**МИНОБРНАУКИ РОССИИ**

**Санкт-Петербургский государственный**

**электротехнический университет**

**«ЛЭТИ» им. В.И. Ульянова (Ленина)**

**Кафедра МО ЭВМ**

отчет

**по лабораторной работе №2**

**по дисциплине «ООП»**

Тема: Конструкторы и деструкторы

|  |  |  |
| --- | --- | --- |
| Студент гр. 2383 |  | Лустенкова Д.Д. |
| Преподаватель |  | Жангиров Т.Р. |

Санкт-Петербург

2024

## Цель работы

Создать классы клетки игрового поля и класс игрового поля со всеми необходимыми ему конструкторами, что является важным шагом в реализации проекта первой игры на языке программирования С++.

## Задание

а) Создать класс клетки игрового поля. Клетка игрового поля может быть проходимой или нет, тем самым определяя возможность игрока встать на эту клетку. Возможность задать проходимость клетки должна быть реализована через конструктор и через метод клетки. В будущем в клетке будет храниться указатель на интерфейс события.

б) Создать класс игрового поля. Игровое поле представляет собой прямоугольник из клеток (двумерный массив). В учебных целях, клетки хранятся как чистый массив на указателях (использовать контейнеры stl запрещено в этой лаб. работе). Размер поля передается в конструктор поля, в котором динамически выделяется память под массив клеток. Также должна быть возможность вызвать конструктор поля без аргументов. Так как происходит выделение память, то необходимо реализовать деструктор, в котором будет происходить очистка память. Также для класса поля необходимо реализовать конструкторы копирования и перемещения, а также соответствующие им операторы присваивания.

Также добавить в игровое поле добавьте информацию о входе (где в начале появляется игрок) и выходе (куда игрок должен дойти)

в) В класс управления игрока добавить взаимодействия с полем. При перемещении игрока должна быть проверка на проходимость клетки, если клетка непроходима, то перемещение не должно производиться.

Примечания:

* Так как в клетке будет храниться указатель, то при копировании и перемещении должен быть предусмотрен механизм копирования объекта хранящегося по указателю.
* Убедитесь, что присутствует проверка контроля размера поля, чтобы его нельзя было сделать слишком маленьким или с отрицательными размерами.
* В конструкторе перемещения и соответствующем ему операторе присваивания не должно происходить никакого копирования данных.
* Через класс поля должен быть доступ к каждой индивидуальной клетке. Создавать метод, который возвращает указатель на весь массив или указатель на каждую клетку, плохая практика, так как появляется возможность очистки памяти.

## Выполнение работы

Для удобства работы с клетками поля с помощью *enum* создан *cell\_type*, который может принимать одно из следующий значений: *Floor, Spikes, Barrier, Teleport, Exit\_door*: обычный пол, шипы, барьер, телепорт и выход соответственно.

Так как у игрока и его врагов есть много общих свойств, создан класс entity или сущность.

class cell

У класса всего 2 поля: *cell\_type type\_* для определения типа клетки и *cords tp\_cords\_* – поле, с которым будут происходить взаимодействия только для клетки типа *Teleport*.

Методы класса *cell*:

* *cell(cell\_type type, cords tp = {1, 1})* – конструктор класса по типу клетки*;*
* *cell()* – конструктор класса по умолчанию;
* *void set\_type\_(cell\_type type)* – метод присвоения полю *type\_* нового значения типа клетки;
* void set\_tp\_cords\_(cords tp\_cords) – метод присвоения полю tp\_cords\_ нового значения координат для телепортации;
* *cell\_type get\_type\_()* – метод, возвращающий тип клетки;
* *cords get\_tp\_cords\_()* – метод, возвращающий координаты для телепортации для клеток типа *Teleport*.

К методам клетки реализован доступ через соответствующие им методы поля.

class field

В классе созданы следующие поля:

* *uint32\_t height\_* – высота поля;
* *uint32\_t width\_* – ширина поля;
* *cords entrance\_* – координаты точки входа на поле;
* *cords exit\_* – координаты точки выхода с поля;
* *cell\*\* cells\_* – двумерный массив клеток (само поле).

Приватные методы (методы для работы с памятью):

* *cell\*\* allocate\_field()* – метод, выделяющий память под *height\_*\* *width\_* клеток;
* *void deallocate\_field()* – метод очистки памяти.

