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

ОТЧЕТ
по лабораторной работе №2

по дисциплине «Объектно-ориентированное программирование»

Тема: Интерфейсы, динамический полиморфизм

Студент гр. 1381 Возмитель В. Е

Преподаватель Жангиров Т.Р.

Санкт-Петербург
2022

Цель работы.

Изучить понятия интерфейсы, виртуальные функции и реализовать динамический полиморфизм.

Задание.

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

Выполнение работы.

Используется стандартная библиотека C++ и её заголовочные файлы *iostream*, *cstdlib, vector, string, random.*

Были реализованы следующие классы:

1. Определяется класс-интерфейс события *Event*, от которого наследуются классы-группы событий.

Реализуются виртуальные методы класса с модификатором доступа *public*:

- *virtual void execute(void* obj)* = 0; – метод срабатывания события.

1. Определяется абстрактный класс *Waste_events*, от которого наследуются классы событий, связанные с изменением игрока.

Реализуются методы класса с модификатором доступа *public*, аналогичные классу Event.

2. Определяется абстрактный класс *Game_events*, от которого наследуются классы событий, связанные с правилами игры.

Реализуются методы класса с модификатором доступа *public*, аналогичные классу Event.

3. Определяется абстрактный класс *Profit_events*, от которого наследуются классы событий, связанные с правилами игры.

Реализуются методы класса с модификатором доступа *public*, аналогичные классу Event.

4. Определяется класс *Event_Hp*, объектом которого является событие, увеличивающее здоровье игрока.

   Реализуются методы класса с модификатором доступа *public*:

• *void execute(void* obj) override* – метод, в который передаётся указатель на объект класса игрока и который добавляет к здоровью игрока +20.

1. Определяется класс *Event_Enemy,* объектом которого является событие столкновения с врагом.

   Реализуются методы класса с модификатором доступа *public*:

• *void execute(void* obj) override* – метод, в который передаётся указатель на объект класса игрока и который отнимает от здоровья игрока 50 *hp*, отнимает от *armor* игрока 15 и прибавляет 15 к *cache* игрока(что необходимо для победы в игре)

1. Определяется класс *Event_Win* – наследуемый класс от абстрактной группы *Game_events,* объектами которого являются события, отслеживающие победу и поражение в игре.

   Реализуются метод класс с модификатором *public*:

   - *void execute(void *ptr) override;* - в этом классе переопределенная виртуальная функция проверяет условие на изменение характеристики игрока, при выполнении условия – присваивание полю игрока *end* значение *true.*

     Само условие состоит из проверки поля *gold* на достижение значения 40 и проверки поле игрока *hp (*Должно быть больше 0*)*

1. Определяется класс *Event_Damage* - наследуемый класс-ловушка абстрактной группы *Waste_events*. Содержит конкретное событие – изменение характеристики игрока.

Реализованы методы класса с модификатором доступа *public:*

- *void execute(void *ptr) override* - метод, в который передаётся указатель на объект класса игрока и который отнимает от *hp* игрока 25 единиц. При наличии *armor* у игрока здоровье игрока уменьшается на меньшую величину, но также уменьшается значение *armor*.

1. Определяется класс *Event_Hp*, объектом которого является событие, увеличивающее *armor(броню)* игрока.

Реализуются методы класса с модификатором доступа *public*:

- *void execute(void* obj) override* – метод, в который передаётся указатель на объект класса игрока и который добавляет к броне игрока +20 единиц.

