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

ОТЧЕТ

по лабораторной работе №1
по дисциплине «Объектно-ориентированное программирование»
Тема: Интерфейсы, динамический полиморфизм

Студент гр. 1383 _____ Куликов М.Д.

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”)

Задание.

-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

Выполнение работы.

Был разработан интерфейс события, от него унаследованы классы групп событий, от них унаследованы непосредственно события. Такая иерархия классов сделана для того, чтобы можно было обращаться к интерфейсу по одному и тому же методу, вне зависимости от типа события на клетке. Поле заполняется с помощью нового класса, взаимодействующим с игровым полем.

ICell_Event – интерфейс события с одним виртуальным методом virtual void interact (Player* hero, int player_x_coord, int player_y_coord).

MapChange – класс группы событий, связанных с изменением игрового поля. Наследуется от интерфейса события и содержит в себе поля с игровым полем, его высотой и шириной, которые будут использоваться в производных классах.

BreakWalls – класс события, которое уничтожает стены в радиусе 1 клетки от игрока, на клетках уже находились события, так что добавлять их отдельно не требуется.

Методы класса:

1)Конструктор, принимающий игровое поле, его длину и ширину и инициализирующий поля класса.

2)Переопределен метод interact. В этом классе он инициализирует поля, соответствующие координатам игрока, и вызывает метод break_walls_around.

3)Метод break_walls_around , изменяющий тип клеток на Free в радиусе 1 клетки от игрока.

Show_Exit – класс события, которое делает выход видимым для игрока и вместо символа F он будет отображаться символом E.

Методы класса:

1)Конструктор, принимающий игровое поле, его длину и ширину и инициализирующий поля класса.

2)Переопределен метод interact. В этом классе он вызывает метод set_exit_ shown_flag.

3)Метод set_exit_shown_flag, устанавливающий соответствующий флаг в 1 со всех клетках на поле, если до этого он был нулем. Этот флаг используется в классе FieldView для выбора символа для отображения пользователю. По умолчанию выход находится в правом верхнем углу поля.

Stats_Change – класс группы событий, связанных с увеличением характеристик игрока. Он содержит в себе поле, ответственное за число, на которое увеличится соответствующая характеристика игрока.

Increase_Health, Increase_Frost_Armor, Increase_Fire_Armor, Increase_Agility – классы событий, ответственные за увеличение характеристик игрока. Их конструкторы инициализирует псевдослучайной величиной единственное поле класса. Величина зависит от ее полезности для игрока. Увеличение брони и ловкости – от 1 до 2. Увеличение здоровья – от 1 до 3. В переопределенном методе interact вызывается метод, изменяющий соответствующие характеристики игрока.

Trap – класс группы событий, представляющих собой ловушку двух разных типов с разными типами урона игроку. Он содержит в себе поле, ответственное за силу ловушки.

Fire_Trap – класс события огненной ловушки, наносящий от 1 до 8 урона игроку.

Методы класса:

1)Конструктор, инициализирующий поле, ответственное за случайную силу ловушки.

2)Переопределенный метод interact, в котором сначала проверяется с помощью метода игрока avoid, увернется игрок от ловушки или нет (шанс зависит от ловкости игрока), а потом наносится урон игроку, учитывая его сопротивляемость к огню. Если сопротивляемость игрока больше чем урон ловушки, то событие на него никак не воздействует.

Frost_Trap - класс события морозной ловушки, наносящий от 1 до 8 урона игроку.

Методы класса:

1) Конструктор, инициализирующий поле, ответственное за случайную силу ловушки.

2)Переопределенный метод interact, в котором сначала проверяется с помощью метода игрока avoid, увернется игрок от ловушки или нет (шанс зависит от ловкости игрока), а потом наносится урон игроку, учитывая его сопротивляемость к холоду. Если сопротивляемость игрока больше чем урон ловушки, то событие на него никак не воздействует.

Field_Event_Filler – класс, заполняющий каждую клетку поля случайными событиями и имеющий одно поле, в котором хранится игровое поле.

Методы класса:

1)Конструктор, принимающий игровое поле и инициализирующее поле класса.

