# 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.

Там же реализовано перечисление типов клеток *enum TYPE*.

* 1. 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*.
  1. 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.*
  1. 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.

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

1. Определяется класс *Player*, объектoм которого является игрок.
   1. Реализуются методы класса с модификатором доступа *public*:

* *explicit Player(int health = 100, int armour = 100, int damage = 10);* - конструктор класса *Player.*
  1. Инициализируются поля класса с модификатором доступа *private*:
* *int health;* – здоровье игрока.
* *int armour;* – защита игрока.
* *int coins;* – монеты игрока.
* *int damage*; – урон игрока.

1. Определяется класс *CellView*, объектом которого является визуализатор клетки игрового поля.
   1. Реализуются методы класса с модификатором доступа *public*:

* *explicit CellView(Cell c)* – конструктор класса CellView.
* *char* get\_cell() – возвращает поле cell.
  1. Инициализируются поля класса с модификатором доступа *private*:
* char cell – символ клетки, выводимый в консоль.

1. Определяется класс *FieldView*, объектом которого является визуализатор игрового поля.
   1. Реализуются методы класса с модификатором доступа *public*:

* *explicit FieldView(Field \*other);* - конструктор класса *FieldView*
* *void show\_field();* - вывод поля в консоль.
  1. Инициализируются поля класса с модификатором доступа *private*:
* *Field \*field\_obj*; - указатель на объект класса *Field*.

1. Определяется абстрактный класс *Event*, объектом которого является событие.
   1. Реализуются методы класса с модификатором доступа *public*:

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

1. Определяется класс *CommandReader*, объектом которого является считыватель команд. Также в нём реализовано перечисление *MOVES,* содержащее стороны перемещения игрока.
   1. 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.
  1. 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.

1. Определяется класс *Controller*, отвечающий за запуск и контроль процесса игры.
   1. 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.
  1. Инициализируются поля класса с модификатором доступа *private*:
* *Player player*; - объект класса игрока.
* *FieldView view*; - объект класса *FieldView.*
* *Field \*field;* - указатель на объект класса *Field.*

1. Определяется класс *Mediator*, обеспечивающий общение между классами *ComandReader* и *Controller* .
   1. 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)
  1. Инициализируются поля класса с модификатором доступа *private*:
* *Controller controller*; - объект класса контроллера.
* *CommandReader reader*; - объект класса считывателя команд.

# 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.