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. 1384 _____ Тапеха В. А.

Преподаватель _____ Жангиров Т. Р.

Санкт-Петербург
2022

Цель работы.

Изучить основы объектно-ориентированного программирования, научиться реализовывать простые классы и связывать их между собой.

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.

Примечание:

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

Выполнение работы.

Для выполнения лабораторной работы были созданы классы, отвечающие за игрока, создание клетки поля, создания поля, их вывод и взаимодействие пользователя с программой.

Классы:

1. *Cell* — класс, который определяет клетку. Внутри находится перечисление OBJECT, которое показывает, что расположено на клетке.

Поля:

1) OBJECT obj — показывает состояние клетки (что расположено на ней).

2) Event* event — хранит в себе событие, которое происходит в этой клетке.

Методы:

1) Cell() - конструктор, который инициализирует поле obj с помощью списков инициализации.

2) void set_object(OBJECT obj) — позволяет вручную задать состояние клетки.

3) OBJECT get_obj() const — позволяет получить значение поля OBJECT obj.

4) void set_event(Event* event) — устанавливает событие на клетке.

5) void update(Player& player) — вызывает у события виртуальный метод execute(), который позволяет выполнить некоторое действие.

2. *Field* — класс, определяющий поле.

Поля:

1) std::vector<std::vector<Cell>> field — двумерный массив, хранящий матрицу игрового поля.

2) int height — содержит высоту.

3) int height — содержит ширину.

4) std::pair<int, int> player_location — координаты игрока.

Методы:

1) explicit Field(int width = 10, int height = 10) — конструктор, инициализирующий все поля.

2) Field(const Field &other) — конструктор копирования.

3) Field& operator=(const Field &other) — оператор присваивания. Реализован с помощью конструктор копирования, создания временного объекта и метода swap.

4) Field(Field&& other) — конструктор перемещения. Реализован с помощью метода swap.

5) Field& operator=(Field&& other) — оператор присваивания перемещения. Реализован с помощью метода swap.

6) void make_field() - метод, с помощью которого случайным образом генерируются состояния клеток.

7) void change_player_pos(Player::STEP s) — меняет позицию игрока, но перед этим проверяет допустимость перехода на клетку.

8, 9, 10) int get_height() const, int get_width() const;, std::vector<std::vector<Cell>> get_field() const; - геттеры полей.

11) bool check_cell(Cell cell) const — проверяет клетку на проходимость.

12) void swap(Field& other); - меняет поля местами.

3. *CellView* — определяет, каким образом клетка будет выводиться.

Поля:

1) char cell — символ, на основе типа клетки выводит эту клетку.

Методы:

1) explicit CellView(const Cell& c) — конструктор, который заполняет поле исходя из типа клетки.

2) char get_cell() const — возвращает поле.

4. *FieldView* — выводит поле.

Поля:

1) Field field — хранит игровое поле.

Методы:

1) explicit FieldView(Field& other) — конструктор, заполняющий поле.

2) void print() const — выводит поле.

3) void print_border() const — выводит границу.

5. *Player* — класс игрока. Внутри него определено перечисление, определяющее шаг игрока.

Поля:

1, 2, 3) int health, int damage, int xp - поля определяющие характеристики игрока.

Методы:

1) explicit Player(int health = 1, int damage = 1, int xp = 1) — конструктор, заполняющий характеристики игрока.

2, 3, 4) void set_health(int health), void set_damage(int damage), void set_xp(int xp) — сеттеры для полей.

5, 6, 7) int get_health() const, int get_damage() const, int get_xp() const — геттеры для полей.

6. *Event* — абстрактный класс(интерфейс) — класс, от которого будут наследоваться другие события(понадобится для следующих лабораторных работ). Полей нет.

Методы:

1) virtual void execute(Player& player) = 0 — метод выполняющий некоторое действие, которое зависит от конкретного события.

2) virtual ~Event() = 0; - деструктор.

7. *CommandReader* — класс, считывающий данные.

Поля:

1, 2)int width, height — размеры поля.

3) char choice — символ, определяющий стандартное ли будет игровое поле.

4) Player::STEP step — определяет направление шага игрока.

Методы:

1, 2, 3) void read_step(), void read_size(), void read_char() - считывают данные для полей.

