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

ОТЧЕТ №2

по лабораторной работе

по дисциплине «Объектно-ориентированное программирование»

Тема: Интерфейсы, динамический полиморфизм

Студент гр. 1381 Дудко М.А.

Преподаватель Жангиров Т.Р.

Санкт-Петербург
2022

Цель работы.

Ознакомиться с работой основных принципов ООП. Написать примитивную игру на языке C++.

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

Примечания:

- Классы событий не должны хранить никакой информации о типе события (никаких переменных и функций дающие информации о типе события)
- Для создания события можно применять абстрактную фабрику/прототип/строитель

Выполнение работы.

Сначала был создан интерфейс Event_Interface с чисто виртуальным методом effect, которому передается ссылка на игрока, и деструктором. Затем был унаследован 3 вида событий от интерфейса. И уже от этих 3 групп наследовал конкретный события. В которых чисто виртуальный метод переопределяется с помощью final и взаимодействует с игроком через его методы. В классе cell я создал ссылку на интерфейс some_event. В методах some_event_new был создан и инициализирован some_event через указатель на одно из конкретных событий. Так же есть методы some_event_is, которые через dynamic_cast проверяет указатель на какое событие лежит в some_event. Если лежит верный указатель, то метод вернет этот указатель но событие, если же указатель на иное событие, то null. А в поле я уже задаю события в клетках через new, и проверяю какое событие лежит в клетке через dynamic_cast.

Интерфейс Event_Interface хранит чисто виртуальный метод effect.

```
virtual void effect(Player& player)=0;
```

Событие Event_Damage_Arrow уменьшает поле health класса Player на единицу.

```
void Event_Damage_Arrow::effect(Player& player){
    player.damage();
}
```

Событие Event_Damage_Skull обнуляет поле health класса Player.

```
void Event_Damage_Skull::effect(Player& player){
    player.death();
}
```

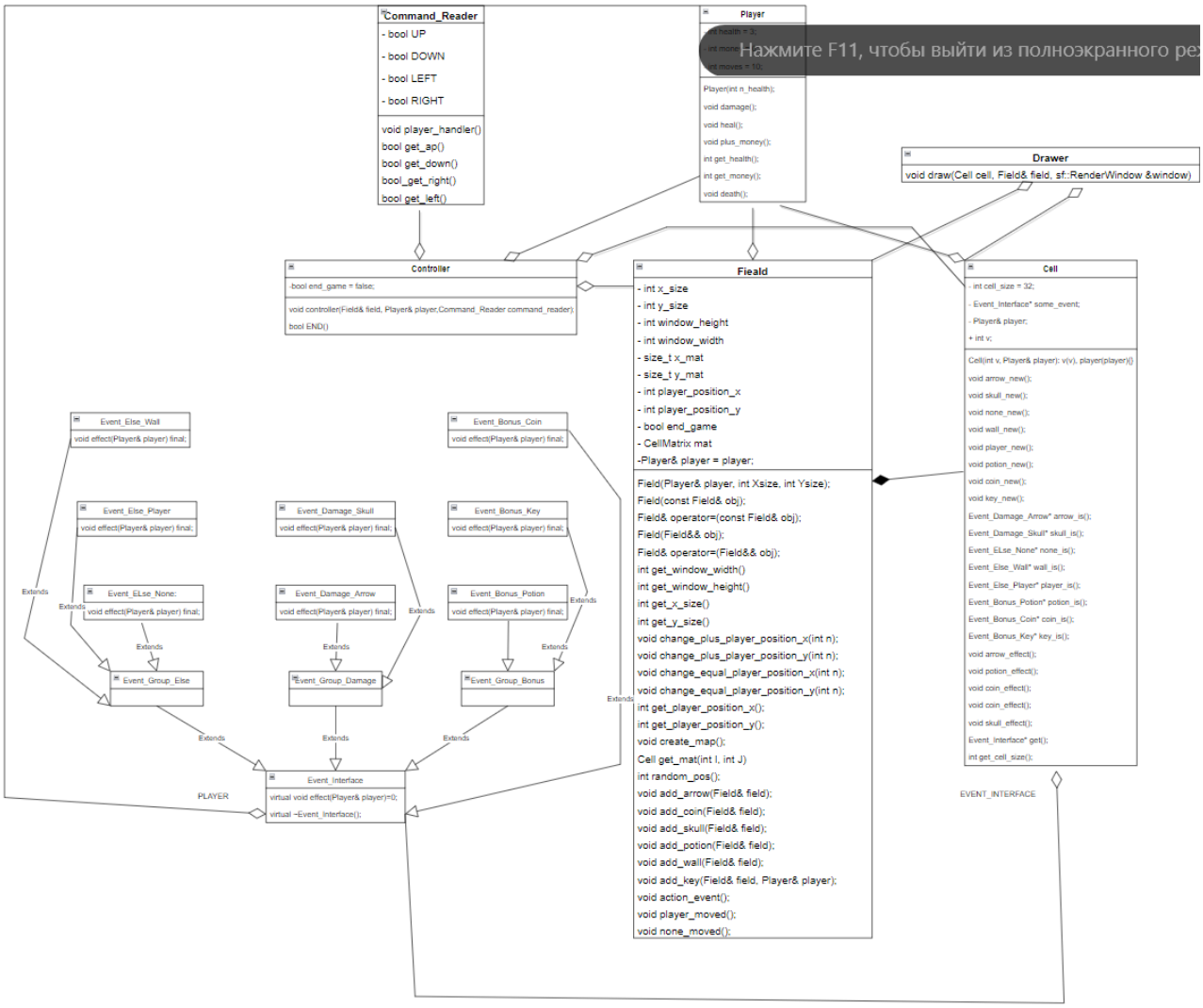
Событие Event_Bonus_Potion увеличивает поле health класса Player на единицу.

```
void Event_Bonus_Potion::effect(Player& player){
player.heal();
}
```

