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»

Студент гр. 4384 _____ Водолазко В.О.

Преподаватель _____ Жангиров Т.Р.

Санкт-Петербург

2025

Цель работы.

Изучить принципы объектно-ориентированного программирования. Написать программу на языке C++, которая будет прототипом пошаговой игры с перемещением персонажа и сражением с врагами.

Задание.

На 6/3/1 баллов:

Создать класс игрока, который должен хранить информацию об игроке (его жизни, урон, очки, и т.д. - студент сам определяет необходимые для работы характеристики). Объект класса игрока должен перемещаться по карте. Если у игрока кончаются жизни, то происходит конец игры.

Создать класс врага, который хранит параметры жизней и урона. Объектами класса врага управляет компьютер. При перемещении, если враг пытается перейти на клетку с игроком, то перемещение не происходит, и игроку наносится урон.

Создать класс квадратного/прямоугольного игрового поля, по которому перемещаются игрок и враги. Игровое поле не должно быть меньше 10 на 10 клеток, и не больше 25 на 25 клеток. Размеры поля задаются через конструктор. Рекомендуется для хранения информации об отдельных клетках поля создать отдельный класс.

Реализовать конструкторы перемещения и копирования для поля, а также соответствующие операторы присваивания с копированием и перемещением (должна происходить глубокая копия).

На 8/4/1.5 баллов:

Реализовать непроходимые клетки на поле. При попытке врагов или игрока перейти на такую клетку, перемещение не происходит. Заполнения поля непроходимыми клетками происходит в момент создания поля.

Добавить возможность для игрока переключаться на ближний или дальний бой с изменением значения наносимого урона. Такое переключение требует один ход.

На 10/5/2 баллов:

Добавить класс вражеского здания. Такое здание размещается на карте, и раз в несколько ходов создает нового врага возле себя. Количество ходов до создания нового врага задается в конструкторе.

Реализовать замедляющие клетки на поле. Если игрок переходит на такую клетку, то он он не может двигаться на следующий ход.

Выполнение работы.

Была реализована программа, содержащая все указанные в условие лабораторной работы классы и их поля и методы, а именно:

- Класс игрока с возможностью переключения типа атаки;
- Класс врага;
- Класс игрового поля;
- Непроходимый и замедляющий тип клеток;
- Класс вражеского здания.

Архитектура программы.

В программе реализована иерархия классов, соответствующая принципам ООП.

Основные классы:

- Character – базовый класс персонажа
- Hero – класс героя, наследуемый от Character
- Enemy – класс врага, наследуемый от Character
- EnemyTower – класс вражеской башни
- Cell – класс клетки поля
- Field – класс поля

Дополнительные enum классы:

- CellType – класс типов клеток
- AttackType – класс типов атаки
- Direction – класс направлений движения

