**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. 1384 |  | Усачева Д.В. |
| Преподаватель |  | Жангиров Т.Р. |
|  |  |  |

Санкт-Петербург

2022

**Цель работы.**

Целью работы является создание основных классов, налаживание их базового взаимодействия и проработка общей архитектуры проекта.

**Задание.**

Нужно 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).

Также нужно 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.

## Реализовать класс игрового поля;

* Для игрового поля реализовать конструктор с возможностью задать размер по умолчанию (то есть конструктор, который можно вызвать без аргументов);

## Реализовать класс клетки с конструктором, позволяющим задать ей начальные параметры;

* Для клетки реализовать методы реагирования на то, что игрок перешел на клетку;

## Для клетки реализовать методы, позволяющие заменять событие (то есть клетка в ходе игры может динамически меняться);

* Реализовать конструкторы копирования и перемещения, и соответствующие им операторы присваивания для игрового поля и при необходимости клетки;

## Реализовать класс игрока минимум с 3 характеристиками и соответствующие ему конструкторы;

* Реализовать перемещение игрока по полю с проверкой допустимости на переход по клеткам;
* Игровое поле должно быть зациклено по вертикали и горизонтали.

# Формат выхода.

## При написании конструкторов нужно учитывать, что события должны храниться по указателю для соблюдения полиморфизма;

* Для управления игроком можно использовать медиатор, команду, цепочку обязанностей.

**Выполнение работы.**

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

Класс Cell описывает клетку игрового поля. Она имеет три параметра, описывающих ее состояние, событие, назначенное на эту клетку, тип клетки, и присутствие игрока в данной клетке. Был создан конструктор класса и заданы методы класса, которые отвечают за получение данных о клетке (TYPE get\_obj() - возвращает тип клетки, bool check\_player() const - позволяет проверить присутствие игрока в клетке,) и методы для её изменения(voidset\_obj(TYPE obj) - меняет тип клетки, set\_event(Event \*new\_event) - позволяет задать событие в клетке, хранящееся в поле Event\* event, void set\_player() - позволяет изменить поле bool player\_loc в зависимости от наличия игрока в клетке). Также были написаны конструктор копирования и оператор присваивания при копировании (Cell(const Cell& other), Cell& operator=(const Cell& other)), конструктор перемещения и оператор присваивания при копировании(Cell(Cell&& other), Cell& operator=(Cell&& other)), которые реализованы при помощи private метода void swap(Cell &other), меняющего два объекта местами при помощи std::swap.

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

В классе CellView описываются символ клетки, выводимый в консоль. В конструкторе класса определяется какой символ будет записан в поле char cell в зависимости от типа клетки. Метод char get\_cell()возвращает значение ранее записанное в поле для дальнейшего вывода на экран.

3. Определяется абстрактный класс Event, объектом которого является событие.

Класс Event описывает событие. По условиям этой лабораторной реализация событий еще не требуется, поэтому на все клетки не назначено никакое событие. Реализуется виртуальный метод абстрактного класса

4. Определяется класс Player, объектом которого является игрок.

Класс Player описывает сущность игрока. У него есть три характеристики: здоровье, урон, очки маны (int health, int mana\_points, int damage). Пока что нет возможности изменять эти характеристики в процессе игры, так как нет в программе не заданы события. Был создан конструктор класса и заданы методы класса, которые отвечают за получение данных о текущем состоянии игрока (int get\_health() const, int get\_mana\_points() const, int get\_damage() const) и методы для дальнейшего изменения этих количественных характеристик (void set\_health(int arg), void set\_mana\_points(int arg), void set\_damage(int arg)).

5. Определяется класс Field, объектом которого является игровое поле.

В классе Field описывается игровое поле. У него заданы поля 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. У него реализованы конструкторы копирования и перемещения и соответствующие им операторы присваивания, как и у клетки обмен двух объектов местами реализован в методе void swap(Field &other). В этом классе заданы методы для изменения (bool move\_player(int x, int y)) и получения текущей информации о местоположении игрока (int get\_player\_x(), int get\_player\_y()). Также методы для возвращения размеров игрового поля (std::vector<int> get\_size() const) и самого поля (std::vector<std::vector<Cell>> get\_field()), представленного как двумерный массив клеток. Далее были описаны private методы: 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.

6. Определяется класс FieldView, объектом которого является визуализатор игрового поля.

В классе FieldView, происходит вывод поля на консоль. Его единственным полем является указатель на объект класса Field - Field \*game\_field. Реализуются методы класса FieldView(Field \*field); - конструктор класса FieldView, void show\_field() - вывод поля в консоль.

7. Определяется класс CommandReader, объектом которого является считыватель команд.

В классе CommandReader описывается считывание команд для выбора размера поля или поля по умолчанию, передвижения игрока по игровому полю. Также в нём реализовано перечисление MOVES*,* содержащее стороны перемещения игрока. Поля заданные в этом классе: int width – ширина поля, int height – высота поля, int health – здоровье игрока, int mana\_points – защита игрока, int damage – урон игрока. Реализуются конструктор и методы 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.

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

В классе Controller определяются поля Player player - объект класса игрока, FieldView view - объект класса FieldView, Field \*field; - указатель на объект класса Field и реализуются конструктор и методы класса void show\_field()- вывод поля в консоль, void move\_player(CommandReader::MOVES move)- передвижение игрока в направлении move, void crate\_field();- создание игрового поля.

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

В классе Mediator заданы поля Controller controller - объект класса контроллера CommandReader reader - объект класса считывателя команд. "Посредник" определяет интерфейс для обмена информацией с объектами полей и координирует их действия. Реализуются методы класса void start()- запуск игры(инициализация объектов классов Controller и

CommandReader) и void show\_game() - визуализация игры(создание, вывод поля, перемещение игрока).

