**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. 0304: |  | Шквиря Е.В. |
| 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, используя стандартную библиотеку языка C++.

## Задание.

* 1. 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. Для удобства все файлы программы были разбиты на несколько директорий: Object — будущие объекты на поле, Tools — инструменты для конструирования поля, UI — элементы демонстрации интерфейса пользователю, Models — модели элементов поля для вывода на экран.
2. Чтобы обеспечить логическое структурирование системы классов, все методы были добавлены в классы в соответствии с их задачами: CellPoint — объект, хранящий координаты клетки; CellObject — объект, который будет располагаться в клетке; Cell — элемент сетки поля; Grid — сетка, которая отвечает за структуру поля; Field — обёртка Grid, которая работает непосредственно с объектами на поле и занимается их генерацией и взаимодействием друг с другом; FieldIterator — итератор для обхода поля; FieldScreen — главный экран игры, который получает параметры поля от пользователя, регистрирует нажатия и выводит обновлённую информацию о поле после этого.
3. Класс Cell хранит в себе информацию об объекте CellObject (герое, враге, вещи, стене и т.п) и его координатах CellPoint на сетке поля. Класс обладает геттером/сеттером элемента поля, чтобы класс Field мог генерировать нужные объекты.
4. Класс Grid содержит в себе двумерный массив объектов типа Cell, а также высоту и ширину сетки. Также в классе присутствуют методы для проверки корректности высоты/ширины и индексов относительно размеров поля, геттеры размеров сетки, геттер/сеттер элемента сетки, методы очистки и изменения размеров поля. Для удобства передачи данных о сетке был реализован конструктор копирования и перемещения и соответствующие им операторы присваивания. Отсутствие утечки памяти гарантирует наличие вручную написанного деструктора, вызываемого при удалении класса. Класс Grid имеет дружественные классы FieldScreen для доступа к функциям проверки вводимых пользователем данных о размерах поля. Также дружественными классами являются Field и FieldIterator, так как они взаимодействуют с элементами сетки.
5. Класс Field содержит в себе сетку поля, стартовую и финишную точку, позицию игрока, статусы генерации поля, количество стен и нужное расстояние между стартом и финишем. Также в классе присутствуют методы: проверка корректности генерируемых данных, генерация граничных точек для старта и финиша, генерацию пути между стартом и финишем для гарантирования возможности пройти игру, генерация непроходимых точек. Для удобства использования класса все генерации можно вызывать с помощью метода *generateFullField*, также есть возможность получать данные о поле и его статусе через геттеры, очищать сгенерированный путь и двигать героя. Для данного класса FieldScreen является дружеским, чтобы получить доступ к проверкам индексов при запросе перемещения героя.
6. Класс FieldScreen содержит в себе указатель на поле (чтобы любые изменения поля не нужно было загружать заново) для дальнейшей работы с ним. Также в классе присутствуют методы для ввода начальных параметров поля, вывода обновлённого поля, регистрации нажатия пользователем кнопки управления героем и создания запроса на передвижение объекта поля. Для начала функционирования игры в main вызываются методы *showStartFieldScreen* для ввода стартовых параметров и *gameStatusObserver* для наблюдения за состоянием игры.

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

|  |  |  |  |
| --- | --- | --- | --- |
| № п/п | Тест | Результат | Комментарии |
|  | Field field1 = Field(30, 30);  field1.generateFullField(500); | OK | Поле сгенерировано |
|  | auto iter = field1. GetFieldIterator();  for (int i = 0; i < field1.getHeight(); ++i) { for (int j = 0; j < field1.getWidth(); ++j) { iter++;}} | OK | Итератор верно обошёл всё поле |
|  | Field copyField = field1;  copyField = field2; | OK | Конструктор копирования и оператор присваивания корректны |
|  | Field moveField = std::move(field1);  moveField = std::move(field2); | OK | Конструктор перемещения и оператор присваивания корректны |
|  | mainScreen.showStartFieldScreen();  mainScreen.gameStatusObserver(); | OK | Данные пользователя получены и поле по ним построено |

## Выводы.

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

Разработана программа по типу миниигры, получающая от пользователя параметры поля и генерирующая его с заданным количеством непроходимых клеток и стартом/финишем.