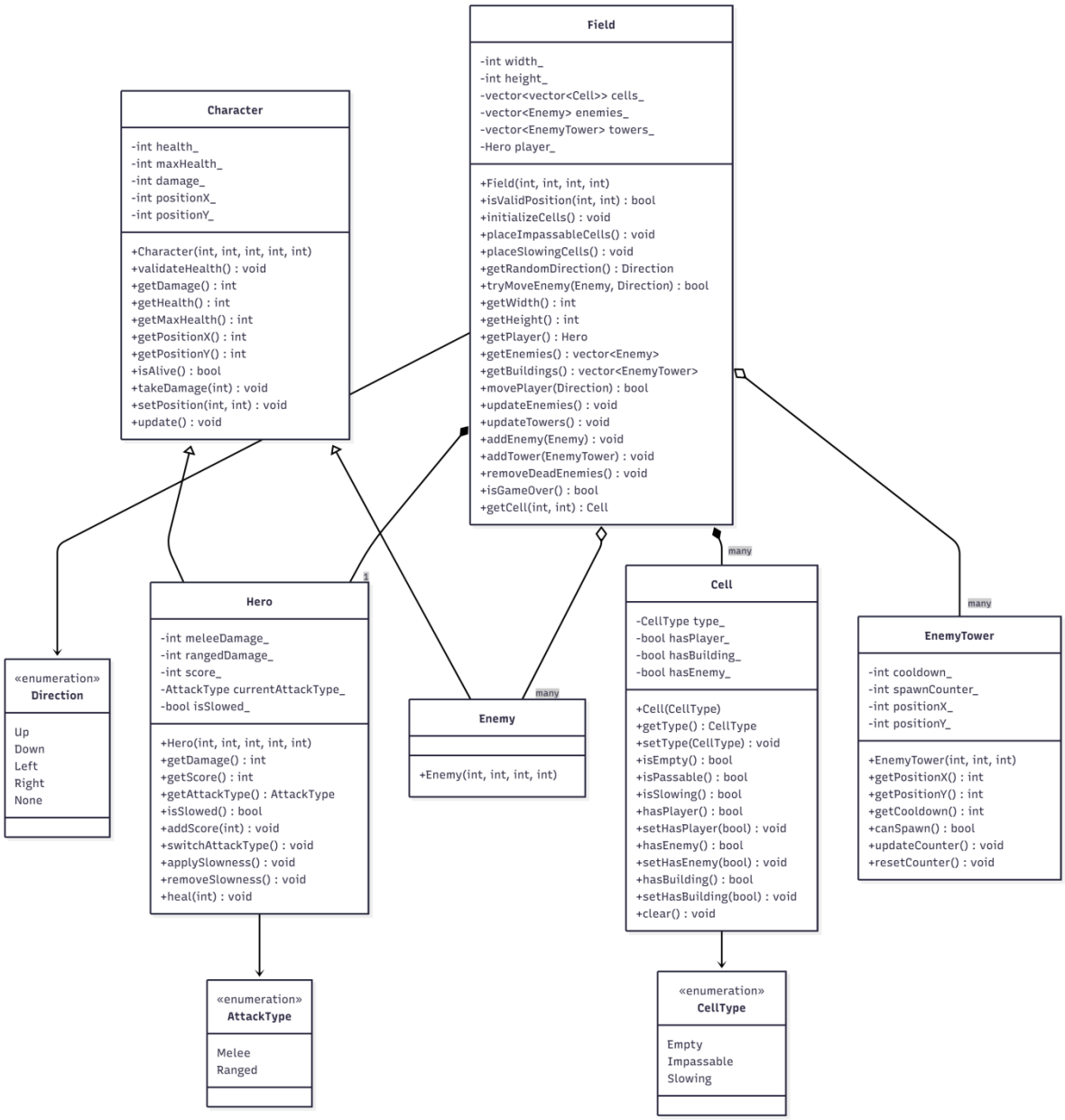

Рисунок 1. UML диаграмма классов реализованной программы

Описание классов.

Основные классы:

- Класс Character

  Класс содержит общие характеристики и методы всех персонажей. Создан для избегания дублирования кода в классах Hero и Enemy.

  Поля класса:

  - health_ - текущее здоровье
  - maxHealth_ - максимальное здоровье

- damage_ - базовый урон
- positionX_, positionY_ - координаты на поле

Методы класса:

- getDamage – виртуальный метод получения значения урона персонажа
- takeDamage – метод получения урона
- setPosition – метод изменения позиции персонажа

- Класс Hero

Класс содержит характеристики и методы героя игры.

Поля класса:

- meleeDamage_ – урон ближнего боя
- rangedDamage_ – урон дальнего боя
- score_ – опыт героя
- currentAttackType_ – текущий тип атаки
- isSlowed_ – состояние замедления

Методы класса:

- addStore – метод получения опыта
- switchAttackType – метод смены типа атаки

  Метод запрашивает текущий тип атаки и меняет его на новый.
- applySlowness – добавление состояния замедления
- removeSlowness – отмена состояния замедления

- Класс Enemy

Класс содержит характеристики и методы вражеских персонажей.

Класс не содержит полей и методов отличных от родительского класса.

- Класс EnemyTower

Класс содержит характеристики и методы вражеских башен.

