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 класса

Студент гр. 0383:	 Самара Р.Д.
Преподаватель:	Жангиров Т.Р

Санкт-Петербург

2021

Цель работы.

Научиться создавать собственные классы, их конструкторы и деструкторы, конструкторы копирования и перемещения и специальные методы класса.

Задание.

Игровое поле представляет из себя прямоугольную плоскость разбитую на клетки. На поле на клетках в дальнейшем будут располагаться игрок, враги, элементы взаимодействия. Клетка может быть проходимой или непроходимой, в случае непроходимой клетки, на ней ничего не может располагаться. На поле должны быть две особые клетки: вход и выход. В дальнейшем игрок будет появляться на клетке входа, а затем выполнив определенный набор задач дойти до выхода.

При реализации класса поля запрещено использовать контейнеры из stl Требования:

- Реализовать класс поля, который хранит набор клеток в виде двумерного массива.
- Реализовать класс клетки, которая хранит информацию о ее состоянии, а также того, что на ней находится.
 - Создать интерфейс элемента клетки.
- Обеспечить появление клеток входа и выхода на поле. Данные клетки не должны быть появляться рядом.
- Для класса поля реализовать конструкторы копирования и перемещения, а также соответствующие операторы.
 - Гарантировать отсутствие утечки памяти.

Потенциальные паттерны проектирования, которые можно использовать:

• Итератор (Iterator) - обход поля по клеткам и получение косвенного доступа к ним

• Строитель (Builder) - предварительное конструирование поля с необходимым параметрами. Например, предварительно задать кол-во непроходимых клеток и алгоритм их расположения

Выполнение работы.

Порядок выполнения поставленной задачи программой:

- 1 Все файлы программы были разбиты на две директории: include заголовочные файлы классов с разрешением, и src сами классы.
- 2 Реализован класс Game, запускающий основную бизнес-цепочку игры, создающий игровое поле с указанными пользователем размерами поля.
- 3 Реализован класс поля Field, состоящей из клеток Grid, который хранит набор клеток в виде двумерного массива, а также параметры, задающие размеры поля. Класс Grid хранит всю информацию о каждой клетке её координаты, проходима ли она, является ли входом или выходом.
- 4 Создан файл grid_type.h, хранящий возможные состояния клеток (является ли клетка стеной, входом, выходом, или на ней ничего нет.)
- 5 В классе Field созданы конструкторы копирования и перемещения, а также операторы для них. В перегрузке оператора перемещения мы очищаем память, выделенную под сетку, копируем из объекта параметры и координаты клеток. В перегрузке оператора копирования мы делаем то же самое, но без очищения памяти.
- 6 Создан абстрактный класс-интерфейс GridEntity, который в будущем будет хранить информацию о сущностях, расположенных на клетках.
- 7 В классе Field происходит отрисовка поля с помощью метода drawField.
- 8 Реализовано случайное появление клеток входа и выхода на поле. Данные клетки не появляются рядом.
- 9 Гарантированно отсутствие утечки памяти.

Разработанный программный код см. в приложении А Диаграмму с зависимостями классов см. в приложении Б

Тестирование.

Результаты тестирования представлены в табл. 1.

Таблица 1 – Результаты тестирования

No	ца 1 — Результаты тестир Тест	Результат	Комментарии
п/п			
1	10 10 Field field = new Field(10, 10);	######################################	Поле сгенерировано и отрисовано.
2	Field field2 (15, 15) field = field2; cout « field.getHeight()	15	Конструктор и оператор присваивания корректны.
3	field=std::move(field2) cout«field2.getHeight()	0	Конструктор перемещения и оператор присваивания корректны

1 Выводы.

В ходе работы было изучены способы создания собственных классов, их конструкторов и деструкторов, полей и методов.

Разработана программа по типу мини игры, генерирующая поле с непроходимыми клетками и стартом/финишем.

ПРИЛОЖЕНИЕ А ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: main.cpp

```
#include "../include/game.h"
int main(){
  int x, y;
  cin >> x;
  cin >> y;
  Game game (x, y);
  game.runGame(x, y);
  return 0;
Название файла: game.h
using namespace std;
#pragma once
#include "field.h"
class Game{
private:
  Field *field;
public:
  Game(int, int);
  ~Game();
  void runGame(int x, int y);
};
Название файла: game.cpp
#include "../include/game.h"
Game::Game(int x, int y){}
void Game::runGame(int x, int y){
  this->field = new Field(x, y);
Game::~Game(){
  delete field;
}
Название файла: GridEntity.h
#pragma once
class GridEntity{}; //абстрактный класс-интерфейс, отвечающий за будущее добавление сущностей в игру.
```

Название файла: GridEntity.cpp

```
#include "../include/GridEntity.h"
```

Название файла: grid.h

```
#include <iostream>
using namespace std;
#pragma once
#include "grid_type.h"
#include "../include/GridEntity.h"
class Grid{
private:
  GridEntity entity;
  int x = 0;
  int y = 0;
  int type;
  bool passable;
  char value = '_';
public:
  Grid();
  Grid(int x, int y, int type);
  int getX();
  int getY();
  int getItem();
  int getTypeOfGrid();
  void setValue(int type);
  bool isPassable();
  char getValue();
};
```