# Приложение А Исходный код программы

Название файла: CellObject.cpp

#include "CellObject.h"  
  
char CellObject::getCellAsChar() const {  
 char typeCell = '?';  
 switch (this->getTypeCell()) {  
 case TypeCell::*WAY*:  
 typeCell = '#';  
 break;  
 case TypeCell::*WALL*:  
 typeCell = '+';  
 break;  
 case TypeCell::*START*:  
 typeCell = 'S';  
 break;  
 case TypeCell::*FINISH*:  
 typeCell = 'F';  
 break;  
 case TypeCell::*EMPTY*:  
 typeCell = ' ';  
 break;  
 }  
 char typeObject = '?';  
 switch (this->getTypeObject()) {  
 case TypeObject::*HERO*:  
 typeObject = 'H';  
 break;  
 case TypeObject::*ENEMY*:  
 typeObject = 'E';  
 break;  
 case TypeObject::*THING* :  
 typeObject = 'T';  
 break;  
 }  
 if (typeObject != '?')  
 return typeObject;  
 else  
 return typeCell;  
}  
  
int CellObject::getTypeCell() const {  
 return this->typeCell;  
}  
  
void CellObject::setTypeCell(TypeCell typeCell) {  
 this->typeCell = typeCell;  
}  
  
int CellObject::getTypeObject() const {  
 return this->typeObject;  
}  
  
void CellObject::setTypeObject(TypeObject typeObject) {  
 this->typeObject = typeObject;  
}

Название файла: CellPoint.cpp

#include "CellPoint.h"  
  
bool CellPoint::isIndexedCurr() const {  
 return 0 <= this->x && 0 <= this->y;  
}  
  
int CellPoint::getX() const {  
 return this->x;  
}  
  
int CellPoint::getY() const {  
 return this->y;  
}  
  
bool CellPoint::operator==(CellPoint point) const {  
 return this->x == point.x && this->y == point.y;  
}

Название файла: FieldIterator.cpp  
  
#include "FieldIterator.h"  
  
Cell FieldIterator::getElem() const {  
 if (rootGrid->isValidIndexes(posX, posY))  
 return rootGrid->getElem(CellPoint(posX, posY));  
 else  
 throw -1;  
}  
  
void FieldIterator::setElem(Cell cell) {  
 this->rootGrid->setElem(CellPoint(posX,posY), cell);  
}  
  
CellPoint FieldIterator::getCurrentPosition() const {  
 return {posX, posY};  
}  
  
void FieldIterator::moveDelta(int deltaX, int deltaY) {  
 if (rootGrid->isValidIndexes(posX + deltaX, posY + deltaY)) {  
 posX += deltaX;  
 posY += deltaY;  
 }  
}  
  
void FieldIterator::moveTo(int posX, int posY) {  
 if (rootGrid->isValidIndexes(posX, posY)) {  
 this->posX = posX;  
 this->posY = posY;  
 }  
}  
  
FieldIterator& FieldIterator::operator++() {  
 if (rootGrid->isValidIndexes(posX + 1, posY)) {  
 posX++;  
 return \*this;  
 }  
 if (rootGrid->isValidIndexes(0, posY + 1)) {  
 posX = 0;  
 posY++;  
 return \*this;  
 }  
 posX = 0; // end-значение  
 posY = rootGrid->getHeight();  
 return \*this;  
}  
  
FieldIterator FieldIterator::operator++(int) {  
 FieldIterator prev = \*this;  
 ++(\*this);  
 return prev;  
}  
  
FieldIterator &FieldIterator::operator--() {  
 if (rootGrid->isValidIndexes(posX - 1, posY)) {  
 posX--;  
 return \*this;  
 }  
 if (rootGrid->isValidIndexes(rootGrid->getWidth() - 1, posY - 1)) {  
 posX = rootGrid->getWidth() - 1;  
 posY--;  
 return \*this;  
 }  
 posX = 0; // end-значение  
 posY = rootGrid->getHeight();  
 return \*this;  
  
}  
  
FieldIterator FieldIterator::operator--(int) {  
 auto prev = \*this;  
 --(\*this);  
 return prev;  
}

Название файла: Grid.cpp

#include "Grid.h"  
  