Поля класса:

- cooldown_ – период спавна врага
- spawnCounter_ – счётчик ходов с последнего спавна

- positionX_, positionY_ – координаты на поле

Методы класса:

- canSpawn – метод проверки возможности спавна
- updateCounter – увеличение счётчика ходов
- resetCounter – обнуление счётчика ходов

- Класс Cell

Класс содержит характеристики и методы клеток поля.

Поля класса:

- type_ – тип клетки
- hasPlayer_ – наличие героя в клетке
- hasBuilding_ – наличие башни в клетке
- hasEnemy_ – наличие врага в клетке

Методы класса:

- setType – метод установки типа клетки
- setHasPlayer – метод установки наличия героя в клетке
- setHasEnemy – метод установки наличия врага в клетке
- setHasBuilding – метод установки наличия башни в клетке
- clear – метод отчистки клетки

- Класс Field

Класс содержит характеристики и методы игрового поля, а так же методы взаимодействия различных классов с игровым полем.

Поля класса:

- width_ – ширина поля
- height_ – высота поля
- cells_ – массив клеток поля
- enemies_ – массив вражеских персонажей
- towers_ – массив вражеских башен
- player_ – герой

Методы класса:

- movePlayer – метод перемещения игрока

Получает направления движения героя и в случае, если герой не замедлен, передвигает его по направлению, если клетка свободна, иначе, если в клетке находиться враг, атакует его, иначе движение не происходит.

- updateEnemies – метод процесса хода врага

  Передвигает врага в случайном направлении, если клетка свободна, иначе, если в клетке находиться герой, атакует его, иначе движение не происходит.

- updateTowers – метод процесса хода башни

  Проверяет прошло ли необходимое количество ходов с момента предыдущего спавна. Если да, то спавнит нового врага рядом с собой.

- addEnemy – метод добавления на поле врага

  В случае возможности появления, создаёт врага в заданной клетке и добавляет его в массив врагов.

- addTower – метод добавления на поле башни

  В случае возможности появления, создаёт башню в заданной клетке и добавляет его в массив башен.

- removeDeadEnemies – метод удаления мёртвого врага

  Проверяет всех врагов в массиве на «живость» и удаляет мёртвых врагов.

- isGameOver – метод окончания игры

  Проверяет здоровье героя, если оно равно 0, то игра заканчивается.

Дополнительные enum классы:

- Класс CellType

  Значения:

  - Empty – пустая клетка
  - Impassable – непроходимая клетка
  - Slowing – замедляющая клетка

- Класс AttackType

  Значения:

  - Melee – ближний бой
  - Ranged – дальний бой

- Класс Direction

  Значения:

  - Up – движение вверх
  - Down – движение вниз
  - Left – движение влево
  - Right – движение вправо
  - None – отсутствие движения (в случае невозможности)

Разработанный программный код см. в приложении А.

Выводы.

Была изучена парадигма объектно-ориентированного программирования. Была реализована программа на языке C++ содержащая основные классы игры с необходимыми полями и методами.

ПРИЛОЖЕНИЕ А
ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: AttackType.h

```
#ifndef ATTACKTYPE_H
#define ATTACKTYPE_H

enum class AttackType {
    Melee,
    Ranged
};

#endif
```

Название файла: CellType.h

```
#ifndef CELLTYPE_H
#define CELLTYPE_H


enum class CellType {
    Empty,
    Impassable,
    Slowing
};


#endif
```

Название файла: Direction.h

```
#ifndef DIRECTION_H
#define DIRECTION_H


enum class Direction {
    Up,
    Down,
    Left,
    Right,
    None
};


#endif
```

Название файла: Character.h

```
#ifndef CHARACTER_H
#define CHARACTER_H

#include <stdexcept>

class Character {
protected:
    int health_;
    int maxHealth_;
    int damage_;
    int positionX_;
    int positionY_;

    void validateHealth();

public:
    Character(int x, int y, int health, int maxHealth, int damage);
    virtual ~Character() = default;

    Character(const Character& other);
    Character& operator=(const Character& other);
    Character(Character&& other) noexcept;
    Character& operator=(Character&& other) noexcept;

    virtual int getDamage() const;
    int getHealth() const;
    int getMaxHealth() const;
    int getPositionX() const;
    int getPositionY() const;
    bool isAlive() const;
    void takeDamage(int damage);
    void setPosition(int x, int y);
    virtual void update();
};

#endif
```

