**МИНОБРНАУКИ РОССИИ**

**Санкт-Петербургский государственный**

**электротехнический университет**

**«ЛЭТИ» им. В.И. Ульянова (Ленина)**

**Кафедра МО ЭВМ**

отчет

**по лабораторной работе №2**

**по дисциплине «Объектно-ориентированное программирование»**

Тема: Конструкторы и деструкторы

|  |  |  |
| --- | --- | --- |
| Студент гр. 2300 |  | Войнов А.Н. |
| Преподаватель |  | Жангиров Т.Р. |

Санкт-Петербург

2023

**Цель работы.**

Познакомиться с конструкторами и деструкторами в языке с++. Изучить различные виды конструкторов (без аргументов, с аргументами, копирования, перемещения) и реализовать их. На основе полученных знаний создать класс клетки и поля.

**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 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**

Были созданы 2 класса и 1 структура и 1 перечисление:

1. Point – структура, состоит из 2 полей: x и y. Используется для хранения координат, а также размера поля. В структуре перегружены операторы математических операций и метод для вывода данных в консоль для упрощения работы с ней.
2. CellType – перечисление, используемое для хранения состояния клетки.
3. Cell – класс клетки, хранит данные о конкретной клетки, её проходимости/непроходимости, является ли она финишем или стартом
4. Field – класс поля, хранит данные о всём игровом поле: размер, массив клеток, координаты старта и финиша, а также осуществляет обработку клеток и доступ к ним.

Также был доработан класс PlayerController и Player.

Все классы объявлены в соответствующих заголовочных файлах, а их реализация представлена в .cpp файлах. Исходный код см. в приложении А.

В структуре Point перегружены следующие операторы:

1. Point operator+(const Point& other) const – складывает x и y двух точек
2. Point operator-(const Point& other) const – вычитает из координат текущей точки координаты переданной
3. Point& operator+=(const Point& other) – аналогичен первому, но с присваиванием
4. Point& operator-=(const Point& other) – аналогичен второму, но с присваиванием
5. bool operator==(const Point& other) const – точки считаются равными, если равны их соответствующие координаты
6. bool operator!=(const Point& other) const – точки не равны, если не выполняется условие из пункта 5

Состояния, которые хранит CellType:

1. UNPASSABLE – клетка непроходима, игрок не может попасть на неё и пройти сквозь неё. Все остальные состояния означают, что клетка проходима.
2. START – клетка является стартом.
3. FINISH – клетка является финишем.
4. PASSABLE – клетка проходима, но не является ни стартом ни финишем.

В классе Cell создано одно поле private type\_, хранящее переменную типа CellType, определяющую состояние клетки.

В Cell реализованы следующие методы:

1. Cell() – конструктор по умолчанию, инициализирует поле type\_ значением CellType::PASSABLE, т.е. по умолчанию клетка проходима.
2. Cell(CellType type) – конструктор с переданным значением типа клетки.
3. Cell(const Cell& other) – конструктор копирования клетки
4. Cell& operator=(const Cell& other) – оператор присваивания с копированием.
5. Cell(Cell&& other) – конструктор перемещения клетки
6. Cell& operator=(Cell&& other) – оператор присваивания с перемещением.
7. void set\_type(CellType type) – устанавливает type\_ равным type.
8. bool isStart() const – возвращает true, если клетка старт, иначе false.
9. bool isFinish() const - возвращает true, если клетка финиш, иначе false.
10. bool isPassable() const - возвращает true, если клетка проходима, иначе false.

В классе Field созданы следующие поля:

1. private Point size\_ – размер поля.
2. private Cell\*\* field\_ - двумерный массив размером size\_.x на size\_.y , хранящий все клетки поля
3. private Point start\_ - координаты старта.
4. private Point finish\_ - координаты финиша.

