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: Реализация игры на языке C++

Студент гр. 4383 Шлеин Г. А.

Преподаватель Жангиров Т. Р.

Санкт-Петербург

2025

ЦЕЛЬ РАБОТЫ

Разработать систему заклинаний, а также реализовать различные типы заклинаний. Дополнительно создать класс вражеской башни, которая контратакует игрока ослабленными заклинаниями при нахождении в радиусе поражения сооружения. Также необходимо создать ловушки, которые должны наносить урон при переходе игрока/врага на соответствующую клетку. В системе должны соблюдаться: ограниченность ресурсов заклинаний и пошаговое управление, где каждое использование заклинания требует траты хода игрока.

ЗАДАНИЕ

1. Создать интерфейс карточки заклинания. Заклинание должно применяться игроком. На использование заклинания игрок тратит один ход.
2. Создать класс “руки” игрока, которая содержит все карточки заклинаний, которые игрок может применить в свой ход. Изначально рука игрока содержит только одно случайное заклинание. Реализовать возможность получать новые заклинание игроком, например, тратить очки на покупку или после уничтожения определенного кол-ва врагов. Размер “руки” должен быть ограничен и задается через конструктор.
3. Реализовать интерфейс заклинанием прямого урона. Это заклинание при использовании должно наносить урон врагу или вражескому зданию, если они находятся в достижимом радиусе. Если в качестве цели не выбран враг или вражеское здание, то заклинание не используется.
4. Реализовать интерфейс заклинания урона по площади. Это заклинание при использовании в допустимом радиусе наносит урон по области 2 на 2 клетки. Заклинание используется, даже если там нет никого.
5. Реализовать интерфейс заклинания ловушки. Заклинание размещает на поле ловушку, если враг наступает на клетку с ловушкой, то ему наносится урон, и ловушка пропадает.
6. Создать класс вражеской башни. Вражеская башня размещается на

поле, и если в радиусе ее атаки появляется игрок, то применяет ослабленную версию заклинания прямого урона. Не может применять заклинание несколько ходов подряд.

ВЫПОЛНЕНИЕ РАБОТЫ

Для демонстрации зависимостей между классами была построена UML-диаграмма. (см. рис. 1)