Название файла: Hero.h

```
#ifndef HERO_H
#define HERO_H

#include "Character.h"
#include "AttackType.h"

class Hero : public Character {
private:
    int meleeDamage_;
    int rangedDamage_;
    int score_;
    AttackType currentAttackType_;
    bool isSlowed_;

public:
    Hero(int startX, int startY, int health = 50, int meleeDamage =
5, int rangedDamage = 3);
    Hero(const Hero& other);
    Hero& operator=(const Hero& other);
    Hero(Hero&& other) noexcept;
    Hero& operator=(Hero&& other) noexcept;

    int getDamage() const override;
    int getScore() const;
    AttackType getAttackType() const;
    bool isSlowed() const;

    void addScore(int points);
    void switchAttackType();
    void applySlowness();
    void removeSlowness();
};
#endif
```

Название файла: Enemy.h

```
#ifndef ENEMY_H
#define ENEMY_H
```

```
#include "Character.h"

class Enemy : public Character {
public:
    Enemy(int x, int y, int health = 10, int damage = 3);
};

#endif
```

Название файла: EnemyTower.h

```
#ifndef ENEMYTOWER_H
#define ENEMYTOWER_H

#include <stdexcept>

class EnemyTower {
private:
    int cooldown_;
    int spawnCounter_;
    int positionX_;
    int positionY_;

public:
    EnemyTower(int x, int y, int cooldown = 5);

    int getPositionX() const;
    int getPositionY() const;
    int getCooldown() const;
    bool canSpawn() const;
    void updateCounter();
    void resetCounter();
};

#endif
```

Название файла: Cell.h

```
#ifndef CELL_H
#define CELL_H

#include "CellType.h"

class Cell {
private:
    CellType type_;
    bool hasPlayer_;
    bool hasBuilding_;
    bool hasEnemy_;

public:
    explicit Cell(CellType type = CellType::Empty);

    CellType getType() const;
    void setType(CellType type);
    bool isEmpty() const;
    bool isPassable() const;
    bool isSlowing() const;
    bool hasPlayer() const;
    void setHasPlayer(bool hasPlayer);
    bool hasEnemy() const;
    void setHasEnemy(bool hasEnemy);
    bool hasBuilding() const;
    void setHasBuilding(bool hasBuilding);
    void clear();
};

#endif
```

Название файла: Field.h

```
#ifndef FIELD_H
#define FIELD_H

#include "Cell.h"
#include "Hero.h"
#include "Enemy.h"
```

```
#include "EnemyTower.h"
#include "Direction.h"
#include <vector>
#include <random>
#include <memory>

class Field {
private:
    int width_;
    int height_;
    std::vector<std::vector<Cell>> cells_;
    std::vector<Enemy> enemies_;
    std::vector<EnemyTower> towers_;
    Hero player_;

    bool isValidPosition(int x, int y) const;
    void initializeCells();
    void placeImpassableCells();
    void placeSlowingCells();
    Direction getRandomDirection() const;
    bool tryMoveEnemy(Enemy& enemy, Direction direction);

public:
    Field(int width, int height, int startX, int startY);
    Field(const Field& other);
    Field(Field&& other) noexcept;
    Field& operator=(const Field& other);
    Field& operator=(Field&& other) noexcept;

    int getWidth() const;
    int getHeight() const;
    const Hero& getPlayer() const;
    const std::vector<Enemy>& getEnemies() const;
    const std::vector<EnemyTower>& getBuildings() const;

    bool movePlayer(Direction direction);
    void updateEnemies();
    void updateTowers();
    void addEnemy(const Enemy& enemy);
```

```
    void addTower(const EnemyTower& tower);
    void removeDeadEnemies();
    bool isGameOver() const;
    const Cell& getCell(int x, int y) const;
};

#endif
```