Публичные методы класса:

* *field(uint32\_t h, uint32\_t w, cords entrance = {1, 1})* – конструктор класса по высоте и ширине поля;
* *field()* – конструктор класса по умолчанию;

*// copying*

* *void copy(const field& other)* – метод копирования объекта класса;
* *field(const field& other)* – конструктор копирования;
* *field& operator = (const field& obj)* – переопределение оператора копирования;

*// move*

* *void move(field&& other)* – метод перемещения объекта класса;
* *field(field&& other)* – конструктор перемещения;
* *field& operator = (field&& obj)* – переопределение оператора перемещения;

*// getters and setters*

* *uint32\_t get\_height\_()* – метод, возвращающий высоту поля;
* *uint32\_t get\_width\_()* – метод, возвращающий ширину поля;
* *cords get\_entrance\_()* – метод, возвращающий координаты точки входа в поле;
* *cords get\_exit\_()* – метод, возвращающий координаты точки выхода из поля;
* *void set\_exit\_(cords exit\_cords)* – метод присвоения полю *exit\_* нового значения координат;
* void set\_type\_of\_cell(int x, int y, cell\_type type) – метод присвоения полю type\_ клетки по координатам x, y нового значения типа клетки;
* void set\_tp\_cords\_of\_cell(int x, int y, cords tp\_cords) – метод присвоения полю tp\_cords\_ клетки по координатам x, y нового значения координат для телепортации;
* *cell\_type touch\_cell(int x, int y)* – доступ к методу *get\_type\_()* клетки по координатам *x*, *y*;
* *cords get\_tp\_cords\_of\_cell(int x, int y)* – доступ к методу *get\_tp\_cords\_()* клетки по координатам *x*, *y*;

*// std::cout*

* *void str()* – метод вывода поля в консоль в виде цифр от 0 до 5;
* *~field()* – деструктор класса.

main()

В функции main() на данный момент реализации проекта находятся тестовые элементы создания, копирования и перемещения объектов класса.

Программный код приведен в приложении А.

## UML-диаграмма классов

entity

* # bool alive\_;
* # uint32\_t lives\_;
* # uint32\_t attack\_power\_;
* + entity();
* + void move(field& f, Direction move\_direction);
* + void kill();
* + void healthpointer(bool treat, unsigned int how\_many);
* + unsigned int attack\_power();

controller

* # entity& e;
* # field& f;
* # cords\_ xy;
* + controller(entity &somebody, field &somewhere, cords xy);
* + cords get\_cords\_();
* + void move(Direction move\_direction);

field

* - uint32\_t height\_;
* - uint32\_t width\_;
* - cell\*\* cells\_;

- void allocate\_field();

- void deallocate\_field();

+ field(uint32\_t h, uint32\_t w, cords entrance = {1, 1});

+ field();

+ void copy(const field& other);

+ field(const field& other);

+ field& operator = (const field& obj);

+ void move(field&& other);

+ field(field&& other);

+ field& operator = (field&& obj);

// getters and setters

uint32\_t get\_height\_();

uint32\_t get\_width\_();

cords get\_entrance\_();

cords get\_exit\_();

void set\_exit\_(cords exit\_cords);

cell\_type touch\_cell(int x, int y);

cords get\_tp\_cords\_of\_cell(int x, int y);

// std::cout

void str();

// Destructor

~field();

cell

* - cell\_type type\_;
* - cords tp\_cords\_;
* + cell(cell\_type type, cords tp = {1, 1});
* + cell();
* + void set\_type\_(cell\_type type);
* + void set\_tp\_cords\_(cords tp\_cords);
* + cell\_type get\_type\_();
* + cords get\_tp\_cords\_();

enemy

* + enemy();
* + cords step\_to\_move\_to\_player(int my\_x, int my\_y, int p\_x, int p\_y);

player

* # uint32\_t level\_;
* + player();
* + void kill();
* + void raise\_power(int raise);
* + void levelup();

## Тестирование

Результаты тестирования представлены в табл. 1.

