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 классов.

Студент гр. 1381 _____ Мамин Р.А.

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.

Выполнение работы.Ход решения:

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

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

a. Реализуются методы класса с модификатором доступа *public*:

- *explicit Cell(TYPE type = empty)* – конструктор класса.
- *TYPE get_obj();* - возвращает тип клетки.
- *void set_obj(TYPE obj);* - меняет тип клетки.
- *bool check_player() const;* - позволяет проверить присутствие игрока в клетке, возвращая поле *bool player_loc*.
- *set_event(Event *new_event);* - позволяет задать событие в клетке, хранящееся в поле *Event* event*.
- *Cell& operator=(const Cell& other)* – оператор присваивания при копировании. Реализован при помощи конструктора копировании, создания временного объекта и метода *swap*.
- *Cell(Cell&& other) noexcept ;* - Реализован при помощи метода swap.

- *Cell& operator=(Cell&& other) noexcept*; - оператор присваивания при перемещении. Реализован при помощи конструктора перемещения и метода *swap*.

b. Реализуются методы класса с модификатором доступа *private*:

- *void set_player();* - позволяет изменить поле *bool player_loc* в зависимости от наличия игрока в клетке.
- *void swap(Cell &other);* - меняет два объекта местами при помощи *std::swap*.

c. Инициализируются поля класса с модификатором доступа *private*:

- *TYPE obj* – тип клетки.
- *bool player_loc* – наличие игрока в клетке.
- *Event *event;* - указатель на объект класса события.

2. Определяется класс *Field*, объектом которого является игровое поле.

a. Реализуются методы класса с модификатором доступа *public*:

- *explicit Field(int width = 10, int height = 10);* - конструктор класса.
- *Field(Field &&other) noexcept* – конструктор перемещения реализован аналогично конструктору класса Cell (как и последующие операторы присваивания с конструкторами данного класса);
- *Field(const Field &other)* – конструктор копирования.
- *Field & operator=(const Field &other)* – оператор присваивания при копировании.
- *Field &Field::operator=(Field &&other) noexcept* – оператор присваивания при перемещении.
- *Field(Field &&other);* - конструктор перемещения.
- *std::vector<int> get_size() const* – возвращает размеры игрового поля.
- *std::vector<std::vector<Cell>> get_field()* возвращает поле *field*.
- *bool move_player(int x, int y)*; - меняет координаты игрока и перемещает его по полю.
- *int get_player_x()* – возвращает координату игрока по горизонтали.

- *int get_player_y()* – возвращает координату игрока по вертикали.

b. Реализуются методы класса с модификатором доступа *private*:

- void swap(Field &other); - меняет два объекта местами при помощи std::swap;
- int get_new_x(int x) const – генерация нового x с учётом цикличности передвижения по полю.
- int get_new_y(int y) const – генерация нового y с учётом цикличности передвижения по полю.
- *void* update_player(*int* prev_x, *int* prev_y); - перемещение игрока путём замены типа соответствующих клеток в поле field.

c. Инициализируются поля класса с модификатором доступа *private*:

- *int* width – ширина поля.
- *int* height; – высота поля.
- *int* player_x – координата игрока по горизонтали.
- *int* player_y – координата игрока по вертикали.
- *std::vector<std::vector<Cell>> field* – двумерный вектор игрового поля, состоящий из объектов класса *Cell*.

3. Определяется класс *Player*, объектом которого является игрок.

a. Реализуются методы класса с модификатором доступа *public*:

- *explicit Player(int health = 100, int armour = 100, int damage = 10);* - конструктор класса *Player*.

b. Инициализируются поля класса с модификатором доступа *private*:

- *int health;* – здоровье игрока.
- *int armour;* – защита игрока.
- *int coins;* – монеты игрока.
- *int damage*; – урон игрока.

4. Определяется класс *CellView*, объектом которого является визуализатор клетки игрового поля.

a. Реализуются методы класса с модификатором доступа *public*:

- *explicit CellView(Cell c)* – конструктор класса CellView.
- *char* get_cell() – возвращает поле cell.

b. Инициализируются поля класса с модификатором доступа *private*:

- char cell – символ клетки, выводимый в консоль.