Название файла: Character.cpp

```
#include "Character.h"

Character::Character(int x, int y, int health, int maxHealth, int
damage)
    :      positionX_(x),      positionY_(y),      health_(health),
maxHealth_(maxHealth), damage_(damage)
{
    validateHealth();
    if (damage_ < 0) {
        throw std::invalid_argument("Damage must be non-negative");
    }
}

void Character::validateHealth() {
    if (health_ < 0) {
        health_ = 0;
    }
    if (maxHealth_ <= 0) {
        throw std::invalid_argument("Max health must be positive");
    }
    if (health_ > maxHealth_) {
        health_ = maxHealth_;
    }
}

Character::Character(const Character& other)
    :     health_(other.health_),     maxHealth_(other.maxHealth_),
damage_(other.damage_),
      positionX_(other.positionX_), positionY_(other.positionY_) {}
```

```
Character& Character::operator=(const Character& other) {
    if (this != &other) {
        health_ = other.health_;
        maxHealth_ = other.maxHealth_;
        damage_ = other.damage_;
        positionX_ = other.positionX_;
        positionY_ = other.positionY_;
    }
    return *this;
}

Character::Character(Character&& other) noexcept
    :     health_(other.health_),     maxHealth_(other.maxHealth_),
damage_(other.damage_),
      positionX_(other.positionX_), positionY_(other.positionY_) {
    other.health_ = 0;
    other.maxHealth_ = 0;
    other.damage_ = 0;
}

Character& Character::operator=(Character&& other) noexcept {
    if (this != &other) {
        health_ = other.health_;
        maxHealth_ = other.maxHealth_;
        damage_ = other.damage_;
        positionX_ = other.positionX_;
        positionY_ = other.positionY_;

        other.health_ = 0;
        other.maxHealth_ = 0;
        other.damage_ = 0;
    }
    return *this;
}

int Character::getDamage() const {
    return damage_;
}
```

```
int Character::getHealth() const {
    return health_;
}

int Character::getMaxHealth() const {
    return maxHealth_;
}

int Character::getPositionX() const {
    return positionX_;
}

int Character::getPositionY() const {
    return positionY_;
}

bool Character::isAlive() const {
    return health_ > 0;
}

void Character::takeDamage(int damage) {
    if (damage > 0) {
        health_ -= damage;
        validateHealth();
    }
}

void Character::setPosition(int x, int y) {
    positionX_ = x;
    positionY_ = y;
}

void Character::update() {}
```

Название файла: Hero.cpp

```
#include "Hero.h"
```

```
Hero::Hero(int startX, int startY, int health, int meleeDamage, int
rangedDamage)
    : Character(startX, startY, health, health, meleeDamage),
      meleeDamage_(meleeDamage),       rangedDamage_(rangedDamage),
score_(0),
      currentAttackType_(AttackType::Melee), isSlowed_(false)
{}

Hero::Hero(const Hero& other)
    :     Character(other),     meleeDamage_(other.meleeDamage_),
rangedDamage_(other.rangedDamage_),
      score_(other.score_),
currentAttackType_(other.currentAttackType_),  isSlowed_(other.isSlowed_)
{}

Hero& Hero::operator=(const Hero& other) {
    if (this != &other) {
        Character::operator=(other);
        meleeDamage_ = other.meleeDamage_;
        rangedDamage_ = other.rangedDamage_;
        score_ = other.score_;
        currentAttackType_ = other.currentAttackType_;
        isSlowed_ = other.isSlowed_;
    }
    return *this;
}

Hero::Hero(Hero&& other) noexcept
    : Character(std::move(other)), meleeDamage_(other.meleeDamage_),
rangedDamage_(other.rangedDamage_),
      score_(other.score_),
currentAttackType_(other.currentAttackType_),  isSlowed_(other.isSlowed_)
{
    other.meleeDamage_ = 0;
    other.rangedDamage_ = 0;
    other.score_ = 0;
}

Hero& Hero::operator=(Hero&& other) noexcept {
```

```
        if (this != &other) {
            Character::operator=(std::move(other));
            meleeDamage_ = other.meleeDamage_;
            rangedDamage_ = other.rangedDamage_;
            score_ = other.score_;
            currentAttackType_ = other.currentAttackType_;
            isSlowed_ = other.isSlowed_;

            other.meleeDamage_ = 0;
            other.rangedDamage_ = 0;
            other.score_ = 0;
        }
        return *this;
    }

    int Hero::getDamage() const {
        return (currentAttackType_ == AttackType::Melee) ?
meleeDamage_ : rangedDamage_;
    }

    int Hero::getScore() const {
        return score_;
    }

    AttackType Hero::getAttackType() const {
        return currentAttackType_;
    }

    bool Hero::isSlowed() const {
        return isSlowed_;
    }

    void Hero::addScore(int points) {
        if (points > 0) {
            score_ += points;
        }
    }

    void Hero::switchAttackType() {
```

```
        currentAttackType_ = (currentAttackType_ == AttackType::Melee) ?
AttackType::Ranged : AttackType::Melee;
    }

    void Hero::applySlowness() {
        isSlowed_ = true;
    }

    void Hero::removeSlowness() {
        isSlowed_ = false;
    }
```