Grid::Grid() {  
 height = 0;  
 width = 0;  
 grid = nullptr;  
}  
  
Grid::Grid(int height, int width, Cell \*\*grid) {  
 this->height = std::min(UP\_LIMIT\_HEIGHT, std::max(DOWN\_LIMIT\_HEIGHT, height));  
 this->width = std::min(UP\_LIMIT\_WIDTH, std::max(DOWN\_LIMIT\_WIDTH, width));  
 this->grid = new Cell\*[this->height];  
 for (int i = 0; i < this->height; ++i) {  
 this->grid[i] = new Cell[this->width];  
 }  
 if (grid != nullptr) {  
 for (int i = 0; i < this->height; ++i) {  
 for (int j = 0; j < this->width; ++j) {  
 this->grid[i][j] = grid[i][j];  
 }  
 }  
 }  
}  
  
Grid::Grid(const Grid &grid) {  
 this->grid = new Cell\*[grid.height];  
 for (int i = 0; i < grid.height; ++i) {  
 this->grid[i] = new Cell[grid.width];  
 }  
}  
  
Grid &Grid::operator=(const Grid &grid) {  
 if (&grid == this)  
 return \*this;  
 for (int i = 0; i < this->height; ++i) {  
 delete[] this->grid[i];  
 }  
 delete[] this->grid;  
 this->grid = new Cell\*[grid.height];  
 for (int i = 0; i < grid.height; ++i) {  
 this->grid[i] = new Cell[grid.width];  
 for (int j = 0; j < grid.width; ++j) {  
 this->grid[i][j] = grid.grid[i][j];  
 }  
 }  
 height = grid.height;  
 width = grid.width;  
 return \*this;  
}  
  
Grid::Grid(Grid &&grid): grid(grid.grid), height(grid.height), width(grid.width) {  
 grid.grid = nullptr;  
 grid.height = 0;  
 grid.width = 0;  
}  
  
Grid& Grid::operator=(Grid &&grid) {  
 if (&grid == this)  
 return \*this;  
 for (int i = 0; i < this->height; ++i) {  
 delete[] this->grid[i];  
 }  
 delete[] this->grid;  
 this->grid = grid.grid;  
 this->height = grid.height;  
 this->width = grid.width;  
 grid.grid = nullptr;  
 grid.height = 0;  
 grid.width = 0;  
 return \*this;  
}  
  
Grid::~Grid() {  
 for (int i = 0; i < height; ++i) {  
 delete[] grid[i];  
 }  
 delete[] grid;  
}  
  
bool Grid::isValidIndexes(int x, int y) const {  
 return isValidXPos(x) && isValidYPos(y);  
}  
  
bool Grid::isValidSizes(int height, int width) const {  
 return isValidHeight(height) && isValidWidth(width);  
}  
  
bool Grid::isValidHeight(int height) {  
 return DOWN\_LIMIT\_HEIGHT <= height && height <= UP\_LIMIT\_HEIGHT;  
}  
  
bool Grid::isValidWidth(int width) {  
 return DOWN\_LIMIT\_WIDTH <= width && width <= UP\_LIMIT\_WIDTH;  
}  
  
bool Grid::isValidXPos(int x) const {  
 return 0 <= x && x < this->width;  
}  
  
  
bool Grid::isValidYPos(int y) const {  
 return 0 <= y && y < this->height;  
}  
  
int Grid::getHeight() const {  
 return height;  
}  
  
int Grid::getWidth() const {  
 return width;  
}  
  
void Grid::setElem(CellPoint point, Cell cell) {  
 if (isValidIndexes(point.getX(), point.getY())) {  
 grid[point.getY()][point.getX()] = cell;  
 } else {  
 throw -1;  
 }  
}  
  
Cell Grid::getElem(CellPoint point) const {  
 if(isValidIndexes(point.getX(), point.getY())) {  
 return grid[point.getY()][point.getX()];  
 } else {  
 throw -1;  
 }  
}  
  
void Grid::clear() {  
 for (int i = 0; i < this->height; ++i) {  
 delete[] grid[i];  
 }  
 delete[] grid;  
 height = 0;  
 width = 0;  
 grid = nullptr;  
}  
  
void Grid::resizeCleanGrid(int height, int width, Cell \*\*newGrid) {  
 if (!isValidSizes(height, width)) {  
 throw -1;  
 }  
 clear();  
 grid = new Cell\*[height];  
 for (int i = 0; i < height; ++i) {  
 grid[i] = new Cell[width];  
 if (newGrid != nullptr) {  
 for (int j = 0; j < width; ++j) {  
 grid[i][j] = newGrid[i][j];  
 }  
 }  
 }  
 this->height = height;  
 this->width = width;  
}