В классе Field реализованы следующие методы:

1. Field() – конструктор по умолчанию. Вызывает конструктор с двумя аргументами типа int.
2. Field(int x, int y) – конструктор, который принимает размеры поля по x и y. Инициализирует size\_ значениями x и y, а start\_ и finish\_ точками с координатами (-1,-1). Это даёт возможность понять были ли заданы точки старта и финиша или нет. Затем вызывает createField() для заполнения поля клетками.
3. Field::Field(const Point& size) – конструктор, который принимает размер поля в обертке Point, вызывает конструктор с двумя целочисленными аргументами.
4. Field::Field(const Field& other) – конструктор копирования. Инициализирует size\_, finish\_ и start\_ соответствующими значениями из other. Вызывает createField, а затем копирует клети из other.field\_
5. Field &Field::operator= (const Field &other) – оператор присваивания с копированием. Создает временный объект Field и меняется данными с ним с помощью std::swap().
6. Field(Field&& other) – конструктор перемещения. Вызывает оператор присваивания с перемещением.
7. Field& operator= (Field&& other) – оператор присваивания с перемещением. Если присваивается не тот же самый объект, то обменивается с ним значениями полей с помощью std::swap().
8. void createField(const Point& size) – если переданные размеры больше максимального или меньше минимального, вызывает исключение. Иначе выделяет память для массива клеток нужного размера
9. void clearField() – если поле field\_ не является nullptr, очищает выделенную память и меняет указатель на nullptr.
10. bool doesContainsPoint(const Point& coordinates) const – проверяет находятся ли координаты внутри поля (между 0 и размером поля). Если да возвращае тtrue, иначе false. Если да возвращае тtrue, иначе false.
11. bool isPassable(const Point& coordinates) – проверяет находится ли клетка внутри поля и проходима ли она. Если да возвращае тtrue, иначе false.
12. Cell& getCell(Point&& coordinates) и Cell& getCell(const Point& coordinates) – возвращают ссылку на клетку по указанным координатам, если она находится в поле. Иначе вызывает исключение.
13. void setCell(const Point& coordinates, const Cell& cell) и void setCell(const Point& coordinates, Cell&& cell) – меняют клетку на новую по заданному адресу, если она существует. Иначе ничего не происходит.
14. Point& size(), Point& finish(), Point& start() – гетеры для соответствующих полей
15. void set\_start(const Point& coordinates) – если точка находится внутри поля и является проходимой, делает её стартом и возвращает 0, иначе возвращает 1.
16. void set\_finish(const Point& coordinates) - если точка находится внутри поля и является проходимой, делает её финишем и возвращает 0, иначе возвращает 1.
17. void createWall(const Point& coordinates) - если точка находится внутри поля и, делает её непроходимой. В отличии от set\_start() и set\_finish() не возвращает ничего, потому что отсутствие старта и финиша губительно для игры, в отличие от стены.
18. void makePassable(const Point& coordinates) - если точка находится внутри поля и, делает её проходимой. В отличии от set\_start() и set\_finish() не возвращает ничего, потому что отсутствие старта и финиша губительно для игры, в отличие от стены.

Изменения в Player

1. созданы конструкторы копирования и перемещения, а также соответствующие им операторы присваивания
2. метод coordinates() теперь возвращает Point вместо std::pair<int,int>
3. исправлены синтаксические ошибки в названии методов.

Изменения в PlayerController:

1. В методе move реализовано перемещение с учётом скорости, наличия стен и краев карты. Для этого созданы переменные direction\_sign (1 – если координата увеличится, иначе -1) и doesChangeX (true – если перемещение происходит по координате х, иначе false). Затем в цикле проверяется можно ли встать на очередную клетку (проверка происходит для расстояния от 1 до текущего значения скорости). Если на очередную клетку встать нельзя, игрок остановится на предыдущей и выполнение метода завершиться. Если можно переместиться на полное значение скорости, игрок будет перемещён на него.
2. get\_coordinates() теперь возвращает Point.

**Тестирование**

В функции main смоделирована простейшая работа игры. Для этого:

1. создана карта, состоящая из символов ASCII, где ‘-‘ значит свободная клетка, ‘=’ клетка со стеной, ‘!’ старт, ‘?’ финиш. Игрок изображается буквой P.
2. Данные о карте переписаны из массива строк в массив клеток
3. Создан PlayerController и ссылками на созданное поле и объект игрока
4. Установлена точка финиша
5. В цикле while ожидается ввод направления перемещения с помощью строчных букв ‘wsad’, после чего происходит перемещение и поле выводится заново.
6. Процесс повторяется до

Исходный код см. в приложении Б.

Результаты тестирования представлены в табл. 1.