Название файла: Enemy.cpp

```
    #include "Enemy.h"

    Enemy::Enemy(int x, int y, int health, int damage)
        : Character(x, y, health, health, damage)
    {}
```

Название файла: EnemyTower.cpp

```
    #include "EnemyTower.h"

    EnemyTower::EnemyTower(int x, int y, int cooldown)
        :    positionX_(x),    positionY_(y),    cooldown_(cooldown),
spawnCounter_(0)
    {
        if (cooldown_ <= 0) {
            throw std::invalid_argument("Cooldown must be positive");
        }
    }

    int EnemyTower::getPositionX() const {
        return positionX_;
    }

    int EnemyTower::getPositionY() const {
        return positionY_;
```

```
}

int EnemyTower::getCooldown() const {
    return cooldown_;
}

bool EnemyTower::canSpawn() const {
    return spawnCounter_ >= cooldown_;
}

void EnemyTower::updateCounter() {
    spawnCounter_++;
}

void EnemyTower::resetCounter() {
    spawnCounter_ = 0;
}
```

Название файла: Cell.cpp

```
#include "Cell.h"

Cell::Cell(CellType type)
    :    type_(type),    hasPlayer_(false),    hasBuilding_(false),
hasEnemy_(false) {}

CellType Cell::getType() const {
    return type_;
}

void Cell::setType(CellType type) {
    type_ = type;
}

bool Cell::isEmpty() const {
    return !hasPlayer_ && !hasEnemy_ && !hasBuilding_;
}

bool Cell::isPassable() const {
```

```
    return type_ != CellType::Impassable;
}

bool Cell::isSlowing() const {
    return type_ == CellType::Slowing;
}

bool Cell::hasPlayer() const {
    return hasPlayer_;
}

void Cell::setHasPlayer(bool hasPlayer) {
    hasPlayer_ = hasPlayer;
}

bool Cell::hasEnemy() const {
    return hasEnemy_;
}

void Cell::setHasEnemy(bool hasEnemy) {
    hasEnemy_ = hasEnemy;
}

bool Cell::hasBuilding() const {
    return hasBuilding_;
}

void Cell::setHasBuilding(bool hasBuilding) {
    hasBuilding_ = hasBuilding;
}

void Cell::clear() {
    hasPlayer_ = false;
    hasEnemy_ = false;
    hasBuilding_ = false;
}
```