Название файла: Cell.cpp

#include "Cell.h"  
  
CellObject Cell::getValue() const {  
 return this->value;  
}  
  
void Cell::setValue(CellObject value) {  
 this->value = value;  
}

Название файла: Field.cpp

#include "Field.h"  
  
Field::Field() {}  
  
  
Field::Field(int height, int width, CellPoint start, CellPoint finish, Grid grid) {  
 if (grid.grid != nullptr)  
 this->field = grid;  
 else  
 this->field = Grid(height, width);  
 this->start = start;  
 this->finish = finish;  
 if (isCorrectStartFinish(start, finish)) {  
 chosenStartFinish = true;  
 }  
}  
  
Field::Field(const Field &field) {  
 this->field = field.field;  
 this->start = field.start;  
 this->finish = field.finish;  
 this->wayGenerated = field.wayGenerated;  
 this->wallsGenerated = field.wallsGenerated;  
 this->chosenStartFinish = field.chosenStartFinish;  
 this->countWalls = field.countWalls;  
}  
  
Field &Field::operator=(const Field &field) {  
 if (&field == this)  
 return \*this;  
 this->field = field.field;  
 start = field.start;  
 finish = field.finish;  
 wayGenerated = field.wayGenerated;  
 wallsGenerated = field.wallsGenerated;  
 chosenStartFinish = field.chosenStartFinish;  
 countWalls = field.countWalls;  
 return \*this;  
}  
  
Field::Field(Field &&field) {  
 this->field = std::move(field.field);  
 this->start = field.start;  
 this->finish = field.finish;  
 this->chosenStartFinish = field.chosenStartFinish;  
 this->wayGenerated = field.wayGenerated;  
 this->wallsGenerated = field.wallsGenerated;  
 this->countWalls = field.countWalls;  
 field.start = CellPoint();  
 field.finish = CellPoint();  
 field.chosenStartFinish = false;  
 field.wayGenerated = false;  
 field.wallsGenerated = false;  
 field.countWalls = 0;  
}  
  
Field &Field::operator=(Field &&field) {  
 if (&field == this)  
 return \*this;  
 this->field = std::move(field.field);  
 this->start = field.start;  
 this->finish = field.finish;  
 this->chosenStartFinish = field.chosenStartFinish;  
 this->wayGenerated = field.wayGenerated;  
 this->wallsGenerated = field.wallsGenerated;  
 this->countWalls = field.countWalls;  
 field.start = CellPoint();  
 field.finish = CellPoint();  
 field.chosenStartFinish = false;  
 field.wayGenerated = false;  
 field.wallsGenerated = false;  
 field.countWalls = 0;  
 return \*this;  
}  
  
bool Field::isCorrectStartFinish(CellPoint start, CellPoint finish) const {  
 return this->field.isValidIndexes(start.getX(), start.getY()) &&  
 this->field.isValidIndexes(finish.getX(), finish.getY()) &&  
 isCorrectDistStartFinish(start, finish);  
}  
  
bool Field::isCorrectDistStartFinish(CellPoint start, CellPoint finish) const {  
 return abs(start.getX() - finish.getX()) +  
 abs(start.getY() - finish.getY()) >= distStartFinish;  
}  
  
  
CellPoint Field::generateBorderPoint() const {  
 switch (rand() % 4) {  
 case 0:  
 return {0,  
 rand() % this->field.getHeight()};  
 case 1:  
 return {this->field.getWidth() - 1,  
 rand() % this->field.getHeight()};  
 case 2:  
 return {rand() % this->field.getWidth(),  
 0};  
 case 3:  
 return {rand() % this->field.getWidth(),  
 this->field.getHeight() - 1};  
 }  
 return {0, 0};  
}  
  
