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

Студентка гр. 0382 _____ Охотникова Г.С.

Преподаватель _____ Жангиров Т.Р.

Санкт-Петербург

2021

Цель работы.

Освоить создание классов, конструкторов и методов в языке C++.

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.

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.

Выполнение работы.

В ходе выполнения лабораторной работы было реализовано 6 классов, а именно: Cell, Floor, FloorBuilder, FloorBuilderTree, FloorDirector, GameStart.

Class Cell:

Это класс одной отдельной клетки поля.

В классе созданы вспомогательные enum классы Elem и Type, в которых содержится информация о том, что находится на клетке и о типе клетки соответственно.

В классе объявлены приватные поля:

- *Elem _elem* — элемент, находящийся на клетке.
- T*ype _type* — тип клетки (начало/конец/стена/обычная).
- *sf::Vector2i _position* — расположение клетки на поле.
- *sf::Sprite* _sprite* — спрайт клетки.
- *sf::Texture* _texture* — картинка клетки.

Реализован конструктор класса *Cell(const std::string& filename, sf::Vector2i position, Elem elem , Type type)*. Принимает на вход файл, который будет текстурой данной клетки, расположение на поле, то, что находится на ней и то, какого она типа.

Публичные методы класса:

- *void setTexture(const std::string& filename)* — метод, который позволяет установить текстуру.
- *sf::Sprite* getSprite()* — метод, который позваляет узнать спрайт клетки.
- *void setElem(Elem elem)* — метод, который позволяет установить элемент находящийся поверх клетки.
- *Elem getElem()* — возвращает тип элемента находящегося на клетке.
- void setType(Type type) — устанавливает тип клетки.
- *Type getType* — возвращает тип клетки.

Class Field:

Это класс поля состоящий из массива клеток и некоторых параметров поля.

В классе объявлены публичные поля:

- *Cell*** cells* — двумерный массив ссылок на клетки.
- *sf::Vector2i entry, exit* — кординаты входа и выхода.
- *int width, heigth* — ширина и высота.

Реализован конструктор, который создает двумерный массив, содержащий в себе ссылки на клетки поля. Также реализованы конструкторы копирования и перемещения и соответствующие операторы для них.

Методы класса:

- *void Draw_Floor(sf::RenderWindow* window)* — отрисовывает клетки в окне.

В классе реализован деструктор, в котором память, выделенная под двумерный массив, освобождается.

Class FloorBuilder:

Это виртуальный класс, для постройки элемента класса Floor. Он состоит из четырёх методов генерирующих определённые клетки в поле(вход, выход, стены, обычные клетки).

Class FieldBuilderTree:

Это единственный строитель поля.

В классе объявлено публичное поле:

- *Floor* floor* — ссылка поле, с которым будет работать строитель

Реализованы конструктор и деструктор, которые создают поле и удаляют его соответственно.

Методы класса:

- *void GenerateNormalCells()* — заполняет всё поле проходимыми клетками(если надо, они впоследствии заменяются на другие).

- void GenerateWallCells() — на границах поля создаёт стены, после чего вызывает *GenerateWallDangeonCells()*.
- *void GenerateWallDangeonCells(int x1, int y1, int x2, int y2, int** mass)* — рекурсивно делит поле на прямоугольники размера не меньше, чем заданный, параллельно создавая стены и проходы в этих стенах.
- *void GenerateEntryCells()* — создаёт клетку входа.
- *void GenerateExitCells()* — создаёт клетку выхода.
- *Void Reset()* — возвращяет поле к начальному состоянию.
- *Floor* getFloor()*— возвращает поле, которое хранит класс.

Class FloorDirector:

Это класс определяюший как будет построено поле.

В классе объявлено приватное поле:

- *FloorBuilder* builder* — какой нибудь строитель, который мы будем использывать

Реализован конструктор, который устанавливает в приватное поле отпределённый builder.

