его геттеры и сеттеры.
- *ManaEvent*: класс хранит в себе поле *_isActive* которое по умолчанию *true*, с помощью указателя на игрока в *protected* поле своего родителя устанавливает новое значение маны игроку, используя его геттеры и сеттеры.
- *CurseEvent*: класс хранит в себе поле *_isActive* которое по умолчанию *true*, с помощью указателя на игрока в *protected* поле своего родителя устанавливает новое случайное значение расы и/или класса игроку, используя его сеттеры, а также обновляет его здоровье и ману соответственно его классу и расе.
- *DamageCurseEvent*: класс хранит в себе поле *_isActive* которое по умолчанию *true*. Наносит урон игроку, в зависимости от имеющегося у него здоровья. Если здоровье игрока меньше единицы – игра заканчивается.

Рассмотрим наследников класса *FieldEvent*:

- *ChangePassability*: класс хранит в себе поле *_isActive* по умолчанию равное *true*, с помощью указателя на класс *FieldChanger* изменяет проходимость поля вокруг персонажа в радиусе 1 клетки (если до реагирования события клетки были заблокированы то они станут свободными после вызова *invoke()*).
- *ReverseField*: класс хранит в себе поле *_isActive* по умолчанию равное *true*, с помощью указателя на класс *FieldChanger* изменяет оккупированные клетке на неисследованные и наоборот, так же и переопределяет события находящиеся на них.

Был также реализован класс *ExitEvent*, который хранится на клетке выхода и может быть вызван только в том случае, если он наследуется от *FieldEvent* и *PlayerEvent*, в связи с этим возникла необходимость решения проблемы ромбовидного наследования с помощью виртуального наследования. Внутри класса проверяется достаточность свободных клеток, для активации выходной клетки. Если условие соблюдено, у игрока вызывается метод *win()*.

У игрока появился наблюдатель, который следит за его здоровьем и выигрышной позицией и в зависимости от этого меняет состояние игры и пишет сообщение о выигрыше или проигрыше.

Для заполнения клеток событиями была реализована фабрика событий, которая в зависимости от того, какому классу наследует конкретное событие, создает определенное событие и заполняет его *protected* поле с помощью сеттера родителя.

Для контроллера была реализована дополнительная опция *research*, отвечающая за исследование неизвестной клетки. При исследовании меняется тип клети с *UNEXPLORED* на *FREE* и запускается событие которое «скрывалось» в этой клетке.

Демонстрация работы программы и тестирование.

При исследовании клетки, можно получить различные сообщения о пребывании персонажа на клетке.

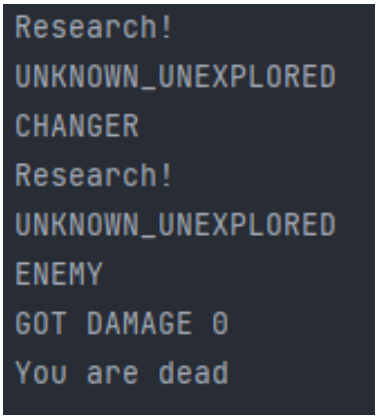

Рисунок 1 — Различное реагирование на проверку клетки.

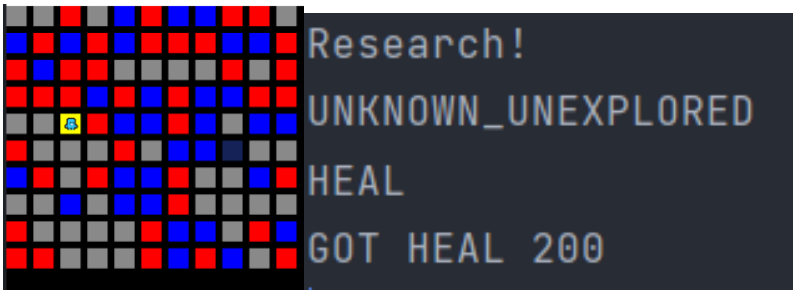

Рисунок 2 — Реакция клеток на исследования игроком.

При нажатии кнопки «R» персонаж производит исследование клетки, на которой он стоит. При открытии 3 или более зеленых клеток и возможен выход из программы со статусом выигрыш, либо же, если получить много урона и будет запущен выход из игры.