Название файла: Field.cpp

```
#include "Field.h"
#include <random>

Field::Field(int width, int height, int startX, int startY)
    : width_(width), height_(height), player_(startX, startY)
{
    if (width_ < 10 || width_ > 25 || height_ < 10 || height_ > 25)
{
        throw std::invalid_argument("Field size must be between
10x10 and 25x25");
    }

    if (!isValidPosition(startX, startY)) {
        throw    std::invalid_argument("Invalid    player    start
position");
    }

    initializeCells();
    placeImpassableCells();
    placeSlowingCells();

    cells_[startY][startX].setHasPlayer(true);
}

bool Field::isValidPosition(int x, int y) const {
    return x >= 0 && x < width_ && y >= 0 && y < height_;
}

void Field::initializeCells() {
    cells_.resize(height_);
    for (int y = 0; y < height_; y++) {
        cells_[y].resize(width_);
        for (int x = 0; x < width_; x++) {
            cells_[y][x] = Cell(CellType::Empty);
        }
    }
}

void Field::placeImpassableCells() {
```

```
        std::random_device rd;
        std::mt19937 gen(rd());
        std::uniform_int_distribution<> dis(0, width_ * height_ - 1);

        int impassableCount = (width_ * height_) / 10;

        for (int i = 0; i < impassableCount; i++) {
            int pos = dis(gen);
            int x = pos % width_;
            int y = pos / width_;

            if (isValidPosition(x, y) && cells_[y][x].getType() ==
CellType::Empty) {
                cells_[y][x].setType(CellType::Impassable);
            }
        }
    }

    void Field::placeSlowingCells() {
        std::random_device rd;
        std::mt19937 gen(rd());
        std::uniform_int_distribution<> dis(0, width_ * height_ - 1);

        int slowingCount = (width_ * height_) / 15;

        for (int i = 0; i < slowingCount; i++) {
            int pos = dis(gen);
            int x = pos % width_;
            int y = pos / width_;

            if (isValidPosition(x, y) && cells_[y][x].getType() ==
CellType::Empty) {
                cells_[y][x].setType(CellType::Slowing);
            }
        }
    }

    Direction Field::getRandomDirection() const {
        std::random_device rd;
```

```
    std::mt19937 gen(rd());
    std::uniform_int_distribution<> dis(0, 3);

    switch (dis(gen)) {
        case 0: return Direction::Up;
        case 1: return Direction::Down;
        case 2: return Direction::Left;
        case 3: return Direction::Right;
        default: return Direction::Up;
    }
}

bool Field::tryMoveEnemy(Enemy& enemy, Direction direction) {
    int newX = enemy.getPositionX();
    int newY = enemy.getPositionY();

    switch (direction) {
        case Direction::Up:
            newY--;
            break;
        case Direction::Down:
            newY++;
            break;
        case Direction::Left:
            newX--;
            break;
        case Direction::Right:
            newX++;
            break;
        case Direction::None:
            return false;
    }

    if (!isValidPosition(newX, newY)) {
        return false;
    }

    Cell& targetCell = cells_[newY][newX];
```

```
        if   (!targetCell.isPassable()   ||   targetCell.hasEnemy()   ||
targetCell.hasBuilding()) {
            return false;
        }

        if (targetCell.hasPlayer()) {
            player_.takeDamage(enemy.getDamage());
            return false;
        }


cells_[enemy.getPositionY()][enemy.getPositionX()].setHasEnemy(false);
        enemy.setPosition(newX, newY);
        targetCell.setHasEnemy(true);

        return true;
    }

    Field::Field(const Field& other)
        : width_(other.width_), height_(other.height_),
          cells_(other.cells_), player_(other.player_),
          enemies_(other.enemies_), towers_(other.towers_) {}

    Field::Field(Field&& other) noexcept
        : width_(other.width_), height_(other.height_),
          cells_(std::move(other.cells_)),
player_(std::move(other.player_)),
          enemies_(std::move(other.enemies_)),
towers_(std::move(other.towers_)) {
        other.width_ = 0;
        other.height_ = 0;
    }

    Field& Field::operator=(const Field& other) {
        if (this != &other) {
            width_ = other.width_;
            height_ = other.height_;
            cells_ = other.cells_;
            player_ = other.player_;
```

```
        enemies_ = other.enemies_;
        towers_ = other.towers_;
    }
    return *this;
}

Field& Field::operator=(Field&& other) noexcept {
    if (this != &other) {
        width_ = other.width_;
        height_ = other.height_;