5. Определяется класс *FieldView*, объектом которого является визуализатор игрового поля.

a. Реализуются методы класса с модификатором доступа *public*:

- *explicit FieldView(Field *field);* - конструктор класса *FieldView*
- *void show_field();* - вывод поля в консоль.

b. Инициализируются поля класса с модификатором доступа *private*:

- *Field *game_field*; - указатель на объект класса *Field*.

6. Определяется абстрактный класс *Event,* объектом которого является событие.

a. Реализуются методы класса с модификатором доступа *public*:

- *virtual void* execute() = 0; - виртуальный метод абстрактного класса.

7. Определяется класс *CommandReader*, объектом которого является считыватель команд. Также в нём реализовано перечисление *MOVES,* содержащее стороны перемещения игрока.

a. Реализуются методы класса с модификатором доступа *public*:

- *explicit ComandReader(int health = 100, int armour = 100, int damage = 100);* - конструктор класса.
- *void set_size();* - запрашивает размеры поля.
- *get_width()* – возвращает ширину поля.
- *get_height()* – возвращает высоту поля.
- *bool set_move();* - позволяет задать поле move исходя из соответствующего введённого символа. Возвращает true, если игрок не остаётся на месте. Иначе – false.

- *MOVES get_move();* - возвращает значения поля move.

b. Инициализируются поля класса с модификатором доступа *private*:

- *int width* – ширина поля.
- *int height* – высота поля.
- *int health;* – здоровье игрока.
- *int armour;* – защита игрока.
- *int damage;* – урон игрока.

8. Определяется класс *Controller*, отвечающий за запуск и контроль процесса игры.

a. Реализуются методы класса с модификатором доступа *public*:

- *explicit Controller(int width = 10, int height = 10, int health = 100, int armour = 100, int damage = 10);* - конструктор класса Controller.
- *void show_field();* - вывод поля в консоль.
- *void move_player(CommandReader::MOVES move);* - передвижение игрока в направлении move.
- *void create_field();* - создание игрового поля.

b. Инициализируются поля класса с модификатором доступа *private*:

- *Player player*; - объект класса игрока.
- *FieldView view*; - объект класса *FieldView*.
- *Field *field;* - указатель на объект класса *Field*.

9. Определяется класс *Mediator*, обеспечивающий общение между классами *ComandReader* и *Controller* .

a. Реализуются методы класса с модификатором доступа *public*:

- *void start();* - запуск игры(инициализация объектов классов *Controller* и *CommandReader*).
- *void show_game();* - визуализация игры(создание, вывод поля, перемещение игрока)

b. Инициализируются поля класса с модификатором доступа *private*:

- *Controller controller*; - объект класса контроллера.
- *CommandReader reader*; - объект класса считывателя команд.

10. Определяется класс *CellType*, от которого наследуются классы конкретных типов клеток .

a. Реализуются методы класса с модификатором доступа *public*:

- *virtual void* execute() = 0; - виртуальный метод абстрактного класса.

Архитектура программы.

В *main.cpp* инициализируется объект класса *Commandreader* (нужен для приёма команд пользователя), запрашивает размер поля, далее с заданными полями инициализируются объекты класса *Controller* (инициализирует внутри себя объекты классов *Player, Field, Fieldview*) и *Mediator*, обеспечивающий общение ридера и контроллера (хранит в себе ссылки на *Controller* и *Commandreader*). Медиатор начинает игру с помощью метода *start_game()*, где, в свою очередь с помощью контроллера за счёт инициализации в нём классов игрока, поля и визуализатора поля создаётся и визуализируется игровое поле, осуществляется передвижение игрока по нему.

Класс *Cell* хранит в себе указатели на объект абстрактного класса *Event* и на объект абстрактного класса *CellType*, от которого наследуются типы клеток – объекты классов *PlayerType, EmptyType, HealType, EnemyType, FixType*.

Класс *Field* хранит в себе двумерный вектор - поле из объектов класса *Cell*, инициализируя внутри себя объекты класса клетки и объекты класса типа клетки. Клетка не знает о существовании поля.

Класс *FieldView* принимает в конструкторе объект класса поля и, отображая его, инициализирует внутри себя класс *CellView*(который отвечает за отображение конкретной клетки в зависимости от принадлежности к наследуемому классу от

CellType), в следствие чего отображаются все клетки поля. Поле не знает о *FieldView*.