Таблица 1 – Результаты тестирования.

|  |  |  |  |
| --- | --- | --- | --- |
| № п/п | Входные данные | Выходные данные | Комментарии |
|  | field f1 = field(20, 20, {5, 1});  std::cout << '\n';  field f2(f1);  f1.str();  std::cout << '\n';  f2.str();  std::cout << '\n';  field f3(field(10, 10));  //field f3 = std::move(field(10, 10));  f3.str();  std::cout << '\n'; | 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2  2 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 2  2 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 2  2 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 2  2 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 2  2 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 2  2 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 2  2 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 2  2 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 2  2 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 2  2 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 2  2 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 2  2 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 2  2 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 2  2 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 2  2 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 2  2 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 2  2 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 2  2 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 2  2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2  2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2  2 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 2  2 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 2  2 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 2  2 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 2  2 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 2  2 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 2  2 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 2  2 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 2  2 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 2  2 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 2  2 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 2  2 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 2  2 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 2  2 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 2  2 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 2  2 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 2  2 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 2  2 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 2  2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2  2 2 2 2 2 2 2 2 2 2  2 0 0 0 0 0 0 0 0 2  2 0 0 0 0 0 0 0 0 2  2 0 0 0 0 0 0 0 0 2  2 0 0 0 0 0 0 0 0 2  2 0 0 0 0 0 0 0 0 2  2 0 0 0 0 0 0 0 0 2  2 0 0 0 0 0 0 0 0 2  2 0 0 0 0 0 0 0 0 2  2 2 2 2 2 2 2 2 2 2  p1: 6 1  p1: 18 18 | Задача выполнена корректно. |

## Выводы

Написаны класс клетки игрового поля и класс игрового поля.

Сделан второй шаг в реализации проекта игры на языке программирования С++.

# **Приложение А Исходный код программы**

Название файла: cords.h

#include <iostream>

enum cell\_type {Floor, Spikes, Barrier, Teleport, Exit\_door};

struct cords

{

int x\_;

int y\_;

};

Название файла: cell.h

#include "cords.h"

// a cell is a unit of a field

class cell

{

private:

cell\_type type\_;

cords tp\_cords\_;

protected:

public:

cell(cell\_type type, cords tp = {1, 1});

cell();

void set\_type\_(cell\_type type);

void set\_tp\_cords\_(cords tp\_cords);

cell\_type get\_type\_();

cords get\_tp\_cords\_();

};

Название файла: cell.cpp

#include "cell.h"

cell::cell(cell\_type type, cords tp)

: type\_(type)

{

if (type\_ == Teleport)

tp\_cords\_ = tp;

}

cell::cell() : type\_(Floor) {}

void cell::set\_type\_(cell\_type type)

{

type\_ = type;

}

void cell::set\_tp\_cords\_(cords tp\_cords)

{

tp\_cords\_ = tp\_cords;

}

cell\_type cell::get\_type\_()

{

return type\_;

}

cords cell::get\_tp\_cords\_()

{

if (type\_ == Teleport)

return tp\_cords\_;

}

Название файла: field.h

#include "cell.h"

#include <cstdint>

class field

{

private:

// memory operations

void allocate\_field();

void deallocate\_field();

//

uint32\_t height\_;

uint32\_t width\_;

cords entrance\_;

cords exit\_;

cell\*\* cells\_;

protected:

public:

field(uint32\_t h, uint32\_t w, cords entrance = {1, 1});

field();

// copying

void copy(const field& other);

field(const field& other);

field& operator = (const field& obj);

// move

void move(field&& other);

field(field&& other);

field& operator = (field&& obj);

// getters and setters

uint32\_t get\_height\_();

uint32\_t get\_width\_();

cords get\_entrance\_();

cords get\_exit\_();

void set\_exit\_(cords exit\_cords);

void set\_type\_of\_cell(int x, int y, cell\_type type);

void set\_tp\_cords\_of\_cell(int x, int y, cords tp\_cords);

cell\_type touch\_cell(int x, int y);

cords get\_tp\_cords\_of\_cell(int x, int y);

// std::cout

void str();

~field();

};

Название файла: field.cpp

#include "field.h"  
  
// memory operations  
  
void field::allocate\_field()  
{  
 cells\_ = new cell \* [height\_];  
 for (int y = 0; y < height\_; y++)  
 {  
 cells\_[y] = new cell[width\_];  
 }  
}  
  
void field::deallocate\_field()  
{  
 if (cells\_ != nullptr)  
 {  
 for (int y = 0; y < height\_; y++)  
 {  
 delete[] cells\_[y];  
 }  
 delete[] cells\_;  
 }  
 cells\_ = nullptr;  
}  
  