void Field::generateStartFinishWay() {  
 srand(time(0));  
 distStartFinish = std::max((field.getWidth() + field.getHeight()) / 2, 2);  
 while (!isCorrectDistStartFinish(this->start, this->finish)) {  
 this->start = generateBorderPoint(); // нельзя выносить за while, тк возможна генерация в середине сетки,  
 this->finish = generateBorderPoint(); // что сократит макс кол-во длины на половину  
 }  
 chosenStartFinish = true;  
 field.setElem(start,  
 Cell(CellObject(TypeCell::*START*, TypeObject::*NOTHING*)));  
 generateWayWithoutWalls(this->start, this->finish);  
 field.setElem(finish,  
 Cell(CellObject(TypeCell::*FINISH*, TypeObject::*NOTHING*)));  
 wayGenerated = true;  
}  
  
void Field::generateWayWithoutWalls(CellPoint start, CellPoint finish) {  
 auto calcDist = [](int stx, int sty, int finx, int finy) {  
 return abs(stx - finx) + abs(sty - finy);  
 }; // функция подсчёта расстояния между точками  
  
 auto rightWay = [this, &calcDist, &finish](int stx, int sty, int curDist) {  
 return 0 <= stx && stx < field.getWidth() && 0 <= sty &&  
 sty < field.getHeight() && calcDist(stx, sty, finish.getX(), finish.getY()) < curDist;  
 }; // функция для контроля приближения к финишной точке  
   
 int dist = calcDist(start.getX(), start.getY(), finish.getX(), finish.getY());  
 srand(time(0));  
 int deltaX = -(start.getX() - finish.getX()) /  
 std::max(1, abs(start.getX() - finish.getX()));  
 int deltaY = -(start.getY() - finish.getY()) /  
 std::max(1, abs(start.getY() - finish.getY()));  
 int stx = start.getX(), sty = start.getY(), finx = finish.getX(), finy = finish.getY();  
 while (stx != finx || sty != finy) {  
 switch (rand() % 2) {  
 case 0:  
 if (rightWay(stx + deltaX, sty, dist)) { // нужно, чтобы блокировать способ, который уже не нужен  
 stx += deltaX;  
 dist = calcDist(stx, sty, finx, finy);  
 field.setElem(CellPoint(stx, sty), Cell(  
 CellObject(TypeCell::*WAY*, TypeObject::*NOTHING*)));  
 }  
 break;  
 case 1:  
 if (rightWay(stx, sty + deltaY, dist)) { // нужно, чтобы блокировать способ, который уже не нужен  
 sty += deltaY;  
 dist = calcDist(stx, sty, finx, finy);  
 field.setElem(CellPoint(stx, sty), Cell(  
 CellObject(TypeCell::*WAY*, TypeObject::*NOTHING*)));  
 }  
 break;  
 }  
 }  
}  
  
void Field::generateWalls(int countWalls) {  
 if (!wayGenerated)  
 throw -1; // путь не сгенерирован  
 if ((double) countWalls / (field.getWidth() \* field.getHeight()) \* 100 > PERCENT\_WALLS) {  
 countWalls = double(PERCENT\_WALLS) / 100 \* (field.getWidth() \* field.getHeight());  
 std::cerr << "Слишком много стен. Количество уменьшено до " << countWalls << '\n';  
 }  
 int cntSetWalls = 0;  
 srand(time(0));  
 while (cntSetWalls < countWalls) {  
 CellPoint point = CellPoint(rand() % field.getWidth(), rand() % field.getHeight());  
 if (field.getElem(point).getValue().getTypeCell() == TypeCell::*EMPTY*) {  
 field.setElem(point, Cell(CellObject(TypeCell::*WALL*, TypeObject::*NOTHING*)));  
 cntSetWalls++;  
 }  
 }  
 this->countWalls = countWalls;  
 wallsGenerated = true;  
}  
  
bool Field::generateFullField(int countWalls) {  
 this->generateStartFinishWay();  
 this->generateWalls(countWalls);  
 return this->getStatusStartFinish() && this->getStatusWay() && this->getStatusWalls();  
}  
  
void Field::cleanStartFinishWay() { // удаляет путь между start и finish, но сами точки не трогает  
 field.setElem(start,  
 Cell(CellObject(TypeCell::*START*, TypeObject::*NOTHING*)));  
  