Стартовое состояние поля:



Таблица 1 – Результаты тестирования

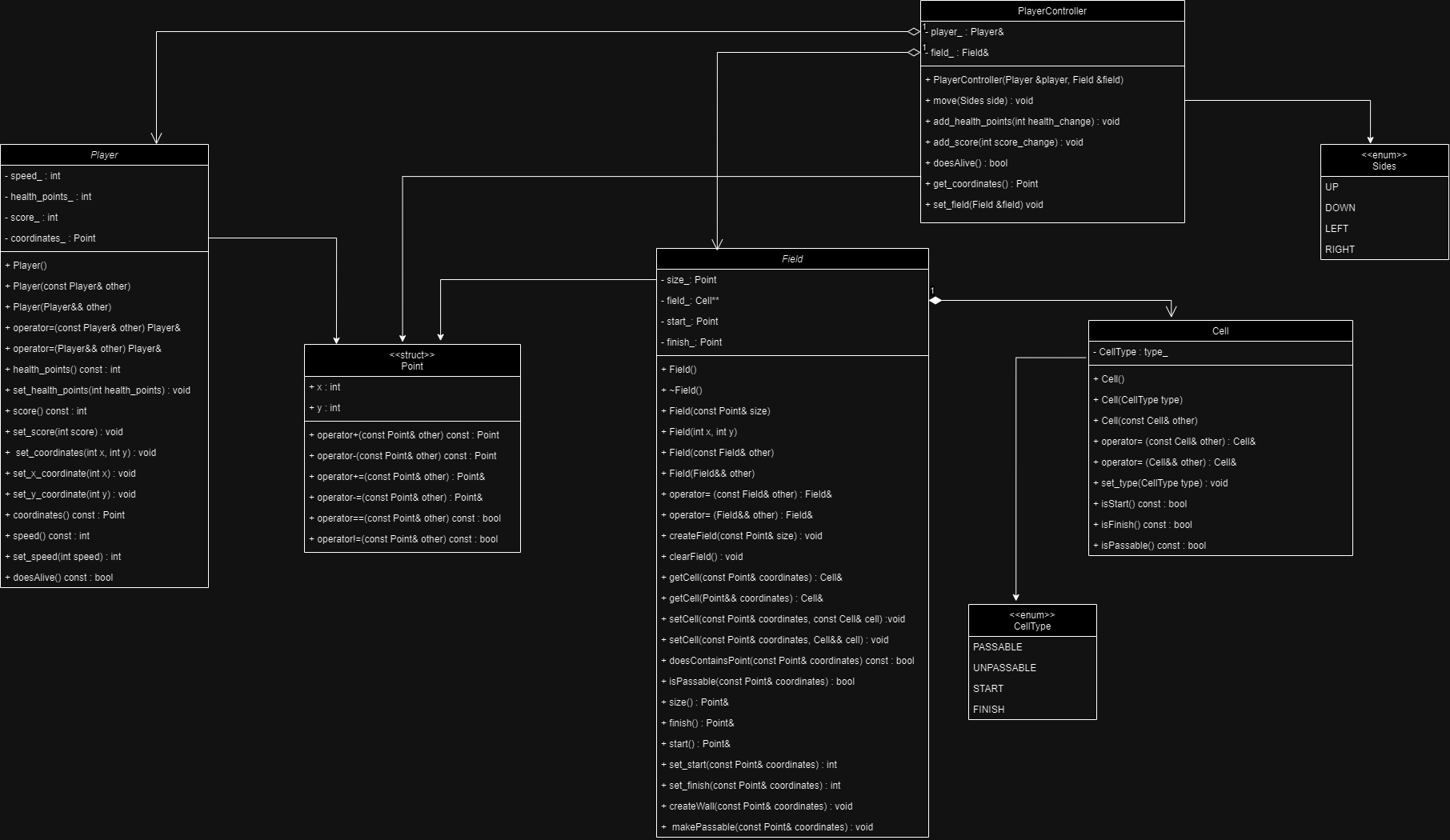
|  |  |
| --- | --- |
| Введенные данные | Состояние игрового поля |
| s |  |
| d |  |
| d |  |
| s |  |

Вывод:

Были изучены различные конструкторы (без аргументов, с аргументами, копирования, перемещения) и деструкторы в языке с++. Полученные знания были применены для работы над игрой, в частности для создания класса клетки и поля, а также доработки уже имеющихся классов

**Приложение А**

UML диаграмма классов



**Приложение Б**

Исходный код программы

Название файла: Point.h

#ifndef CODE\_POINT\_H

#define CODE\_POINT\_H

#include <iostream>

struct Point {

int x;

int y;