2)Метод fill_the_field(), в котором на поле на каждую клетку ставится псевдослучайное событие с помощью функции rand().

Также был изменен класс **Controller**. Теперь в его конструкторе также выполняется заполнение игрового поля событиями с помощью Field_Event_Filler и появился новый метод is_player_dead_or_won, который проверяет здоровье и позицию игрока и в случае победы или смерти вызывает методы FieldView для вывода информации игроку, после чего завершает работу программы.

Тестирование.

Результаты тестирования представлены на рисунках.

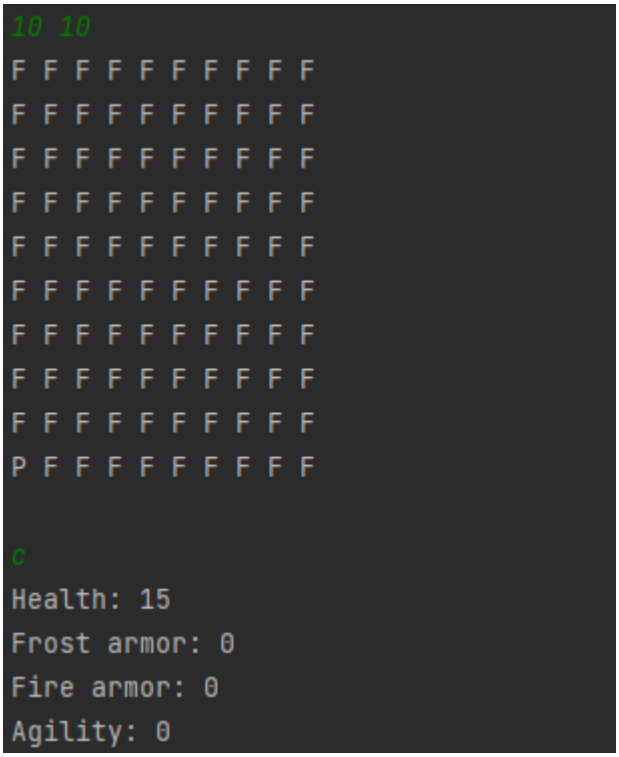

Рисунок 1 – Начальные характеристики игрока.

После одного шага вправо сработало событие увеличения сопротивления игрока к холоду и его броня увеличилась на 2, что показано на рисунке 2.

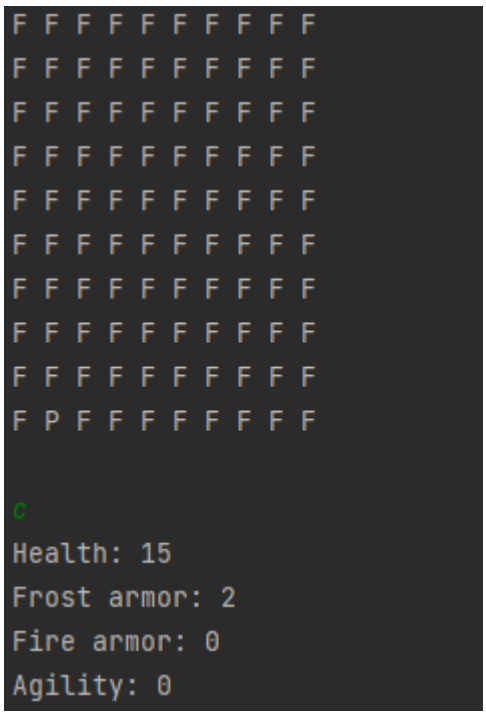

Рисунок 2 – Характеристики после одного шага.

После второго шага либо сработало событие уничтожения стен, либо игрок увернулся от ловушки, так как ничего не произошло. После третьего шага сработало событие увеличения ловкости игрока, что показано на рисунке 3.