Методы класса:

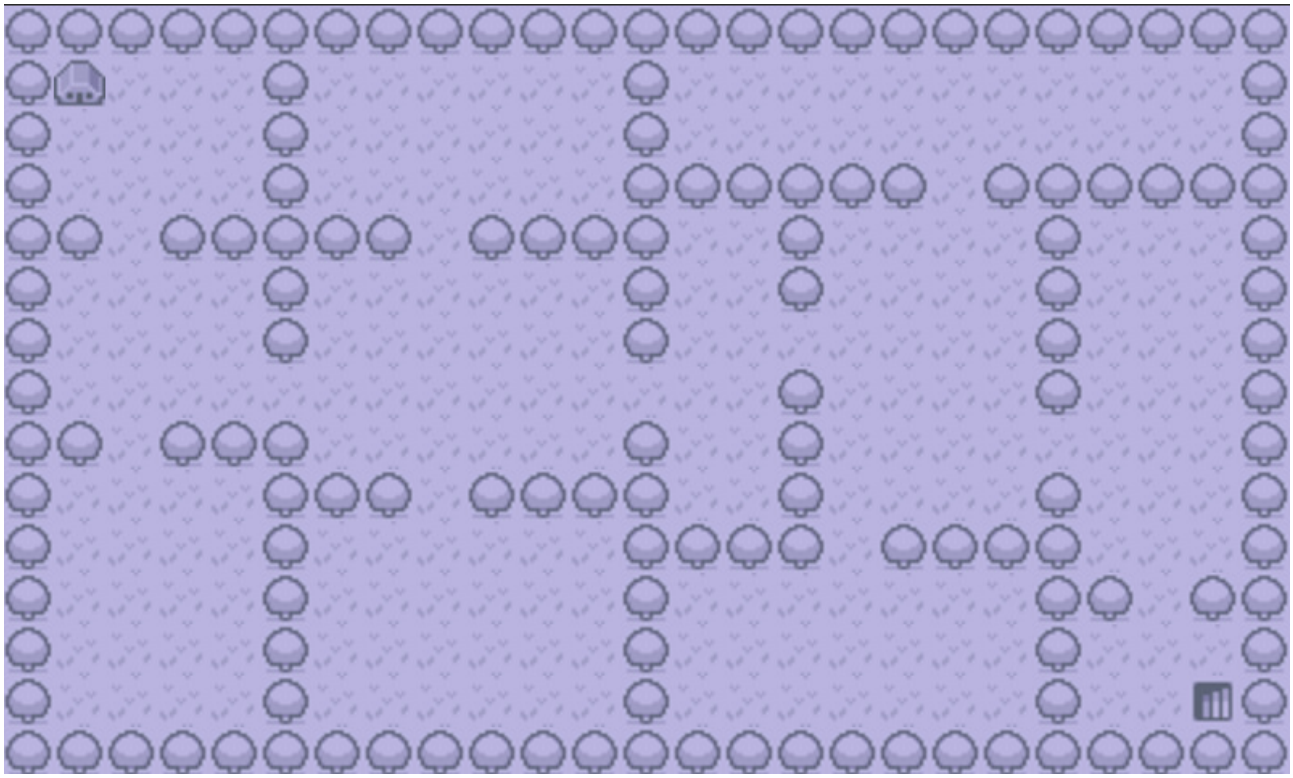
- *void setBuilder(FloorBuilder* b)* — устанавливает определённый builder.
- *void Builder_FloorBuilderTree()* — вызывает методы строителя.

Class GameStart:

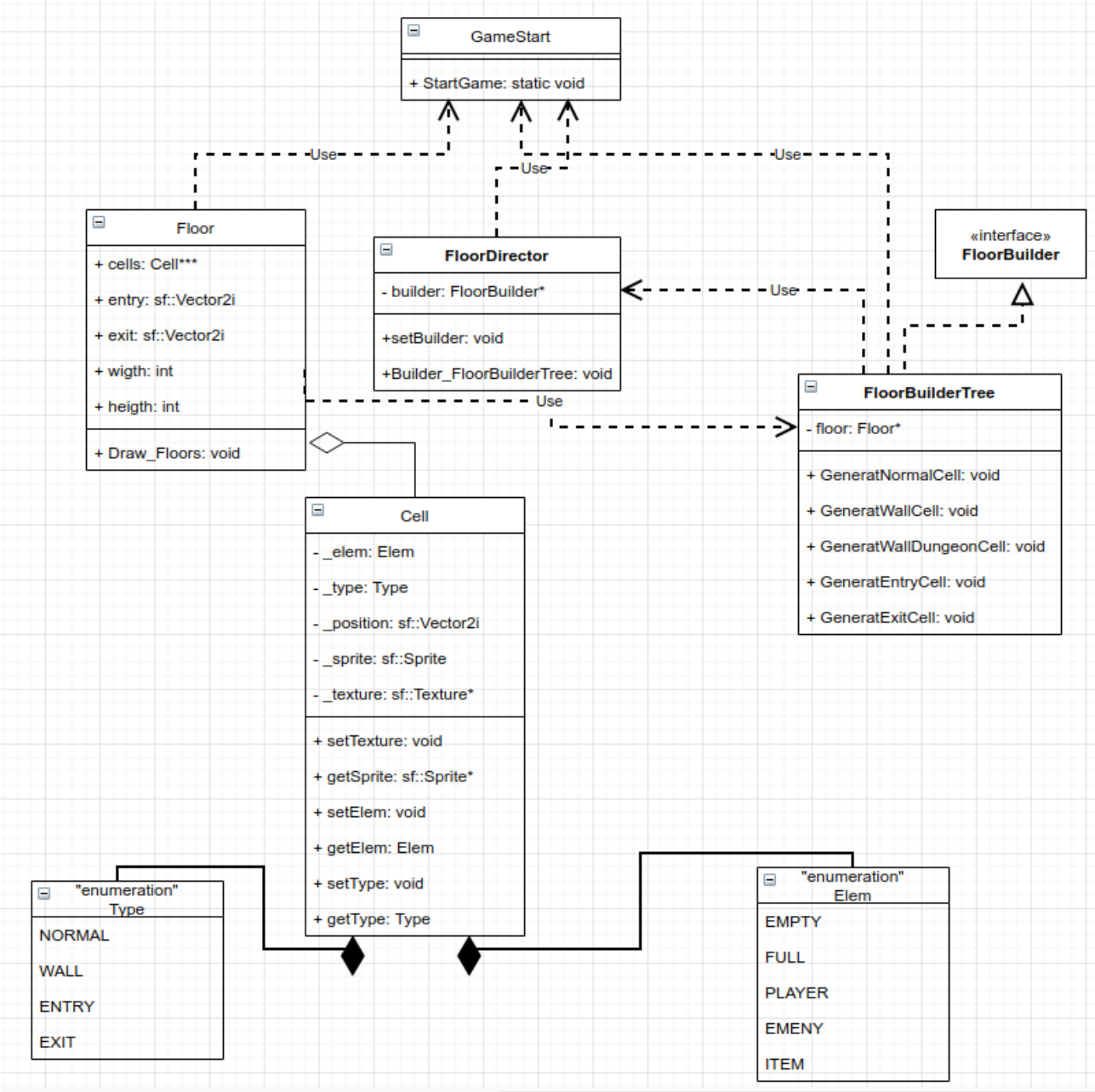
Класс запускающий всё, содержит один метод, который создаёт окно, после чего создаёт строителя, по создаёт нему поле, после чего дожидается пока закроют окно.

Тестирование.

Результат работы программы:



UML диаграммаю



Выводы.

Было исследовано создание классов, методом и конструкторов на языке C++. Разработана программа, которая создает игровое поле и выводит его в окне.

ПРИЛОЖЕНИЕ А
ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: main.cpp

```
#include "GameStart.h"

int main()
{
   GameStart::StartGame();
   return 0;
}
```

Название файла: GameStart.h

```
#ifndef GAME_GAMESTART_H
#define GAME_GAMESTART_H

#include "Floor/FloorDirector.h"
#include "Floor/FloorBuilderTree.h"

class GameStart {
public:
   static void StartGame();
};



#endif //GAME_GAMESTART_H
```

Название файла: GameStart.cpp

```
#include "GameStart.h"

void GameStart::StartGame() {
```

```
        sf::RenderWindow window(sf::VideoMode(WIDTH_OF_FLOOR*WIDTH,
HEIGTH_OF_FLOOR>
        auto builder = new FloorBuilderTree();
        auto director = FloorDirector(builder);
        director.Builder_FloorBuilderTree();
        Floor* floor = builder->getFloor();
        while (window.isOpen()) {
            sf::Event event;
            while (window.pollEvent(event)) {
                if (event.type == sf::Event::Closed)
                    window.close();
            }
            floor->Draw_Floor(&window);
            window.display();
        }
        delete builder;
    }
```

Название файла: FloorDirector.h

```
#ifndef GAME_FLOORDIRECTOR_H
#define GAME_FLOORDIRECTOR_H

#include "FloorBuilder.h"

class FloorDirector {
private:
    FloorBuilder* builder;

public:
    FloorDirector(FloorBuilder* b);
```

```
    void setBuilder(FloorBuilder* b);

    void Builder_FloorBuilderTree();

};


#endif //GAME_FLOORDIRECTOR_H
```