Название файла: grid.cpp

```
#include "../include/grid.h"

Grid::Grid(int x, int y, int type):
    x(x),
    y(y),
    type(type){
       passable = (type != BORDER);
       setValue(type);
    }

Grid::Grid(){
    };
```

```
int Grid::getX(){
  return x;
}
int Grid::getY(){
  return y;
}
char Grid::getValue(){
  return value;
}
int Grid::getItem(){
  //returns item
  return 0;
}
int Grid::getTypeOfGrid(){
  return type;
}
void Grid::setValue(int type){
  if (type == BORDER)
    value = '#';
  else if (type == START)
    value = 'S';
  else if (type == END)
    value = 'E';
  else
     value = '_';
}
bool Grid::isPassable(){
  return passable;
}
Название файла: grid_type.h
#pragma once
enum {BORDER, START, END, EMPTY};
Название файла: field.h
using namespace std;
```

```
#pragma once
#include "grid.h"
#include <string.h>
class Field{
private:
  int height = 0;
  int width = 0;
  void createField(int width, int height);
  void createField();
  void createField(int size);
  void drawField(int width, int height);
  string getRandom(int width, int height);
  Grid** grid;
public:
  Field(int width, int height);
  Field(const Field &); //копирование
  Field & operator=(const Field &);
  Field(Field &&);
  Field & operator=(Field & &);
  ~Field();
};
Название файла: field.cpp
#include "../include/field.h"
Field::Field(int w, int h)
  :width(w),
  height(h)
  createField(width, height);
}
Field::Field(const Field &other): //конструктор копирования
  width(other.width),
  height(other.height)
  createField(width, height);
}
Field &Field::operator=(const Field &other){ //оператор копирования
  for(int i = 0; i < width; i++)
     delete [] grid[i];
```

```
delete [] grid;
  width = other.width;
  height = other.height;
  createField(width, height);
  return *this;
}
Field::Field(Field &&other): //конструктор перемещения
  width(other.width),
  height(other.height),
  grid(other.grid)
  if(&other == this) return;
  other.grid = nullptr;
}
Field &Field::operator=(Field &&other){ //оператор перемещения
  if(this != &other) //попытка присвоить объект самому себе
     return *this;
  for(int i = 0; i < width; i++)
     delete [] grid[i];
  delete [] grid;
  width = other.width;
  height = other.height;
  grid = other.grid;
  return *this;
}
void Field::createField(int width, int height){
  srand(time(NULL));
  int x_start, y_start, x_end, y_end; //задание случайных координат на границах поля
  /*string cord = getRandom(width, height);
  char *chr_start = new char[cord.length() + 1]; //преобразованние полученной строки в char
  strcpy(chr_start, cord.c_str());
  sscanf(chr_start, "%d %d", &x_start, &y_start);
  cord = getRandom(width, height);
  char *chr_end = new char[cord.length() + 1];
  strcpy(chr_end, cord.c_str());
  sscanf(chr_end, "%d %d", &x_end, &y_end);
  printf("start x = %d, y = %d\n", x_start, y_start);
```

```
printf("end x = %d, y = %d\n", x_end, y_end);
  delete [] chr_start;
  delete [] chr_end;*/
  x_start = rand()%(width/4) + 1; //создание случайных координат входа и выхода внутри поля
  y_start = rand()\%(height/4) + 1;
  x_{end} = rand()\%(width/4) + width*3/4 - 1;
  y_{end} = rand()\%(height/4) + height*3/4 - 1;
  grid = new Grid*[width];
  for (int i = 0; i < width; i++) {
    grid[i] = new Grid[height];
    for (int j = 0; j < height; j++) {
       if (i * j == 0 || i == width - 1 || j == height - 1) { //назначение краев карты
          grid[i][j] = Grid(i, j, BORDER);
       }
       else
          grid[i][j] = Grid(i, j, EMPTY); //остальные пустые клетки
     }
  }
  grid[x_start][y_start] = Grid(x_start, y_start, START);
  grid[x_end][y_end] = Grid(x_end, y_end, END);
  drawField(width, height);
}
string Field::getRandom(int width, int height){ //получение случайных входа и выхода по краям поля
  int x = rand() \% width;
  int y = rand() % height;
  if(rand() \% 2 == 0){
    if((x == 0) || (x == width - 1)){} //крайний левый или правый столбец
       if((height - 1 - y) < (0 + y)) //остальные столбцы
         y = height - 1;
       else
         y = 0;
     }
  }
  else{
    if((y == 0) || (y == height - 1)){} //верхняя или нижняя строка
    else{
       if((width - 1 - x) < (0 + x)) //остальные строки
         x = width - 1;
       else
          x = 0;
     }
```

```
}
  string res = to_string(x) + " " + to_string(y);
  return res;
void Field::createField(){
  createField(10, 10);
}
void Field::createField(int size){
  createField(size, size);
}
void Field::drawField(int width, int height) {
  string char_to_str = ""; //преобразование char в string для отрисовки поля
  for (int i = 0; i < width; i++) {
    for (int j = 0; j < height; j++) {
       char_to_str += grid[i][j].getValue();
       cout << " " + char_to_str;
       char_to_str = "";
    }
    cout << "" << endl;
  }
}
Field::~Field() {
  for (int i = 0; i < width; i++) {
     delete [] grid[i];
  delete[] grid;
```

приложение Б

UML-ДИАГРАММА КЛАССОВ grid_type.h main.cpp + enum {BORDER, START, END, EMPTY}; + game: Game + x: int + y: int class Grid class Field + height: int + width: int + **grid: Grid + GridEntity: entity + x: int + y: int + type: int + passable: bool + value: char class Game + Field(int width, int height)
+ Field(const Field &)
+ Field &operator=(const Field &)
+ Field &field &&)
+ Field &operator=(Field &&);
+ -Field();
+ -Field();
+ createField(int width, int height); void
+ createField(int size); void
+ drawField(int width, int height); void
+ getRandom(int width, int height); string + *field: Field Class GridEntity + Game() + Grid()
+ Grid(int x, int y, int type)
+ getX(): int
+ getY(): int
+ getItem(): int
+ getTypeOfGrid(): int
+ setValue(int type): void
+ isPassable(): bool
+ getValue(): bool + Game() + runGame(int x, int y): void