4, 5, 6, 7) int get_height() const, int get_width() const, char get_char() const, Player::STEP get_step() const — геттеры для полей.

8) void check(int& arg) — проверяет корректность ввода ширины и высоты.

8. *Controller* — класс задающий параметры игры и контролирующий ее.

Поля:

1) Field field — содержит игровое поле.

2) FieldView field_view — содержит отображение игрового поля.

Методы:

1) void set_field(int width, int height) — генерирует поле с заданными параметрами.

2) void set_field_standard() - генерирует стандартное поле.

3) void set_step(Player::STEP step) — меняет поле в зависимости от движения игрока.

4) void show_field() - выводит поле.

9. Mediator — класс, связывающий CommandReader и Controller.

Поля:

1) CommandReader input — данные введенные пользователем.

2) Controller game — параметры игры.

Методы:

1) void start() - начинает игру.

Тестирование программы.

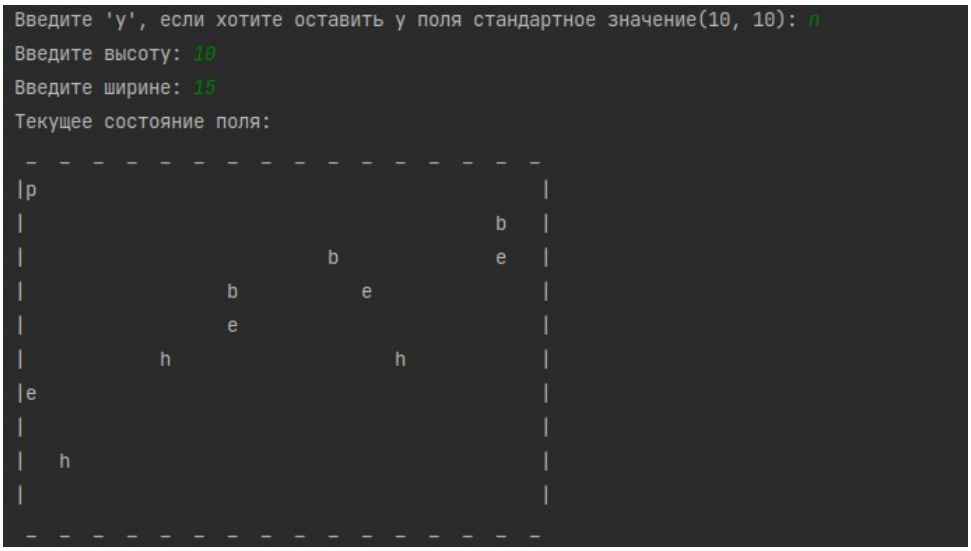

Рис. 1. Тестирование программы

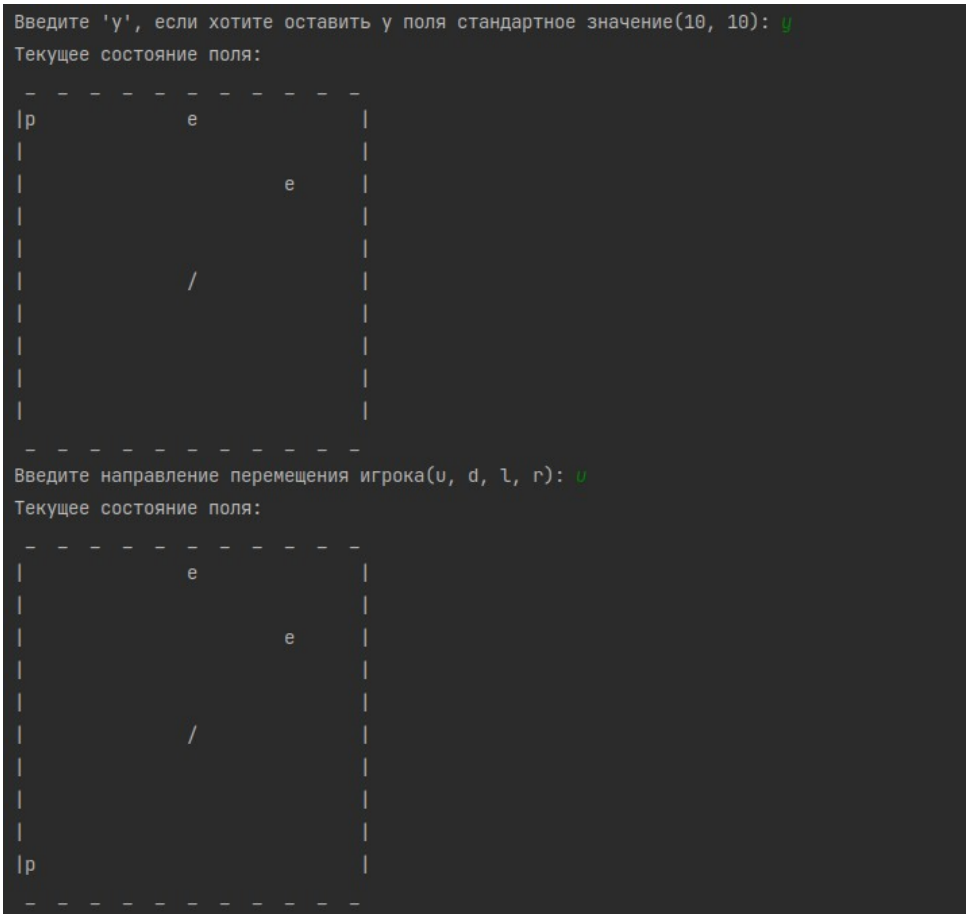

Рис. 2. Тестирование программы

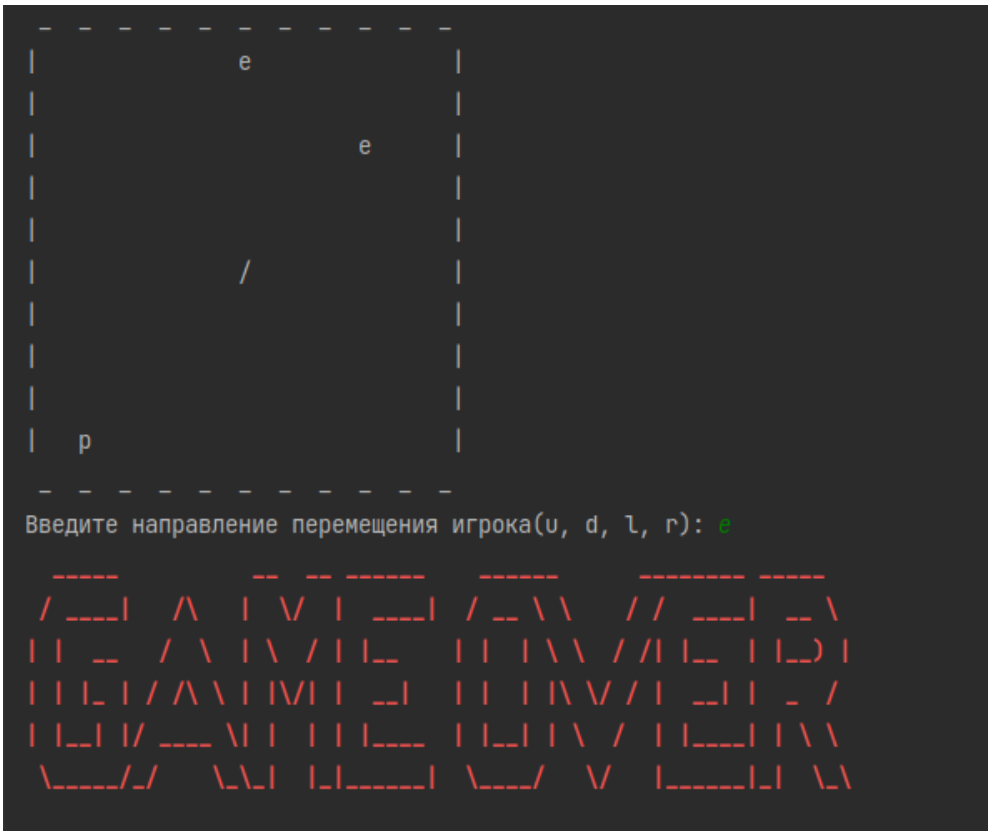

Рис. 3. Тестирование программы.