Название файла: FloorDirector.cpp

```
#include "FloorDirector.h"

FloorDirector::FloorDirector(FloorBuilder* b){
    this->builder = b;
};

void FloorDirector::setBuilder(FloorBuilder* b){
    this->builder = b;
};

void FloorDirector::Builder_FloorBuilderTree(){
    this->builder->GeneratNormalCells();
    this->builder->GeneratWallCells();
    this->builder->GeneratEntryCell();
    this->builder->GeneratExitCell();
};
```

Название файла: FloorBuilderTree.h

```
#ifndef GAME_FLOORBUILDERTREE_H
#define GAME_FLOORBUILDERTREE_H
```

```
/*
#define NORMAL_TEXTURE_PATH "../Game_Obj/Tiles/tile_0019.png"
#define ENTRY_TEXTURE_PATH "../Game_Obj/Tiles/tile_0048.png"
#define EXIT_TEXTURE_PATH "../Game_Obj/Tiles/tile_0083.png"
#define WALL_TEXTURE_PATH "../Game_Obj/Tiles/tile_0013.png"
*/
#define NORMAL_TEXTURE_PATH "../Game_Obj/Normal.jpg"
#define ENTRY_TEXTURE_PATH "../Game_Obj/Entry.jpg"
#define EXIT_TEXTURE_PATH "../Game_Obj/Exit.jpg"
#define WALL_TEXTURE_PATH "../Game_Obj/Wall.jpg"

#include "FloorBuilder.h"
#include <cstdlib>

class FloorBuilderTree : public FloorBuilder{
private:
    Floor* floor;

public:
    FloorBuilderTree();
    ~FloorBuilderTree();

    void GeneratNormalCells() override;
    void GeneratWallCells() override;
    void GeneratWallDungeonCells(int x1, int y1, int x2, int y2, int** mass);
    void GeneratEntryCell() override;
    void GeneratExitCell() override;

    void Reset();
    Floor* getFloor();
};
```

```
#endif //GAME_FLOORBUILDERTREE_H
```

