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: Создание классов, конструкторов и методов

Студент гр. 1381 Возмитель В. Е

Преподаватель Жангиров Т.Р.

Санкт-Петербург
2022

Цель работы.

Изучить понятия класса, его методов и полей, научиться реализовывать простейшие классы и осуществлять межклассовые отношения.

Задание.

Реализовать прямоугольное игровое поле, состоящее из клеток. Клетка - элемент поля, которая может быть проходима или нет (определяет, куда может стать игрок), а также содержит какое-либо событие, которое срабатывает, когда игрок становится на клетку. Для игрового поля при создании должна быть возможность установить размер (количество клеток по вертикали и горизонтали). Игровое поле должно быть зациклено по вертикали и горизонтали, то есть если игрок находится на правой границе и идет вправо, то он оказывается на левой границе (аналогично для всех краев поля).

Реализовать класс игрока. Игрок – сущность, контролируемая пользователем. Игрок должен иметь свой набор характеристик и различный набор действий (например, разные способы перемещения, попытка избежать событие, и так далее).

Требования:

- Реализован класс игрового поля
- Для игрового поля реализован конструктор с возможностью задать размер и конструктор по умолчанию (то есть конструктор, который можно вызвать без аргументов)
- Реализован класс интерфейс события (в данной лабораторной это может быть пустой абстрактный класс)
- Реализован класс клетки с конструктором, позволяющим задать ей начальные параметры.
- Для клетки реализованы методы реагирования на то, что игрок перешел на клетку.

- Для клетки реализованы методы, позволяющие заменять событие. (То есть клетка в ходе игры может динамически меняться)
- Реализованы конструкторы копирования и перемещения, и соответствующие им операторы присваивания для игрового поля и при необходимости клетки
- Реализован класс игрока минимум с 3 характеристиками. И соответствующие ему конструкторы.
- Реализовано перемещение игрока по полю с проверкой допустимости на переход по клеткам.

Примечания:

- При написании конструкторов учитывайте, что события должны храниться по указателю для соблюдения полиморфизма
- Для управления игроком можно использовать медиатор, команду, цепочку обязанностей

Выполнение работы.

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

Были реализованы следующие классы:

1. *class Event(){}.* Класс предназначен для интерфейса событий (в данной работе это пустой абстрактный класс)
2. *class Player {}.* Класс Игрока.

   Реализованы методы класса с модификатором доступа *private:*
   - *int hp* – health points
   - *int armor* - armor
   - *int gold* – money

   Реализованы методы класса с модификатором доступа *public:*
   - *Player(): hp(100), gold(100), armor(100){}* – конструктор класса

1. *class CommandReader{}.* Класс для считывания с консоли команд.

Реализованы методы класса с модификатором доступа *public:*

- *void read(){}* – считывание команды
- *char get_enter(){}* – геттер для получения команды

Реализованы методы класса с модификатором доступа *private:*

- *char enter* - ввод

1. *class Cell {}*. Определяется класс клетки, объектами которого являются клетки поля.

Реализованы методы класса с модификатором доступа *private:*

- *int x, y* – координаты клетки
- *bool reaction* – реагирование
- *bool wall* – стена

Реализованы методы класса с модификатором доступа *public:*

- *Cell(int x, int y, bool reaction = false, bool wall = false): x(x), y(y), reaction(reaction), wall(wall){}* – конструктор класса
- *void cell_reaction()* – метод реагирования на то, что игрок перешел на клетку.
- *void cell_reaction()* – метод реагирования на то, что игрок перешел на клетку.
- *void set_reaction(bool reaction)* – сеттер для установки состояния
- *void cell_wall()* – метод, для реагирования на присутствие стены
- *bool get_wall()* – геттер для получения информации о присутствие стены
- *void set_wall(bool wall)* – сеттер для установки стены
- *void change_Condition()* – метод, позволяющий заменять событие
- *set_event(Event *new_event);* - позволяет задать событие в клетке, хранящееся в поле *Event* event.*

1. *class Field {}*. Определяется класс *Field*, объектом которого является игровое поле.

Реализуются методы класса с модификатором доступа *public*:

- *int height, length* – высота и ширина поля

- *std::vector<std::vector<Cell>> array* – двумерный вектор игрового поля, состоящий из объектов класса *Cell*
- *int x, y* – координаты игрока
- *Field(int height, int length) : height(height), length(length){}* – конструктор класса
- *Field(){}* – конструктор по умолчанию
- *Field(const Field &field){}* – конструктор копирования
- *Field (Field&& field){}* – конструктор перемещения
- *Field &operator= (const Field &field) {}* – оператор присваивания при копировании
- *void GoRight(){}* – метод для изменения координат относительно движения
- *void GoDown(){}* – метод для изменения координат относительно движения
- *void GoLeft(){}* – метод для изменения координат относительно движения
- *void GoUpt(){}* – метод для изменения координат относительно движения

1. *class PrintField : public Field{}*. Класс для отображения на консоли поля с клетками. Наследуется от *class Field{}*.