Point operator+(const Point& other) const;

Point operator-(const Point& other) const;

Point& operator+=(const Point& other);

Point& operator-=(const Point& other);

bool operator==(const Point& other) const;

bool operator!=(const Point& other) const;

};

std::ostream& operator<<(std::ostream &os, const Point& point);

#endif //CODE\_POINT\_H

Название файла: Point.cpp

#ifndef CODE\_POINT\_CPP

#define CODE\_POINT\_CPP

#include "../headers/Point.h"

Point Point::operator+(const Point& other) const {

return Point{x + other.x, y + other.y};

}

Point Point::operator-(const Point& other) const {

return Point{x - other.x, y - other.y};

}

Point& Point::operator+=(const Point& other) {

x += other.x;

y += other.y;

return \*this;

}

Point& Point::operator-=(const Point& other) {

x -= other.x;

y -= other.y;

return \*this;

}

bool Point::operator==(const Point& other) const {

return (x == other.x) && (y == other.y);

}

bool Point::operator!=(const Point& other) const {

return !(operator==(other));

}

std::ostream& operator<<(std::ostream &os, const Point& point) {

return os << '(' << point.x << ", " << point.y << ')';

}

#endif //CODE\_POINT\_CPP

Название файла: Cell.h

#ifndef CODE\_CELL\_H

#define CODE\_CELL\_H

#include "CellType.h"

#include <iterator>

#include <utility>

class Cell {

public:

Cell();

Cell(CellType type);

Cell(const Cell& other);

Cell& operator= (const Cell& other);

Cell(Cell&& other);

Cell& operator= (Cell&& other);

void set\_type(CellType type);

bool isStart() const;

bool isFinish() const;

bool isPassable() const;

private:

CellType type\_;

};

#endif //CODE\_CELL\_H

Название файла: Cell.cpp

#include "../headers/Cell.h"

Cell::Cell() : type\_(CellType::PASSABLE){}

Cell::Cell(CellType type) : type\_(type){}

Cell::Cell(const Cell& other) : type\_(other.type\_) {}

Cell& Cell::operator= (const Cell& other) {

Cell temp(other);

std::swap(type\_, temp.type\_);

return \*this;

}

Cell::Cell(Cell&& other) : type\_(other.type\_) {}

Cell& Cell::operator= (Cell&& other) {

if(this != &other) {

std::swap(type\_, other.type\_);

}

return \*this;

}

void Cell::set\_type(CellType type) {

type\_ = type;

}

bool Cell::isStart() const {

return type\_ == CellType::START;

}

bool Cell::isFinish() const {

return type\_ == CellType::FINISH;

}

bool Cell::isPassable() const {

return type\_ != CellType::UNPASSABLE;

}

Название файла: CellType.h

#ifndef CODE\_CELLTYPE\_H

#define CODE\_CELLTYPE\_H

enum CellType{

PASSABLE, UNPASSABLE, START, FINISH

};

#endif //CODE\_CELLTYPE\_H

Название файла: Field.h

#ifndef CODE\_FIELD\_H

#define CODE\_FIELD\_H

#define MAX\_SIZE 100

#define MIN\_SIZE 5

#include "Cell.h"

#include "Point.h"

class Field {

public:

Field();

~Field();

Field(const Point& size);

Field(int x, int y);

Field(const Field& other);

Field(Field&& other);

Field& operator= (const Field& other);

Field& operator= (Field&& other);

void createField(const Point& size);

void clearField();

Cell& getCell(const Point& coordinates);

Cell& getCell(Point&& coordinates);

void setCell(const Point& coordinates, const Cell& cell);

void setCell(const Point& coordinates, Cell&& cell);

bool doesContainsPoint(const Point& coordinates) const;

bool isPassable(const Point& coordinates);

Point& size();

Point& finish();

Point& start();

int set\_start(const Point& coordinates);

int set\_finish(const Point& coordinates);

void createWall(const Point& coordinates);

void makePassable(const Point& coordinates);

private:

Point size\_;

Cell\*\* field\_;

Point start\_;

Point finish\_;

};

#endif //CODE\_FIELD\_H

Название файла: Field.cpp

#include "../headers/Field.h"