Название файла: FloorBuilderTree.cpp

```
#include "FloorBuilderTree.h"

FloorBuilderTree::FloorBuilderTree(){
    this->floor = new Floor(WIDTH_OF_FLOOR, HEIGTH_OF_FLOOR);
};

FloorBuilderTree::~FloorBuilderTree(){
    delete this->floor;
};

void FloorBuilderTree::GeneratNormalCells(){
    for (int i = 0; i < WIDTH_OF_FLOOR; ++i) {
        for (int j = 0; j < HEIGTH_OF_FLOOR; ++j) {
                this->floor->cells[i][j] = new Cell(NORMAL_TEXTURE_PATH,
sf::Vector2i(i, j), Elem::EMPTY, Type::NORMAL);
        }
    }
};
void FloorBuilderTree::GeneratWallCells(){
    for (int i = 0; i < WIDTH_OF_FLOOR; ++i) {
        for (int j = 0; j < HEIGTH_OF_FLOOR; ++j) {
                if (i == 0 || i == WIDTH_OF_FLOOR - 1 || j == 0 || j ==
HEIGTH_OF_FLOOR - 1){
            this->floor->cells[i][j]->setElem(Elem::FULL);
            this->floor->cells[i][j]->setType(Type::WALL);
            this->floor->cells[i][j]->setTexture(WALL_TEXTURE_PATH);
```

```
            }
        }
    }
    int** mass = new int*[WIDTH_OF_FLOOR];
        for (int i = 0; i < WIDTH_OF_FLOOR; ++i) mass[i] = new int[HEIGTH_OF_FLOOR];
    for (int i = 0; i < WIDTH_OF_FLOOR; ++i) {
        for (int j = 0; j < HEIGTH_OF_FLOOR; ++j) {
            mass[i][j] = 0;
        }
    }
            this->GeneratWallDungeonCells(0, 0, HEIGTH_OF_FLOOR-1, WIDTH_OF_FLOOR-1, mass);
};

void FloorBuilderTree::GeneratWallDungeonCells(int x1, int y1, int x2, int y2, int** mass){
    int small_side, big_side = 3;
    int wall_index = -1;
    int len_xside = x2 - x1 - 1;
    int len_yside = y2 - y1 - 1;
    if ((len_xside * len_yside >= MIN_SIZE_OF_FLOOR*3) && (len_xside >= 3) && (len_yside >= 3)){
        if (len_xside >= len_yside) {
            small_side = len_yside;
            while (small_side * big_side < MIN_SIZE_OF_FLOOR) {
                big_side++;
            }
            if ((x2 - big_side) - (x1 + big_side) + 1 > 2 ) {
                /*wall_index = ((x1 + x2) / 2);
                //while (wall_index == (int) ((x1 + x2) / 2)) {
```

```
                    wall_index = x1 + big_side + rand() % ((x2 - big_side) - (x1 +
big_side) + 1);
                //}
                for (int i = y1 + 1; i < y2 ; ++i) {
                  if (i != (int) ((y1 + y2) / 2)) {
                    this->floor->cells[i][wall_index]->setElem(Elem::FULL);
                    this->floor->cells[i][wall_index]->setType(Type::WALL);
                                                    this->floor->cells[i][wall_index]-
>setTexture(WALL_TEXTURE_PATH);
                  }
                }
                this->GeneratWallDungeonCells(wall_index, y1, x2, y2, mass);
                this->GeneratWallDungeonCells(x1, y1, wall_index, y2, mass);
              }*/
                while (wall_index == -1) {
                    wall_index = x1 + big_side + rand() % ((x2 - big_side) - (x1 +
big_side) + 1);
                  if(mass[y1][wall_index] == 1 || mass[y2][wall_index] == 1){
                    wall_index = -1;
                  }
                  else{
                    mass[(int)(y1+y2)/2][wall_index] = 1;
                  }
                }
                for (int i = y1 + 1; i < y2 ; ++i) {
                  if (i != (int) ((y1 + y2) / 2)) {
                    this->floor->cells[i][wall_index]->setElem(Elem::FULL);
                    this->floor->cells[i][wall_index]->setType(Type::WALL);
                                                    this->floor->cells[i][wall_index]-
>setTexture(WALL_TEXTURE_PATH);
                  }
```

```
                }
                this->GeneratWallDungeonCells(wall_index, y1, x2, y2, mass);
                this->GeneratWallDungeonCells(x1, y1, wall_index, y2, mass);
            }
        }else{
            small_side = len_xside;
            while (small_side * big_side < MIN_SIZE_OF_FLOOR){
                big_side++;
            }
            if ((y2 - big_side) - (y1 + big_side) + 1 > 2) {
                while (wall_index == -1) {
                    wall_index = y1 + big_side + rand() % ((y2 - big_side) - (y1 +
big_side) + 1);
                    if(mass[wall_index][x1] == 1 || mass[wall_index][x2] == 1){
                        wall_index = -1;
                    }
                    else{
                        mass[wall_index][(int)(x1+x2)/2] = 1;
                    }
                }
                for (int i = x1 + 1; i < x2 ; ++i) {
                    if (i != (int) ((x1 + x2) / 2)) {
                        this->floor->cells[wall_index][i]->setElem(Elem::FULL);
                        this->floor->cells[wall_index][i]->setType(Type::WALL);
                                                    this->floor->cells[wall_index][i]-
>setTexture(WALL_TEXTURE_PATH);
                    }
        }
                this->GeneratWallDungeonCells(x1, wall_index, x2, y2, mass);
                this->GeneratWallDungeonCells(x1, y1, x2, wall_index, mass);
            }
```

```
            }
        }
    };

    void FloorBuilderTree::GeneratEntryCell(){
        this->floor->cells[1][1]->setType(Type::ENTRY);
        this->floor->cells[1][1]->setTexture(ENTRY_TEXTURE_PATH);
    };

    void FloorBuilderTree::GeneratExitCell(){
            this->floor->cells[WIDTH_OF_FLOOR - 2][HEIGTH_OF_FLOOR - 2]-
>setType(Type::EXIT);
            this->floor->cells[WIDTH_OF_FLOOR - 2][HEIGTH_OF_FLOOR - 2]-
>setTexture(EXIT_TEXTURE_PATH);
    };

    void FloorBuilderTree::Reset(){
        this->floor = new Floor(WIDTH_OF_FLOOR, HEIGTH_OF_FLOOR);
    };

    Floor* FloorBuilderTree::getFloor(){
        Floor* result = this->floor;
        this->Reset();
        return result;
    };
```

Название файла: FloorBuilder.h

```
    #ifndef GAME_FLOORBUILDER_H
#define GAME_FLOORBUILDER_H

#include "Floor.h"
```

```
class FloorBuilder {
public:
    virtual void GeneratNormalCells() = 0;
    virtual void GeneratWallCells() = 0;
    virtual void GeneratEntryCell() = 0;
    virtual void GeneratExitCell() = 0;
};


#endif //GAME_FLOORBUILDER_H
```