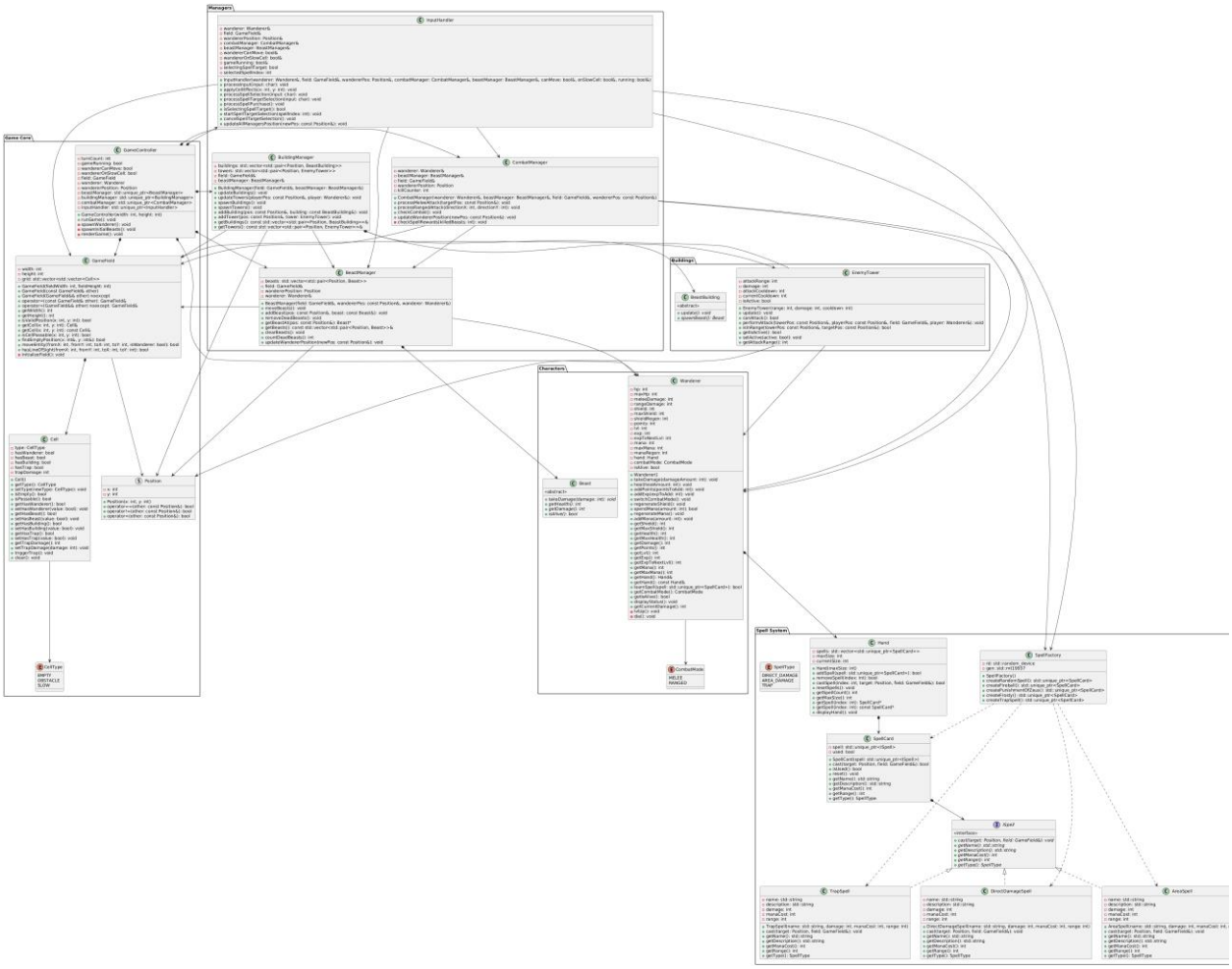

Рис. 1 – UML-диаграмма классов

Архитектура системы с заклинаниями

1. Общая структура и распределение ответственностей

Система построена на объектно-ориентированных принципах, каждый класс выполняет свою определенную задачу. Класс GameController является главным координатором игрового процесса, управляющим жизненным циклом всех сущностей и обработкой взаимодействий между ними.

InputHandler используется для обработки ввода пользователя, который трансформирует команды игрока в игровые действия, также в этом классе описываются методы для применения эффектов клеток и механика выбора заклинаний.

Wanderer содержит информацию о состоянии игрока, включая систему маны и заклинаний. Hand управляет набором заклинаний с контролем размера "руки". Классы BeastManager и BuildingManager, отвечают за отдел игроков, а именно за перемещение сущностей, атаку персонажа и спавн врагов. CombatManager обрабатывает все виды боевых взаимодействий, включая тип атаки (MELEE/RANGED). EnemyTower обрабатывает случаи, когда игрок находится в области поражения вражеской башни, применяя ослабленное заклинание прямого урона.

2. Иерархия заклинаний

Система заклинаний строится в виде общего интерфейса ISpell с разными реализациями. Выбор интерфейса позволяет единым образом работать с любыми заклинаниями через указатели на класс ISpell.

Spell, декоратор для заклинаний, управляющий состоянием (“использовано”) включает std::unique_ptr для управления временем жизни заклинаний, обеспечивая безопасное владение ресурсами, не нужно постоянно вручную освобождать память. Реализован класс DirectDamageSpell, здесь описывается заклинание прямого урона по одной цели. Класс AreaSpell содержит логику обработки площади поражения заклинаниями, с помощью изменений в этом классе можно менять радиус и форму области. Установка ловушек на карте реализуется как изменение состояния поля, а не как мгновенный эффект.

Случайный выбор начального заклинания и дальнейший выбор для покупки реализованы в SpellFactory.

3. Система "руки" и управление ресурсами

Hand реализует механику ограниченного набора заклинаний. Начальный размер руки (1 заклинание) задается через конструктор и

проверяется при добавлении новых карт. Метод addSpell() проверяет доступное место перед добавлением. Метод resetSpells() сбрасывает состояние “использовано” каждый ход. Два способа пополнения: автоматически после убийства 3 врагов или за 50 очков через покупку. Покупка производится по нажатии клавиши “B”.

Система маны инкапсулирована в Wanderer с контролем максимального значения, что обеспечивает защиту от переполнения, и автоматическим восстановлением. Мана автоматические восстанавливается, каждые 3 хода значение увеличивается на 10. Метод spendMana() проверяет достаточность перед применением заклинаний.