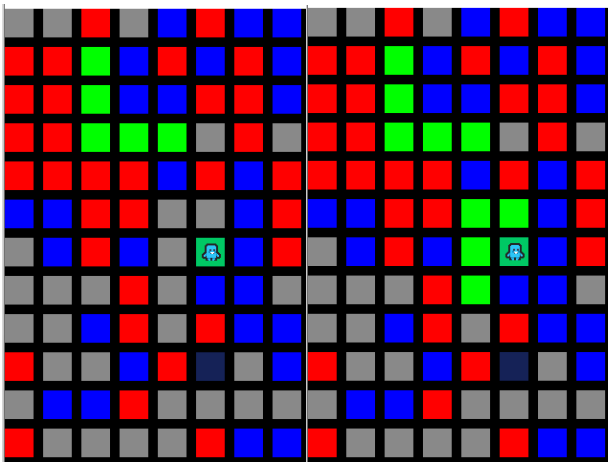

Рисунок 3 — Изменение поля при исследовании игроком.

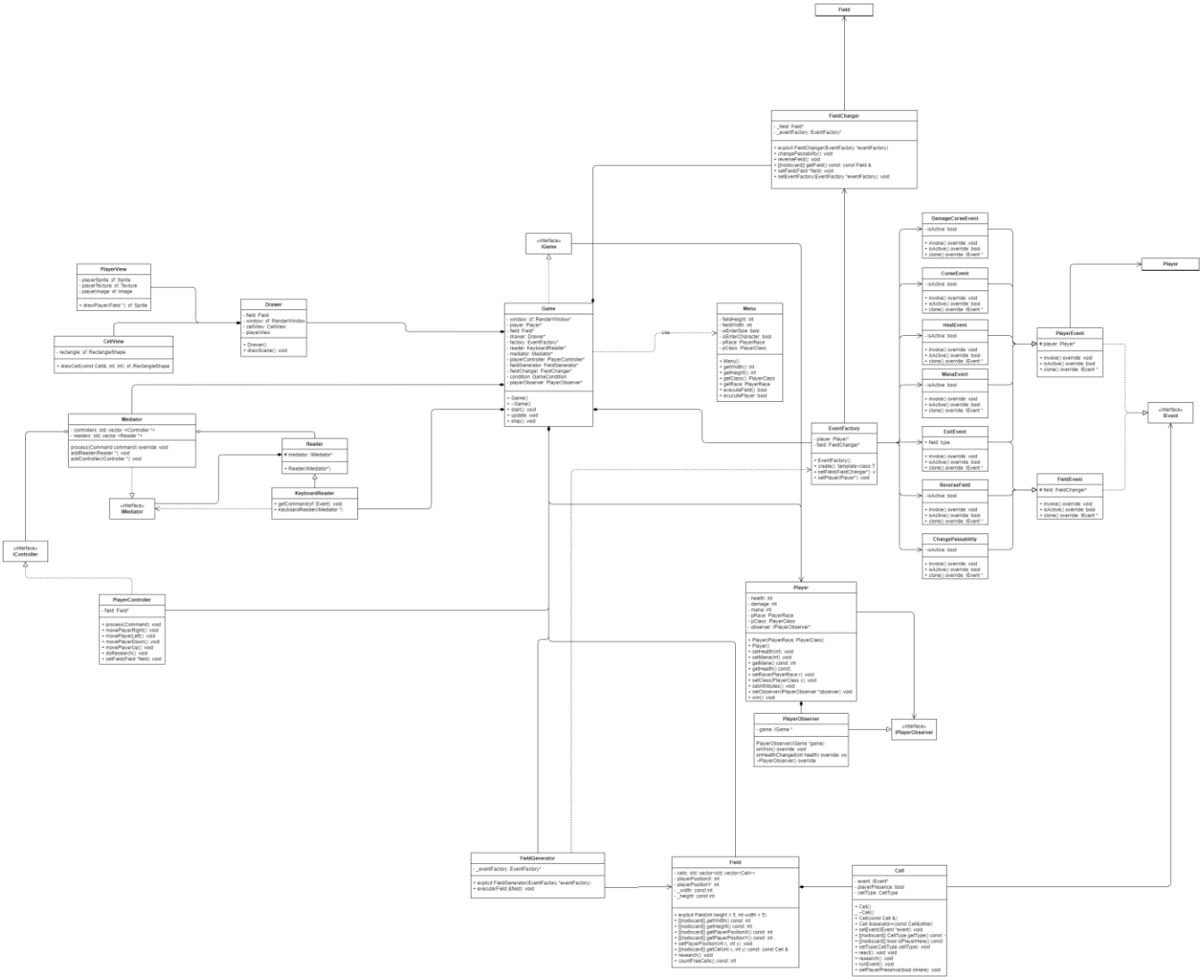

Рисунок 6 — UML-диаграмма классов.

Вывод.

Реализованы события реагирования клетки на пребывания персонажа на ней. Реализованы события, изменяющие поле, а также позволяющие проиграть и закончить игру выигрышем.