 int deltaX = -(start.getX() - finish.getX()) /  
 std::max(1, abs(start.getX() - finish.getX()));  
 int deltaY = -(start.getY() - finish.getY()) /  
 std::max(1, abs(start.getY() - finish.getY()));  
 int stx = start.getX(), sty = start.getY(), finx = finish.getX(), finy = finish.getY();  
 while (stx != finx || sty != finy) {  
 if (this->field.isValidIndexes(stx + deltaX, sty) && (  
 field.getElem(CellPoint(stx + deltaX, sty)).getValue().getTypeCell() == TypeCell::*WAY* ||  
 field.getElem(CellPoint(stx + deltaX, sty)).getValue().getTypeCell() == TypeCell::*FINISH*)  
 ) {  
 stx += deltaX;  
 field.setElem(CellPoint(stx, sty), Cell(CellObject(TypeCell::*EMPTY*, TypeObject::*NOTHING*)));  
 } else if (this->field.isValidIndexes(stx, sty + deltaY) && (  
 field.getElem(CellPoint(stx, sty + deltaY)).getValue().getTypeCell() == TypeCell::*WAY* ||  
 field.getElem(CellPoint(stx, sty + deltaY)).getValue().getTypeCell() == TypeCell::*FINISH*)  
 ) {  
 sty += deltaY;  
 field.setElem(CellPoint(stx, sty), Cell(CellObject(TypeCell::*EMPTY*, TypeObject::*NOTHING*)));  
 }  
 }  
 wayGenerated = false;  
 field.setElem(finish,  
 Cell(CellObject(TypeCell::*FINISH*, TypeObject::*NOTHING*)));  
}  
  
void Field::moveHero(CellPoint to) {  
 if (getElem(to).getValue().getTypeCell() != TypeCell::*WALL*) {  
 setElem(heroPos, CellObject(getElem(heroPos).getValue().getTypeCell(), TypeObject::*NOTHING*));  
 setElem(to, CellObject(getElem(to).getValue().getTypeCell(), TypeObject::*HERO*));  
 heroPos = to;  
 }  
}  
  
void Field::printField() {  
 for (int i = 0; i < field.getWidth() + 2; ++i) {  
 std::cout << '\_';  
 }  
 std::cout << '\n';  
 for (int i = 0; i < field.getHeight(); ++i) {  
 std::cout << '|';  
 for (int j = 0; j < field.getWidth(); ++j) {  
 std::cout << field.getElem(CellPoint(j, i)).getValue().getCellAsChar();  
 }  
 std::cout << '|';  
 std::cout << '\n';  
 }  
 for (int i = 0; i < field.getWidth() + 2; ++i) {  
 std::cout << '\_';  
 }  
 std::cout << '\n';  
}  
  
Cell Field::getElem(CellPoint point) const {  
 return field.getElem(point);  
}  
  
void Field::setElem(CellPoint point, CellObject object) {  
 field.setElem(point, Cell(object));  
}  
  
int Field::getHeight() const {  
 return field.height;  
}  
  
int Field::getWidth() const {  
 return field.width;  
}  
  
bool Field::getStatusWay() const {  
 return wayGenerated;  
}  
  
bool Field::getStatusWalls() const {  
 return wallsGenerated;  
}  
  
bool Field::getStatusStartFinish() const {  
 return chosenStartFinish;  
}  
  
FieldIterator Field::getFieldIterator() {  
 return FieldIterator(&(this->field));  
}  
  
CellPoint Field::getHeroPos() const{  
 return heroPos;  
}  
  
void Field::setHeroOnStart() {  
 field.setElem(start, Cell(CellObject(TypeCell::*START*, TypeObject::*HERO*)));  
 heroPos = start;  
}

Название файла: FieldScreen.cpp

#include "FieldScreen.h"  
  