        cells_ = std::move(other.cells_);
        player_ = std::move(other.player_);
        enemies_ = std::move(other.enemies_);
        towers_ = std::move(other.towers_);

        other.width_ = 0;
        other.height_ = 0;
    }
    return *this;
}

int Field::getWidth() const {
    return width_;
}

int Field::getHeight() const {
    return height_;
}

const Hero& Field::getPlayer() const {
    return player_;
}

const std::vector<Enemy>& Field::getEnemies() const {
    return enemies_;
}

const std::vector<EnemyTower>& Field::getBuildings() const {
    return towers_;
```

```
    }

    bool Field::movePlayer(Direction direction) {
        if (player_.isSlowed()) {
            player_.removeSlowness();
            return false;
        }

        int newX = player_.getPositionX();
        int newY = player_.getPositionY();

        switch (direction) {
            case Direction::Up:
                newY--;
                break;
            case Direction::Down:
                newY++;
                break;
            case Direction::Left:
                newX--;
                break;
            case Direction::Right:
                newX++;
                break;
            case Direction::None:
                return false;
        }

        if (!isValidPosition(newX, newY)) {
            return false;
        }

        Cell& targetCell = cells_[newY][newX];

        if (!targetCell.isPassable() || targetCell.hasEnemy() ||
targetCell.hasBuilding()) {
            return false;
        }
```

```
cells_[player_.getPositionY()][player_.getPositionX()].setHasPlayer(false
);
        player_.setPosition(newX, newY);
        targetCell.setHasPlayer(true);

        if (targetCell.isSlowing()) {
            player_.applySlowness();
        }

        return true;
    }

    void Field::updateEnemies() {
        for (auto& enemy : enemies_) {
            if (enemy.isAlive()) {
                tryMoveEnemy(enemy, getRandomDirection());
            }
        }
    }

    void Field::updateTowers() {
        for (auto& tower : towers_) {
            tower.updateCounter();

            if (tower.canSpawn()) {
                int x = tower.getPositionX();
                int y = tower.getPositionY();

                std::vector<std::pair<int, int>> possiblePositions = {
                    {x-1, y}, {x+1, y}, {x, y-1}, {x, y+1}
                };

                for (const auto& pos : possiblePositions) {
                    if (isValidPosition(pos.first, pos.second)) {
                        Cell& cell = cells_[pos.second][pos.first];
                        if (cell.isPassable() && cell.isEmpty()) {
                            Enemy newEnemy(pos.first, pos.second);
                            addEnemy(newEnemy);
```

```
                            tower.resetCounter();
                            break;
                        }
                    }
                }
            }
        }

    void Field::addEnemy(const Enemy& enemy) {
        int x = enemy.getPositionX();
        int y = enemy.getPositionY();

        if (!isValidPosition(x, y)) {
            throw std::invalid_argument("Invalid enemy position");
        }

        Cell& cell = cells_[y][x];
        if (!cell.isPassable() || !cell.isEmpty()) {
            throw  std::invalid_argument("Cannot  place  enemy  on  this
cell");
        }

        enemies_.push_back(enemy);
        cell.setHasEnemy(true);
    }

    void Field::addTower(const EnemyTower& tower) {
        int x = tower.getPositionX();
        int y = tower.getPositionY();

        if (!isValidPosition(x, y)) {
            throw std::invalid_argument("Invalid tower position");
        }

        Cell& cell = cells_[y][x];
        if (!cell.isPassable() || !cell.isEmpty()) {
            throw  std::invalid_argument("Cannot  place  tower  on  this
cell");
```

```
    }

    towers_.push_back(tower);
    cell.setHasBuilding(true);
}

void Field::removeDeadEnemies() {
    for (auto it = enemies_.begin(); it != enemies_.end(); ) {
        if (!it->isAlive()) {
            int x = it->getPositionX();
            int y = it->getPositionY();
            cells_[y][x].setHasEnemy(false);
            it = enemies_.erase(it);
        } else {
            ++it;
        }
    }
}

bool Field::isGameOver() const {
    return !player_.isAlive();
}

const Cell& Field::getCell(int x, int y) const {
    if (!isValidPosition(x, y)) {
        throw std::invalid_argument("Invalid cell coordinates");
    }
    return cells_[y][x];
}
```