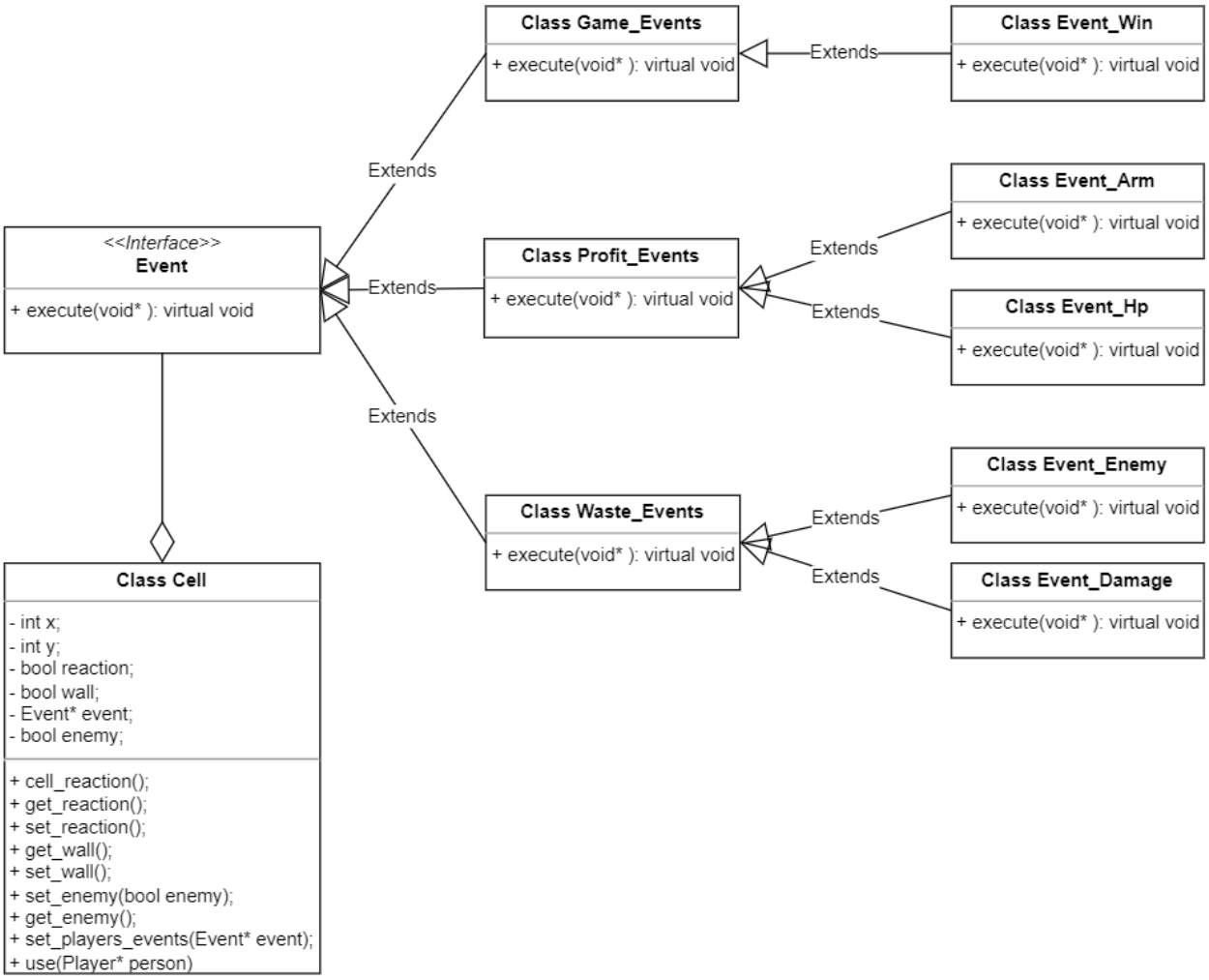
1. Архитектура программы.

В каждой клетке (*объект класса Cell*) хранится указатель на событие (*объект класса Event*). При генерации игрового поля (*класс Field*) для некоторых клеток случайным образом генерируется событие.

При перемещении игрока по полю и попадании его на клетку с событием, вызывается метод клетки *use,* который в свою очередь вызывает метод события *execute ().*

Условие победы в игре зависит от характеристик игрока и от его положения на поле (игрок должен дойти до крайней нижней клетки). Только в этом случае игрок считается победителем.

Uml диаграмма.



Результат работы программы.

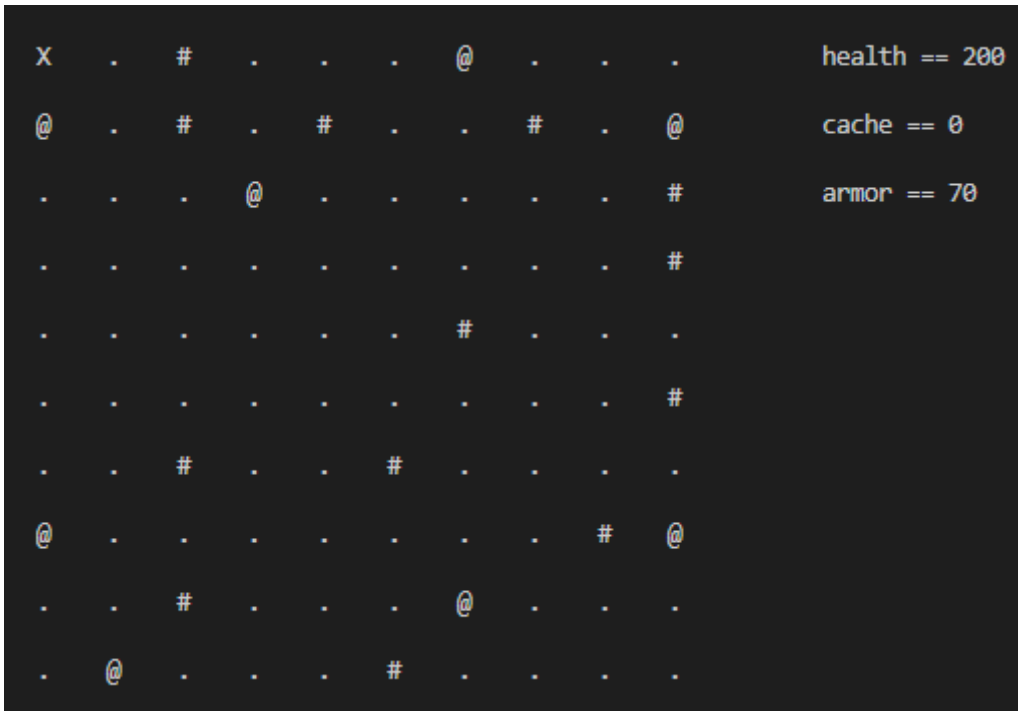


Рисунок №1.

Выводы.

Были изучены приемы ООП, отработаны навыки создания интерфейсов, виртуальных функций и реализации динамического полиморфизма.