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 c 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.

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 СellView(который отвечает за отображение конкретной клетки), 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.

**ПРИЛОЖЕНИЕ 1**

# UML-диаграмма межклассовых отношений

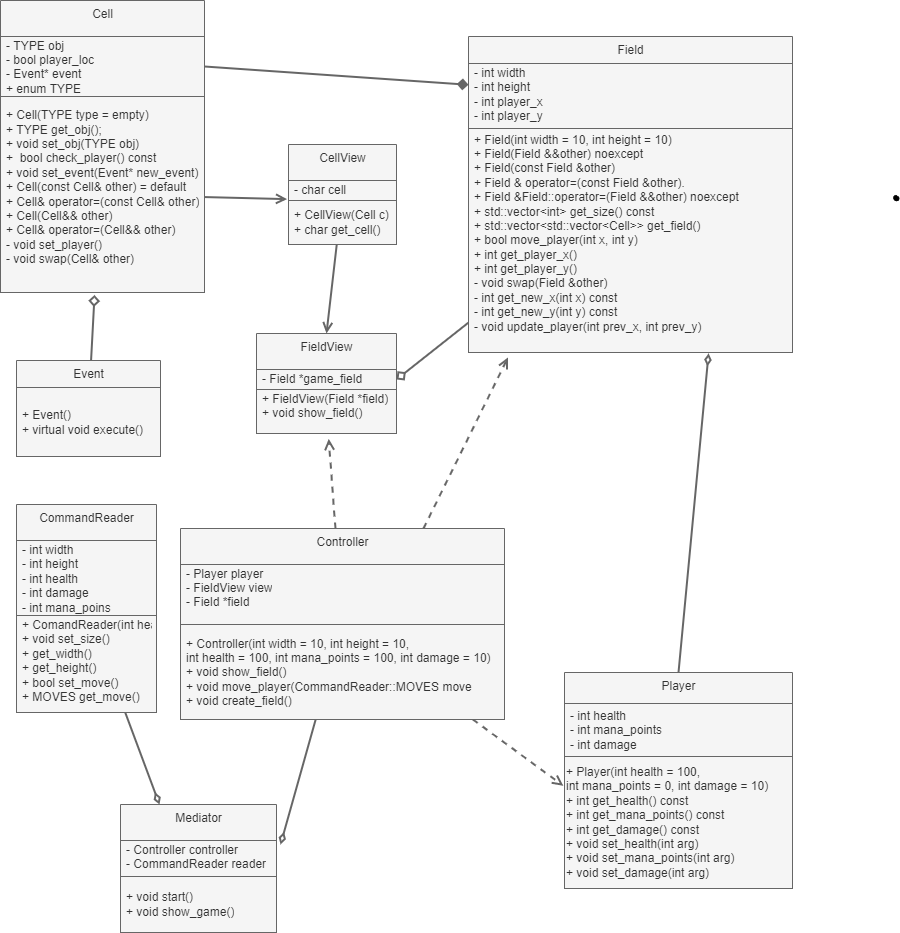


Рис 1. – UML-диаграмма.

**ПРИЛОЖЕНИЕ 2**

Результат работы программы

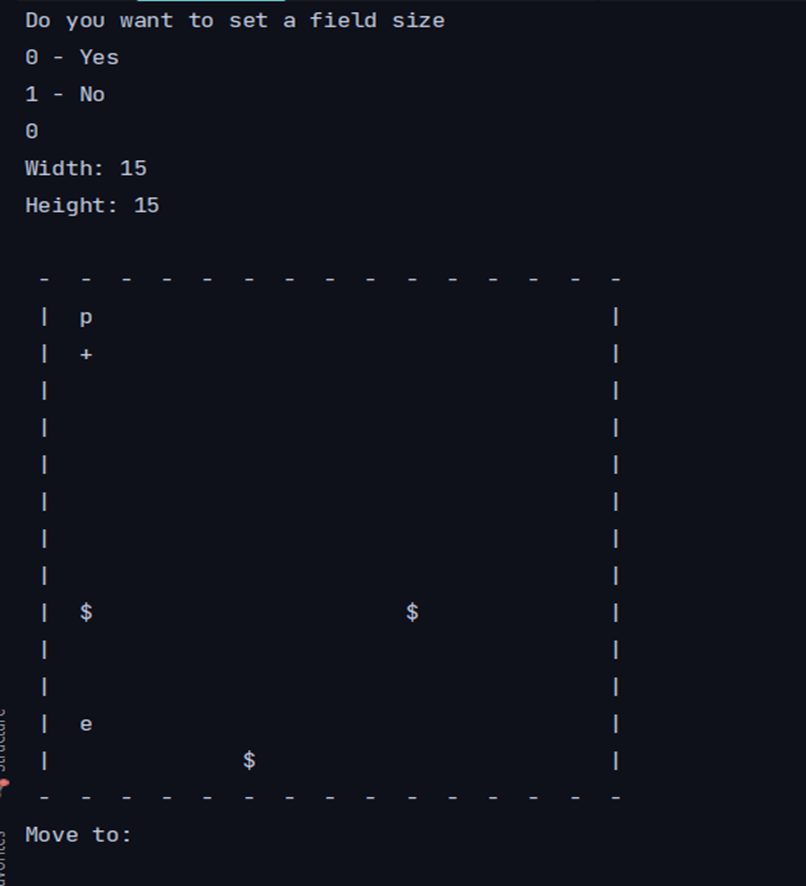


Рис 2. – демонстрация работы программы.

**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.