//  
  
field::field(uint32\_t h, uint32\_t w, cords entrance)  
 : height\_(h), width\_(w), entrance\_(entrance), exit\_({ -1, -1 })  
{  
 // Exception  
 if (h < 10 || w < 10)  
 {  
 std::cerr << "The sizes of the field are not correct" << '\n';  
 }  
 // Normal work  
 allocate\_field();  
 for (int y = 0; y < height\_; y++)  
 {  
 for (int x = 0; x < width\_; x++)  
 {  
 if (x == 0 || y == 0 || x == width\_ - 1 || y == height\_ - 1)  
 cells\_[y][x].set\_type\_(Barrier);  
 }  
 }  
}  
  
field::field()  
{  
 field(50, 50, { 1, 1 });  
}  
  
//copying  
  
void field::copy(const field& other)  
{  
 if (this != &other)  
 {  
 height\_ = other.height\_;  
 width\_ = other.width\_;  
 entrance\_ = other.entrance\_;  
 exit\_ = other.exit\_;  
  
 allocate\_field();  
  
 for (int y = 0; y < height\_; y++)  
 {  
 for (int x = 0; x < width\_; x++)  
 cells\_[y][x] = other.cells\_[y][x];  
 }  
 }  
}  
  
field::field(const field& other)  
{  
 copy(other);  
}  
  
field& field::operator = (const field& obj)  
{  
 copy(obj);  
 return \*this;  
}  
  
//  
  
// move  
  
void field::move(field&& other)  
{  
 if (this != &other)  
 {  
 height\_ = other.height\_;  
 width\_ = other.width\_;  
 entrance\_ = other.entrance\_;  
 exit\_ = other.exit\_;  
 std::swap(cells\_, other.cells\_);  
 other.cells\_ = nullptr;  
 }  
}  
  
field::field(field&& other)  
{  
 move(std::move(other));  
}  
  
field& field::operator = (field&& obj)  
{  
 move(std::move(obj));  
 return \*this;  
}  
  
//  
  
// getters and setters  
  
uint32\_t field::get\_height\_()  
{  
 return height\_;  
}  
  
uint32\_t field::get\_width\_()  
{  
 return width\_;  
}  
  
cords field::get\_entrance\_()  
{  
 return entrance\_;  
}  
  
cords field::get\_exit\_()  
{  
 return exit\_;  
}  
  
void field::set\_exit\_(cords exit\_cords)  
{  
 exit\_ = exit\_cords;  
 cells\_[exit\_cords.y\_][exit\_cords.x\_].set\_type\_(Exit\_door);  
}  
  
void field::set\_type\_of\_cell(int x, int y, cell\_type type)  
{  
 cells\_[y][x].set\_type\_(type);  
}  
  
cell\_type field::touch\_cell(int x, int y)  
{  
 return cells\_[y][x].get\_type\_();  
}  
  
cords field::get\_tp\_cords\_of\_cell(int x, int y)  
{  
 return cells\_[y][x].get\_tp\_cords\_();  
}  
  
//  
  
// std::cout  
  
void field::str()  
{  
 for (int y = 0; y < height\_; y++)  
 {  
 for (int x = 0; x < width\_; x++)  
 {  
 std::cout << cells\_[y][x].get\_type\_() << ' ';  
 }  
 std::cout << '\n';  
 }  
}  
//  
  
// Destructor  
  
field::~field()  
{  
 for (int y = 0; y < height\_; y++)  
 {  
 delete[] cells\_[y];  
 }  
 delete[] cells\_;  
}

Название файла: main.cpp

#include "controller.h"

int main()

{

player p1;

p1.kill();

field f1 = field(20, 20, {5, 1});

std::cout << '\n';

field f2(f1);

f1.str();

std::cout << '\n';

f2.str();

std::cout << '\n';

field f3(field(10, 10));

//field f3 = std::move(field(10, 10));

f3.str();

std::cout << '\n';

controller c\_p1(p1, f1, f1.get\_entrance\_());

c\_p1.move(Right);

std::cout << "p1: " << c\_p1.get\_cords\_().x\_ << ' ' << c\_p1.get\_cords\_().y\_ << '\n';

for (int i = 0; i < 20; i++)

{

c\_p1.move(Right);

c\_p1.move(Top);

}

std::cout << "p1: " << c\_p1.get\_cords\_().x\_ << ' ' << c\_p1.get\_cords\_().y\_ << '\n';

enemy e1;

controller c\_e1(e1, f1, {3, 4});

e1.step\_to\_move\_to\_player(c\_e1.get\_cords\_().x\_, c\_e1.get\_cords\_().y\_, c\_p1.get\_cords\_().x\_, c\_p1.get\_cords\_().y\_);

return 0;

}