  
FieldScreen::FieldScreen() {  
 field = nullptr;  
}  
  
void FieldScreen::showStartingParams() {  
 std::cout << "Введите значения параметров:\n";  
 bool acceptedParams = false;  
 auto enterSizeValue = [](  
 int &val,  
 int height,  
 int width,  
 const char title[],  
 bool (\*compare\_bad)(int, int, int)  
 ) {  
 while (val < 0) {  
 try {  
 std::cout << title;  
 std::cin >> val;  
 if (std::cin.fail())  
 throw -1;  
 } catch (int) {  
 std::cin.clear(); // убирает флаг fail с cin  
 std::cin.ignore(32767, '\n');  
 std::cout << "Введённое значение неверно.\n";  
 val = -1;  
 continue;  
 }  
 if (compare\_bad(val, height, width)) {  
 std::cout << "Введённое значение неверно.\n";  
 std::cin.clear();  
 std::cin.ignore(32767, '\n');  
 val = -1;  
 }  
 }  
 };  
 int height, width, countWalls;  
 while (!acceptedParams) {  
 height = width = countWalls = -1;  
 enterSizeValue(height, height, width, ("Высота поля (мин = " + std::to\_string(DOWN\_LIMIT\_HEIGHT) + ", макс = " + std::to\_string(UP\_LIMIT\_HEIGHT) + "): ").c\_str(),  
 [](int val, int, int) { return !Grid::isValidHeight(val); });  
 enterSizeValue(width, height, width, ("Ширина поля (мин = " + std::to\_string(DOWN\_LIMIT\_WIDTH) + ", макс = " + std::to\_string(UP\_LIMIT\_WIDTH) +"): ").c\_str(),  
 [](int val, int, int) { return !Grid::isValidWidth(val); });  
 enterSizeValue(countWalls, height, width,  
 ("Количество непроходимых клеток (макс процент покрытия - " + std::to\_string(PERCENT\_WALLS) + "% от площади поля): ").c\_str(),  
 [](int val, int height, int width) {  
 return (double) val / (width \* height) \* 100 > PERCENT\_WALLS;  
 });  
 std::cout << "Значения приняты. Сгенерировать поле? (y - сгенерировать / n - изменить параметры) ";  
 char acceptSymbol = getchar(); // считываем лишний символ после ввода числа непроходимых клеток  
 while (true) {  
 acceptSymbol = getchar();  
 if (acceptSymbol != 'y' && acceptSymbol != 'n') {  
 std::cout  
 << "Неверное значение, попробуйте снова. Сгенерировать поле? (y - сгенерировать/n - изменить параметры) ";  
 std::cin.clear();  
 std::cin.ignore(32767, '\n');  
 } else break;  
 }  
 if (acceptSymbol == 'y')  
 acceptedParams = true;  
 }  
 field = new Field(height, width);  
 if(field->generateFullField(countWalls))  
 field->setHeroOnStart();  
 else {  
 std::cout << "Не удалось сгенерировать поле!\n";  
 throw -1;  
 }  
}  
  
void FieldScreen::showUpdatedScreen() const {  
 for (int i = 0; i < field->getWidth() + 2; ++i) {  
 std::cout << '\_';  
 }  
 std::cout << '\n';  
 auto fieldIterator = field->getFieldIterator();  
 for (int i = 0; i < field->getHeight(); ++i) {  
 std::cout << '|';  
 for (int j = 0; j < field->getWidth(); ++j) {  
 char c = (fieldIterator++).getElem().getValue().getCellAsChar();  
 std::cout << (c == '#' ? ' ' : c);  
 }  
 std::cout << '|';  
 std::cout << '\n';  
 }  
 for (int i = 0; i < field->getWidth() + 2; ++i) {  
 std::cout << '\_';  
 }  
 std::cout << '\n';  
}  
  
bool FieldScreen::registerMovement(char &action) {  
 action = getchar();  
 std::cin.clear();  
 std::cin.ignore(32767, '\n');  
 CellPoint heroPos = field->getHeroPos();  
 switch (tolower(action)) {  
 case MoveSide::*UP*:  
 requestMoveObject(heroPos, CellPoint(heroPos.getX(), heroPos.getY() - 1));  
 return true;  
 case MoveSide::*DOWN*:  
 requestMoveObject(heroPos, CellPoint(heroPos.getX(), heroPos.getY() + 1));  
 return true;  
 case MoveSide::*LEFT*:  
 requestMoveObject(heroPos, CellPoint(heroPos.getX() - 1, heroPos.getY()));  
 return true;  
 case MoveSide::*RIGHT*:  
 requestMoveObject(heroPos, CellPoint(heroPos.getX() + 1, heroPos.getY()));  
 return true;  
 case MoveSide::*EXIT*:  
 return false;  
 default:  
 std::cout << "Команда не распознана.\n";  
 }  
 return false;  
}  
  