Класс *Player* агрегационно связан с классом поля с помощью координат игрока, хранящихся в классе *Field* в качестве полей и изменяющихся методом *move_player()*. Игрок не знает о своих координатах.

Результат работы программы:

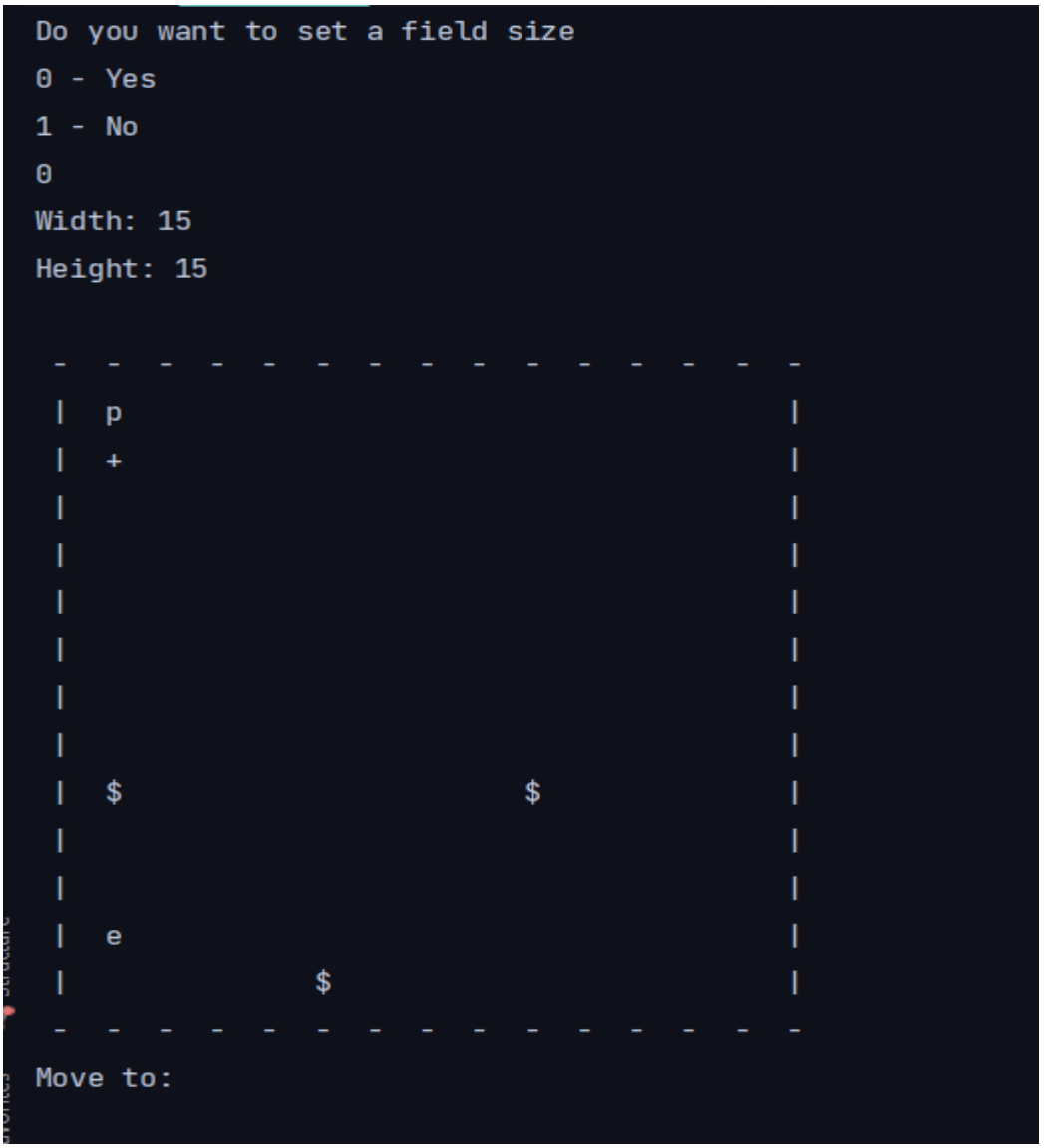

Рис 1. – демонстрация работы программы в терминале Ubuntu.