UML-диаграмма межклассовых отношений.

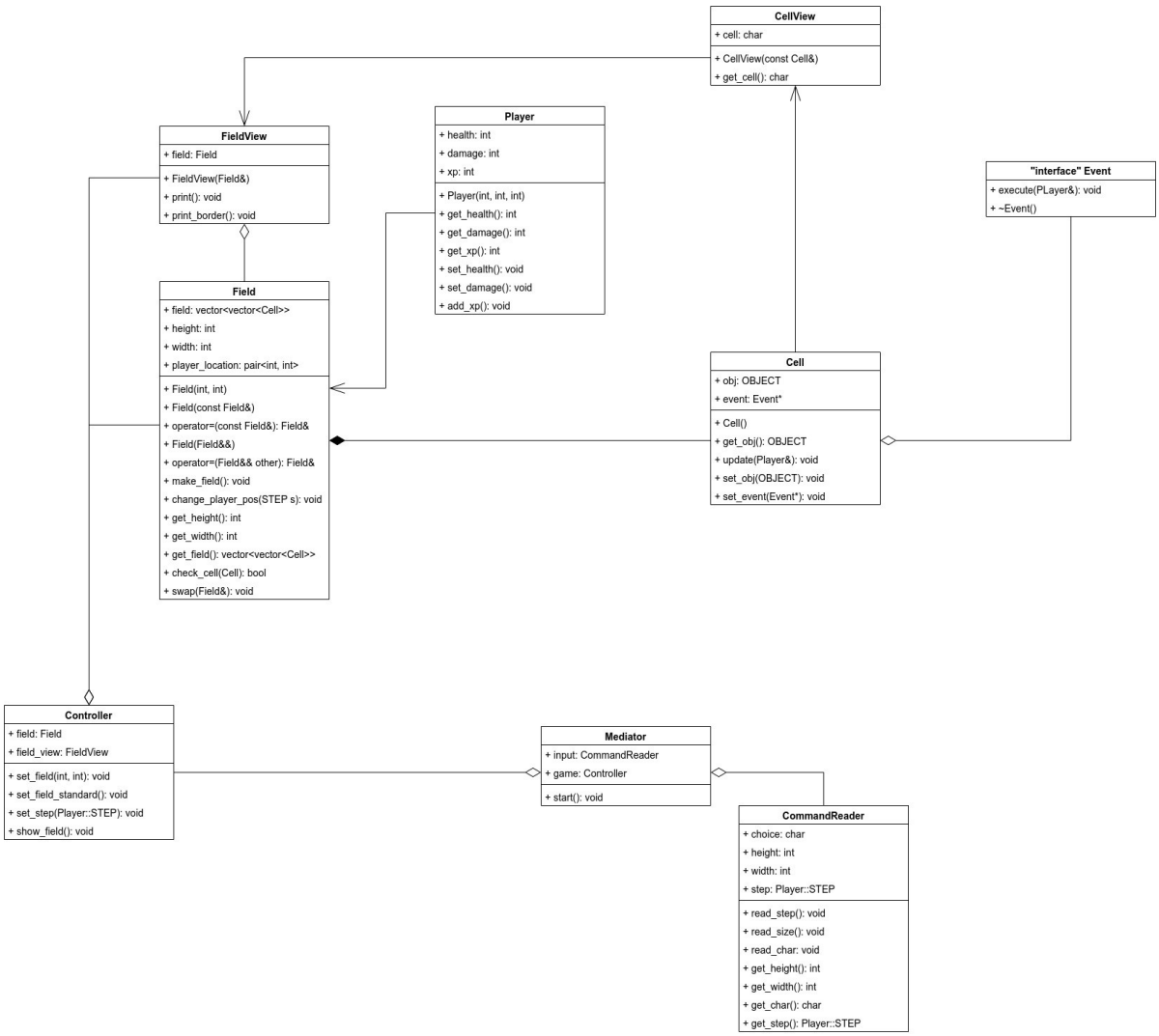

Рис. 4 UML-диаграмма.

Выводы.

Я изучил основы объектно-ориентированного программирования, научился реализовывать простые классы и связывать их между собой.