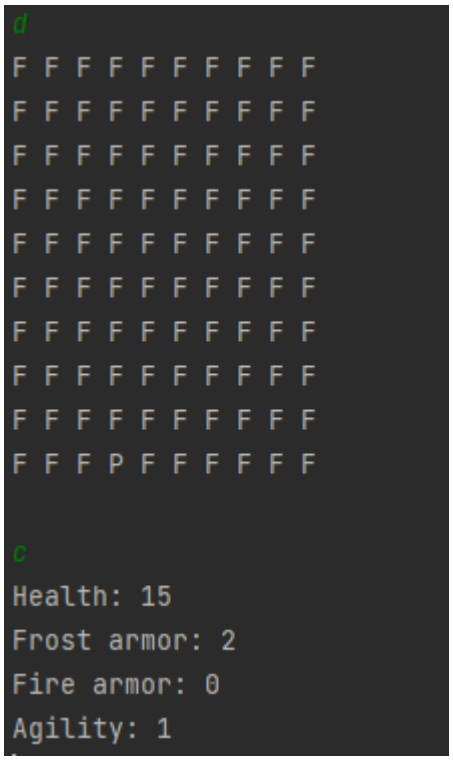

Рисунок 3 – Характеристики после еще двух шагов.

На рисунке 4 показано срабатывание события, которое показывает игроку расположение выхода.

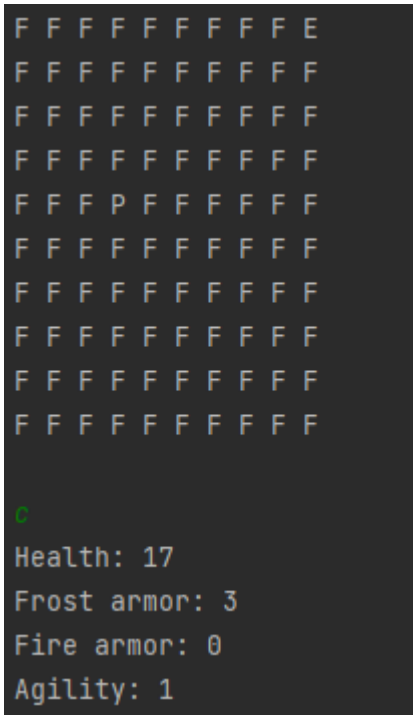

Рисунок 4 – Расположение выхода

На рисунке 5 показано успешное завершение игры.

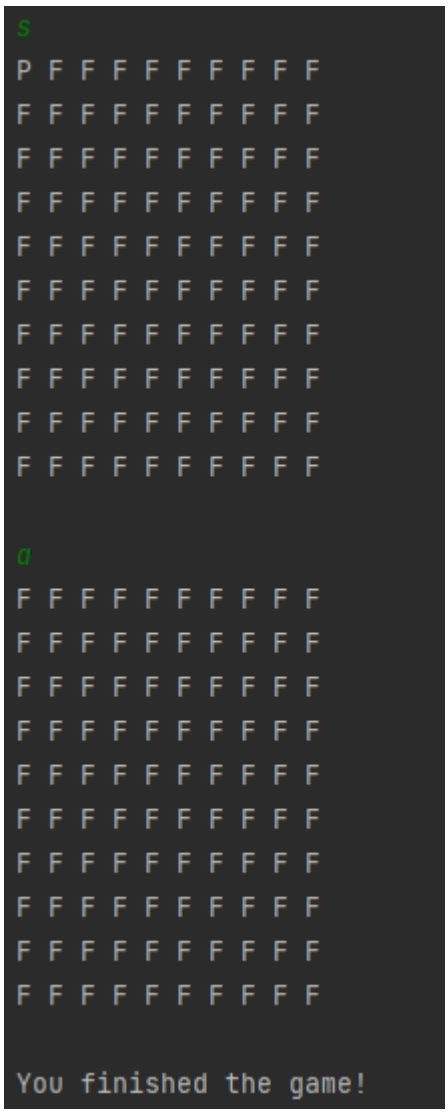

Рисунок 5 – Завершение игры.

Выводы.

Был реализован интерфейс события и были унаследованы от него группы событий и сами события. Также была реализована возможность победы в игре при выходе с карты или проигрыша при смерти игрока.

ПРИЛОЖЕНИЕ А

UML – ДИАГРАММА