4. Координация сложных взаимодействий

CombatManager обрабатывает боевые взаимодействия:

- Обрабатывает атаки игрока, выбор ближнего или дальнего боя.
- Управляет наградами за бой, начисляет очки и опыт за убитых врагов, а также выдает новые заклинания за каждый 3 убийства.
- Взаимодействует с системой заклинаний через SpellFactory.

5. Расширяемость системы заклинаний

Архитектура позволяет легко добавлять новые типы заклинаний. Для интеграции нового типа достаточно:

- Создать класс, наследованный от ISpell.
- Реализовать все виртуальные методы.
- Добавить создание в SpellFactory::createRandomSpell()

6. Безопасность и управление состоянием

Система использует строгую валидацию:

- Проверка индексов заклинаний в Hand::getSpell() в пределах размера руки обеспечивает выход за границы.
- Контроль маны в Wanderer::spendMana() предотвращает отрицательные значения.
- Проверка радиуса в InputHandler::processSpellTargetSelection()

7. Интеграция с существующей архитектурой

Wanderer содержит Hand как компонент, обеспечивая инкапсуляцию системы заклинаний.

GameController инициализирует все системы в правильном порядке и управляет основным игровым циклом.

InputHandler интегрирован с системой заклинаний через:

- Обработку цифровых клавиш для выбора заклинаний.
- Механизм выбора целей для кастования.
- Обработку покупки новых заклинаний.

Игровой цикл дополнен этапами:

- Обработка башен (BuildingManager::updateTowers()).
- Проверка ловушек (в BeastManager::moveBeasts() и InputHandler).
- Сброс состояния заклинаний (Hand::resetSpells()).

Архитектура демонстрирует сбалансированный подход, сохраняя низкую связанность между компонентами при обеспечении их эффективного взаимодействия.

Пайплайн данных с системой заклинаний

1. Инициализация

- Создание игрового поля со случайными препятствиями и медленными клетками.
- Размещение игрока в позиции (0, 0) с начальной "рукой" из одного случайного заклинания.
- Генерация 1-2 врагов и 1 здания в случайных свободных позициях.
- Размещение 2 вражеских башен (символ ‘X’) в случайных свободных клетках карты.

2. Игровой цикл

Ввод команды → Обработка хода игрока → Атаки башен → Движение врагов → Проверка ловушек → Обновление сооружений → Спавн врагов →

Восстановление маны → Вывод состояния

3. Обработка хода

Игрок:

- Движение по WASD с проверкой проходимости и занятости клеток.
- Переключение режима боя (C) - ближний/дальний.
- Применение заклинаний (1-3) с выбором индекса и целевой позиции.
- Покупка заклинаний (B) за 50 очков при неполной руке.
- Автоматическая атака врагов в радиусе при движении.

Заклинания:

- Прямой урон: проверка наличия врага/здания в радиусе 3 клеток.
- Урон по площади: нанесение урона по области 2x2 клетки.
- Ловушки: размещение на свободных клетках, срабатывание при попадании врага на клетку.

Враги: случайное движение к смежным клеткам с проверкой ловушек.

Башни: атака игрока ослабленным заклинанием при нахождении в радиусе 4 клеток.

Здания: отсчет времени до спавна новых врагов в соседних клетках.

4. Особенности логики с заклинаниями

- Клетки типа SLOW пропускают ход игрока.
- Бой происходит при переходе на клетку с врагом или соседстве с ним.
- Система маны: восстановление 10 единиц за ход, трата на заклинания.
- Система "руки": ограниченный размер (по умолчанию 3), пополнение за очки или убийства.
- Ловушки сохраняются между ходами и срабатывают однократно.
- Башни не могут атаковать несколько ходов подряд (перезарядка

3 хода).

5. Ввод-вывод

- Ввод: только в main() через std::cin (WASD - движение, C - режим, M – показать заклинания, 1-3 – применить заклинание, B - покупка, I – показать статус игрока, Q - выход).
- Вывод: отображение здоровья, маны, режима боя, счета, доступных заклинаний, активных ловушек.
- Игровая логика изолирована от ввода-вывода для чистоты архитектуры.

Программа завершается при смерти игрока или команде выхода, выводя финальный счет.