Field::Field() : Field(MIN\_SIZE, MIN\_SIZE) {}

Field::~Field() {

clearField();

}

Field::Field(int x\_size, int y\_size) : Field(Point{x\_size, y\_size}){}

Field::Field(const Point& size) : start\_({-1, -1}), finish\_({-1, -1}), field\_(nullptr){

createField(size);

size\_ = size;

}

Field::Field(const Field& other) : size\_(other.size\_), start\_(other.start\_), finish\_(other.finish\_), field\_(nullptr) {

createField(size\_);

for(int i = 0; i < size\_.y; i++) {

for(int j = 0; j < size\_.x; j++) {

field\_[i][j] = other.field\_[i][j];

}

}

}

Field &Field::operator= (const Field &other) {

Field temp(other);

clearField();

std::swap(field\_, temp.field\_);

std::swap(start\_, temp.start\_);

std::swap(finish\_, temp.finish\_);

std::swap(size\_, temp.size\_);

return \*this;

}

Field::Field(Field &&other) : size\_({0,0}), start\_({0,0}), finish\_({0,0}), field\_(nullptr) {

\*this = std::move(other);

}

Field &Field::operator= (Field &&other) {

if(&other != this) {

clearField();

std::swap(size\_, other.size\_);

std::swap(start\_, other.start\_);

std::swap(finish\_, other.finish\_);

std::swap(field\_, other.field\_);

other.field\_ = nullptr;

}

return \*this;

}

void Field::createField(const Point& size) {

if(size.x > MAX\_SIZE || size.x < MIN\_SIZE || size.y > MAX\_SIZE || size.y < MIN\_SIZE) {

throw std::invalid\_argument("неверно заданы размеры поля");

}

field\_ = new Cell\* [size.y];

for(int i = 0; i < size.y; i++) {

field\_[i] = new Cell[size.x];

}

}

void Field::clearField() {

if(field\_ == nullptr) {

return;

}

for(int i = 0; i < size\_.y; i++) {

delete[] field\_[i];

}

delete[] field\_;

field\_ = nullptr;

}

bool Field::doesContainsPoint(const Point& coordinates) const {

return (coordinates.x >= 0 && coordinates.x < size\_.x && coordinates.y >= 0 && coordinates.y < size\_.y);

}

bool Field::isPassable(const Point& coordinates) {

return doesContainsPoint(coordinates) && getCell(coordinates).isPassable();

}

Cell& Field::getCell(const Point& coordinates) {

if(!doesContainsPoint(coordinates)) {

throw std::out\_of\_range("there is no cell with such coordinates");

}

return field\_[coordinates.y][coordinates.x];

}

Cell& Field::getCell(Point&& coordinates) {

if(!doesContainsPoint(coordinates)) {

throw std::out\_of\_range("there is no cell with such coordinates");

}

return field\_[coordinates.y][coordinates.x];

}

void Field::setCell(const Point &coordinates, Cell &&cell) {

if(!doesContainsPoint(coordinates)) {

field\_[coordinates.y][coordinates.x] = cell;

}

}

void Field::setCell(const Point& coordinates, const Cell& cell) {

if(!doesContainsPoint(coordinates)) {

field\_[coordinates.y][coordinates.x] = cell;

}

}

void Field::createWall(const Point &coordinates) {

if(doesContainsPoint(coordinates)) {

field\_[coordinates.y][coordinates.x].set\_type(CellType::UNPASSABLE);

}

}

void Field::makePassable(const Point &coordinates) {

if(!doesContainsPoint(coordinates)) {

field\_[coordinates.y][coordinates.x].set\_type(CellType::PASSABLE);

}

}

Point& Field::size() {

return size\_;

}

Point& Field::finish() {

return finish\_;

}

Point& Field:: start() {

return start\_;

}

int Field::set\_start(const Point &coordinates) {

if(doesContainsPoint(coordinates) && field\_[coordinates.y][coordinates.x].isPassable()) {

start\_ = coordinates;

field\_[coordinates.y][coordinates.x].set\_type(CellType::START);

return 0;

}

return 1;

}

int Field::set\_finish(const Point &coordinates) {

if(doesContainsPoint(coordinates) && field\_[coordinates.y][coordinates.x].isPassable()) {

finish\_ = coordinates;

field\_[coordinates.y][coordinates.x].set\_type(CellType::FINISH);

return 0;

}

return 1;

}