void FieldScreen::requestMoveObject(CellPoint from, CellPoint to) {  
 if (!field->field.isValidIndexes(to.getX(), to.getY()))  
 return;  
 if (field->getHeroPos() == from) {  
 field->moveHero(to);  
 }  
}  
  
void FieldScreen::showStartFieldScreen() {  
 showStartingParams();  
 std::system("clear");  
}  
  
void FieldScreen::gameStatusObserver() {  
 char action = getchar(); // считываем перенос строки  
 std::cout << "Для выхода введите ` и нажмите enter.\n";  
 showUpdatedScreen();  
 while (action != '`') {  
 bool goodMovement = registerMovement(action);  
 if (goodMovement) {  
 std::system("clear");  
 std::cout << "Для выхода введите ` и нажмите enter.\n";  
 showUpdatedScreen();  
 }  
 }  
 std::system("clear");  
}

Название файла: main.cpp

#include <iostream>  
#include "UI/Models/Field.h"  
#include "UI/FieldScreen.h"  
  
  
#define GAME // Запускать через терминал  
//#define DEMO  
//#define FEATURES  
  
  
  
int main() {  
#ifdef GAME  
 std::setlocale(LC\_ALL, "");  
 FieldScreen mainScreen;  
 mainScreen.showStartFieldScreen();  
 mainScreen.gameStatusObserver();  
 system("clear");  
#endif  
  
#ifdef DEMO  
 int h = 30;  
 int w = 30;  
 int countWalls = 500;  
 int fullTime = clock();  
  
 Field field1 = Field(h, w);  
 field1.generateFullField(countWalls);  
 std::cout << "Full generate time: " << double(clock() - fullTime) / CLOCKS\_PER\_SEC << '\n';  
 std::cout << "Print with func:\n";  
 field1.printField();  
 auto iter = field1.getFieldIterator();  
 std::cout << "Print with iterator:\n";  
 for (int i = 0; i < field1.getHeight(); ++i) {  
 for (int j = 0; j < field1.getWidth(); ++j) {  
 std::cout << iter.getElem().getValue().getCellAsChar();  
 iter++;  
 }  
 std::cout << '\n';  
 }  
  
 std::cout << "Generate second Field\n";  
 Field field2 = Field(h+1, w+1);  
 field2.generateFullField(countWalls);  
  
 int copyTime = clock();  
 Field copyField = field1;  
 std::cout << "Full copy constr time: " << double(clock() - copyTime) / CLOCKS\_PER\_SEC << '\n';  
 copyTime = clock();  
 copyField = field2;  
 std::cout << "Full copy operator= time: " << double(clock() - copyTime) / CLOCKS\_PER\_SEC << '\n';  
 FieldIterator copyIter = copyField.getFieldIterator();  
 for (int i = 0; i < 5; ++i) {  
 copyIter.setElem(Cell(CellObject(TypeCell::START, TypeObject::NOTHING)));  
 copyIter++;  
 }  
 std::cout << "Print copied field1:\n";  
 copyField.printField();  
  
 int moveTime = clock();  
 Field moveField = std::move(field1);  
 std::cout << "Full move constr time: " << double(clock() - moveTime) / CLOCKS\_PER\_SEC << '\n';  
 moveTime = clock();  
 moveField = std::move(field2);  
 std::cout << "Full move operator= time: " << double(clock() - moveTime) / CLOCKS\_PER\_SEC << '\n';  
 FieldIterator moveIter = moveField.getFieldIterator();  
 moveIter.moveTo(5, 0);  
 for (int i = 0; i < 5; ++i) {  
 moveIter.setElem(Cell(CellObject(TypeCell::FINISH, TypeObject::NOTHING)));  
 moveIter++;  
 }  
 std::cout << "Print moved field1:\n";  
 moveField.printField();  
 std::cout << "Cleaned way-symbols\n";  
 moveField.cleanStartFinishWay();  
 moveField.printField();  
#endif  
  
#ifdef FEATURES  
 int h = 30;  
 int w = 30;  
 int countWalls = 400;  
 Field field = Field(h, w);  
 field.generateFullField(countWalls);  
 field.cleanStartFinishWay();  
#endif  
 return 0;  
}

# Приложение Б UML-диаграмма классов