Название файла: Floor.h

```
#ifndef GAME_FLOOR_H
#define GAME_FLOOR_H

#define WIDTH_OF_FLOOR 25
#define HEIGTH_OF_FLOOR 15
#define MIN_SIZE_OF_FLOOR 9


#include "Cell/Cell.h"

class Floor {
public:
    Cell*** cells;
    sf::Vector2i entry, exit;
    int width, height;


    Floor(int width, int height);
```

```
    Floor(const Floor& other);
    Floor& operator=(const Floor& other);
    Floor& operator=(Floor&& other);
    void Draw_Floor(sf::RenderWindow* window);
    ~Floor();
};


#endif //GAME_FLOOR_H
```

Название файла: Floor.cpp

```
#include "Floor.h"

Floor::Floor(int width, int height): width(width), height(height) {
    cells = new Cell**[WIDTH_OF_FLOOR];
            for (int i = 0; i < WIDTH_OF_FLOOR; i++) cells[i] = new
Cell*[HEIGTH_OF_FLOOR];
};

Floor::Floor(const Floor& other){
    *this = other;
}

Floor& Floor::operator=(const Floor& other) {
    this->width = other.width;
    this->height = other.height;
    for(int i = 0; i < WIDTH_OF_FLOOR; i++) {
        for (int j = 0; j < HEIGTH_OF_FLOOR; j++) {
            this->cells[i][j] = other.cells[i][j];
        }
    }
```

```
    return *this;
}

Floor& Floor::operator=(Floor&& other){
    this->width = other.width;
    other.width = 0;
    this->height = other.height;
    other.height = 0;
    for(int i = 0; i < WIDTH_OF_FLOOR; i++) {
        for (int j = 0; j < HEIGTH_OF_FLOOR; j++) {
            this->cells[i][j] = other.cells[i][j];
        }
    }
    return *this;
};

void Floor::Draw_Floor(sf::RenderWindow* window) {
    for (int i = 0; i < width; ++i) {
        for (int j = 0; j < height; ++j) {
            window->draw(*this->cells[i][j]->getSprite());
        }
    }
};

Floor::~Floor() {
    for (int i = 0; i < height; ++i) delete cells[i];
    delete cells;
}
```

Название файла: Cell.h

```
#ifndef GAME_CELL_H
```

```
#define GAME_CELL_H
#define WIDTH 32

#include "iostream"
#include "stdlib.h"
#include <SFML/Graphics.hpp>

enum class Elem : unsigned short {
    EMPTY,
    FULL,
    PLAYER,
    EMENY,
    ITEM
};

enum class Type : unsigned short {
    NORMAL,
    WALL,
    ENTRY,
    EXIT
};

class Cell {
private:
    Elem _elem;
    Type _type;
    sf::Vector2i _position;
    sf::Sprite* _sprite;
    sf::Texture* _texture;

public:
```

```
    Cell(const std::string& filename, sf::Vector2i position, Elem elem , Type type);
    Cell(const Cell&) = default;
    Cell(Cell&&) = default;
    ~Cell();

    void setTexture(const std::string& filename);
    sf::Sprite* getSprite();
    void setElem(Elem elem);
    Elem getElem();
    void setType(Type type);
    Type getType();
};


#endif //GAME_CELL_H
```

Название файла: Cell.h

```
#include "Cell.h"

Cell::Cell(const std::string& filename, sf::Vector2i position, Elem elem, Type type):
_position(position), _elem(elem), _type(type){
    _texture = new sf::Texture;
    _texture->loadFromFile(filename);
    _sprite = new sf::Sprite(*_texture);
    _sprite->setPosition(WIDTH*position.x, WIDTH*position.y);
}

void Cell::setTexture(const std::string& filename){
    _texture = new sf::Texture;
    _texture->loadFromFile(filename);
    _sprite = new sf::Sprite(*_texture);
```

```
    _sprite->setPosition(WIDTH*_position.x, WIDTH*_position.y);
};

sf::Sprite* Cell::getSprite(){
    return _sprite;
};

Cell::~Cell() {
    delete _texture;
    delete _sprite;
}

void Cell::setElem(Elem elem){
    _elem = elem;
};

Elem Cell::getElem(){
    return _elem;
};

void Cell::setType(Type type){
    _type = type;
};

Type Cell::getType(){
    return _type;
};
```