   Реализуются методы класса с модификатором доступа *public*:

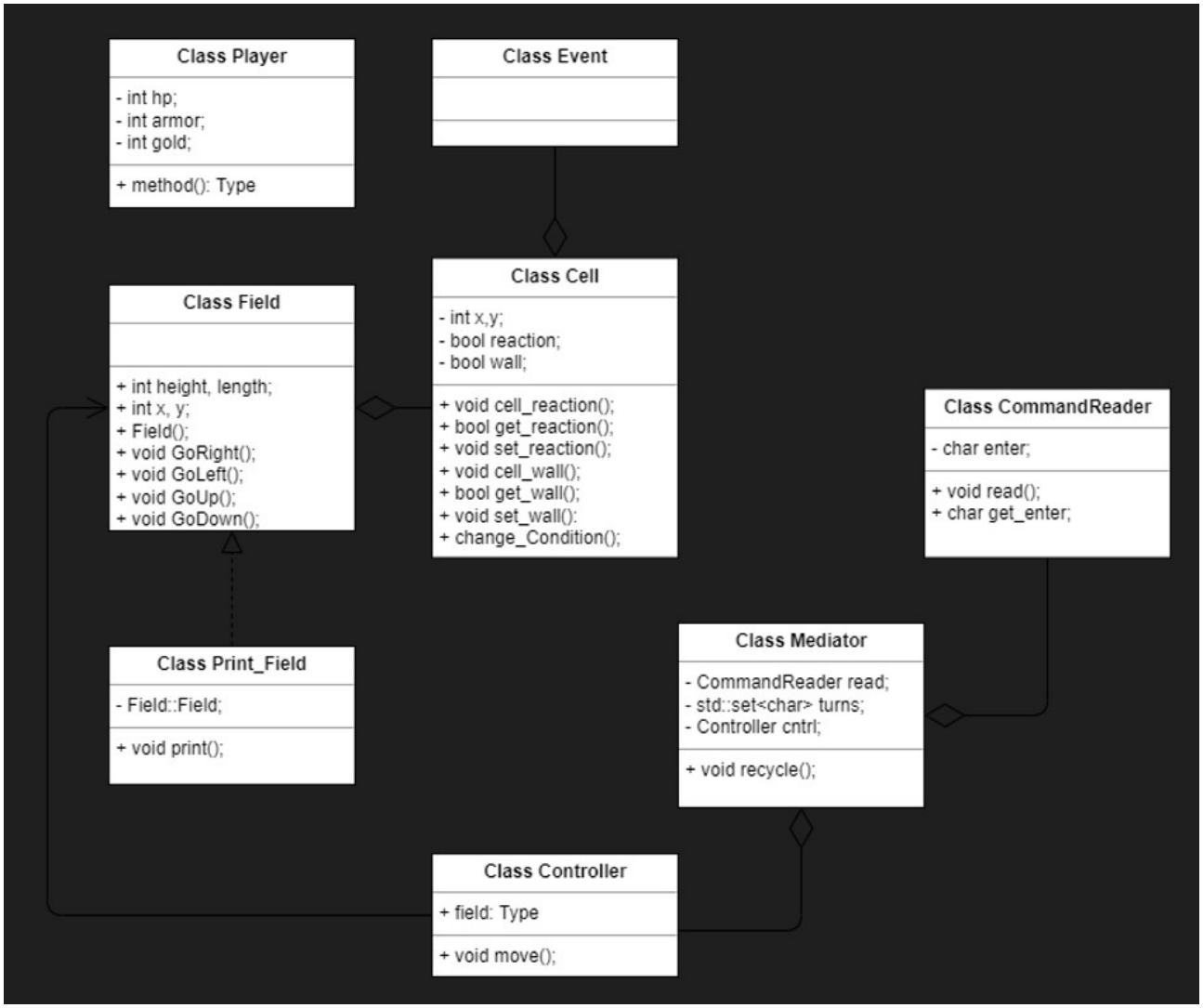
   - *void print(){}* – отображение игрового поля в консоли

1. *class Controller{}*. В данном классе на данный момент вызывается функция *void move (),* которая в зависимости от полученной команды от пользователя, вызывает соответствующую функцию перемещения и отображения поля не экран. Реализовано с помощью *switch*.
2. *Class Mediator{}*. Данный класс «перерабатывает» полученные команды пользователя и затем отправляет результат *class Controller {}*.

   Реализуются методы класса с модификатором доступа *public*:

- *void recycle(char enter)* – «перерабатывает» соответствующие команды

Uml диаграмма.



Выводы.

Были изучены приемы ООП, отработаны навыки создания классов, конструкторов и методов.