UML-диаграмма межклассовых отношений:

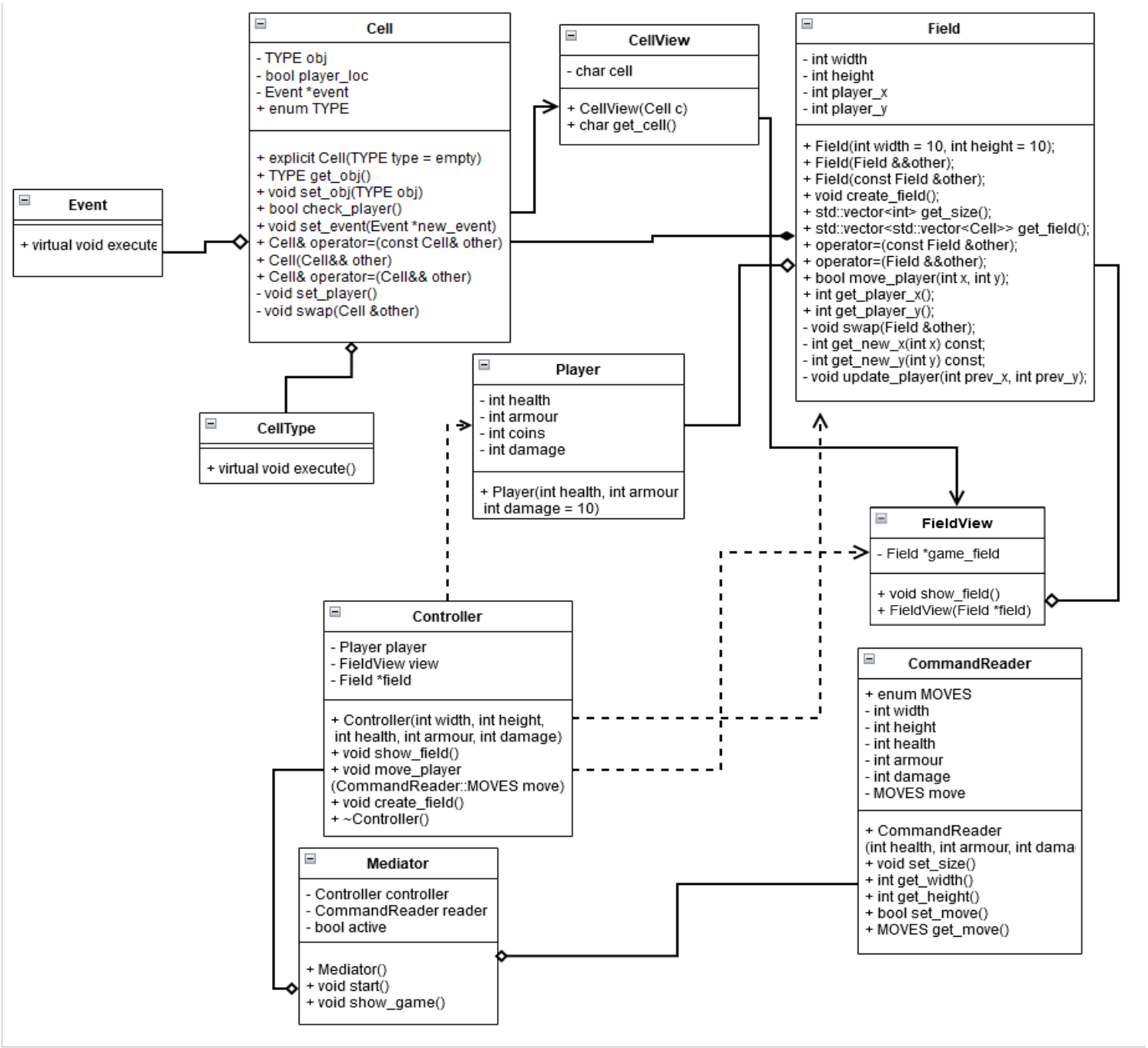

Рис 2. – UML-диаграмма.

Выводы.

Были изучены основы объектно-ориентированного программирования. В ходе лабораторной работы были созданы классы, отвечающие за игрока, клетки поля, поле, их вывод и взаимодействие пользователя с игрой.