Событие Event_Bonus_Coin увеличивает поле money класса Player на единицу.

```
void Event_Bonus_Coin::effect(Player& player) {
player.plus_money();
}
```

Игрок должен собрать 5 событий типа Coin, что бы появилось событие Key, которое является выходом.


Command_Reader
- bool UP
- bool DOWN
- bool LEFT
- bool RIGHT
void player_handler()
bool get_up()
bool get_down()
bool get_right()
bool get_left()
Player
Player(int n_health);
void damage();
void heal();
void plus_money();
int get_health();
int get_money();
void death();
Drawer
void draw(Cell cell, Field& field, sf::RenderWindow &window)
Controller
-bool end_game = false;
void controller(Field& field, Player& player,Command_Reader command_reader);
bool END()
Fieald
- int x_size
- int y_size
- int window_height
- int window_width
- size_t x_mat
- size_t y_mat
- int player_position_x
- int player_position_y
- bool end_game
- CellMatrix mat
-Player& player = player;
Field(Player& player, int Xsize, int Ysize);
Field(const Field& obj);
Field& operator=(const Field& obj);
Field(Field&& obj);
Field& operator=(Field&& obj);
int get_window_width()
int get_window_height()
int get_x_size()
int get_y_size()
void change_plus_player_position_x(int n);
void change_plus_player_position_y(int n);
void change_equal_player_position_x(int n);
void change_equal_player_position_y(int n);
int get_player_position_x();
int get_player_position_y();
void create_map();
Cell get_mat(int I, int J)
int random_pos();
void add_arrow(Field& field);
void add_coin(Field& field);
void add_skull(Field& field);
void add_potion(Field& field);
void add_wall(Field& field);
void add_key(Field& field, Player& player);
void action_event();
void player_moved();
void none_moved();
Cell
- int cell_size = 32;
- Event_Interface* some_event;
- Player& player;
+ int v;
Cell(int v, Player& player): v(v), player(player){}
void arrow_new();
void skull_new();
void none_new();
void wall_new();
void player_new();
void potion_new();
void coin_new();
void key_new();
Event_Damage_Arrow* arrow_is();
Event_Damage_Skull* skull_is();
Event_ELse_None* none_is();
Event_Else_Wall* wall_is();
Event_Else_Player* player_is();
Event_Bonus_Potion* potion_is();
Event_Bonus_Coin* coin_is();
Event_Bonus_Key* key_is();
void arrow_effect();
void potion_effect();
void coin_effect();
void coin_effect();
void skull_effect();
Event_Interface* get();
int get_cell_size();
Event_Else_Wall
void effect(Player& player) final;
Event_Else_Player
void effect(Player& player) final;
Event_ELse_None:
void effect(Player& player) final;
Event_Group_Else
Event_Damage_Skull
void effect(Player& player) final;
Event_Damage_Arrow
void effect(Player& player) final;
Event_Group_Damage
Event_Bonus_Coin
void effect(Player& player) final;
Event_Bonus_Key
void effect(Player& player) final;
Event_Bonus_Potion
void effect(Player& player) final;
Event_Group_Bonus
Event_Interface
virtual void effect(Player& player)=0;
virtual ~Event_Interface();
Extends
PLAYER
EVENT_INTERFACE