Проверка работоспособности

Программа прошла все этапы проверки и демонстрирует стабильную работу. Все классы функционируют в соответствии с проектной спецификацией. Игровой цикл выполняется корректно, система реагирует на пользовательский ввод, обрабатывает взаимодействия между объектами.

Рисунок 1 - Успешная компиляция программы

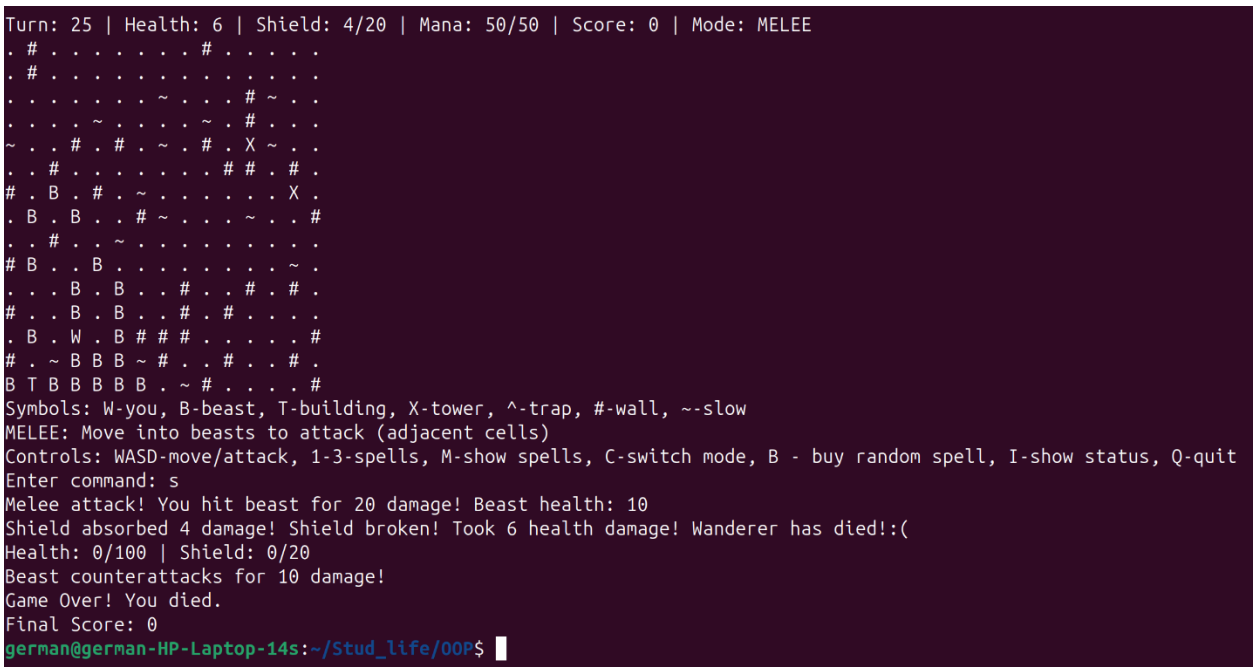

Рисунок 2 - Завершение программы

ВЫВОД

В ходе лабораторной работы была успешно разработана система заклинаний. Основным достижением стало создание механизма магических карточек, добавленного в игровую механику. Реализована иерархия классов заклинаний с использованием полиморфизма, обеспечивающая гибкость и расширяемость системы.

Созданы три типа заклинаний: прямого урона для точечных атак, урона по площади для атаки по области размером 2x2 и тактических ловушек для контроля территории. Разработана система "руки" игрока, ограничивающая доступные заклинания.

Архитектурно система построена с соблюдением принципов ООП, обеспечив четкое разделение ответственностей между классами. Реализованная архитектура позволяет легко добавлять новые типы заклинаний без изменения основной логики.

ПРИЛОЖЕНИЕ А
ИСХОДНЫЙ КОД ПРОГРАММЫ

Файл Spell.h:

```
#ifndef SPELL_H
#define SPELL_H

#include "GameField.h"
#include <string>
#include <memory>

enum class SpellType {
    DIRECT_DAMAGE,
    AREA_DAMAGE,
    TRAP
};

class ISpell {
public:
    virtual ~ISpell() = default;
    virtual void cast(Position target, GameField& field) = 0;
    virtual std::string getName() const = 0;
    virtual std::string getDescription() const = 0;
    virtual int getManaCost() const = 0;
    virtual int getRange() const = 0;
    virtual SpellType getType() const = 0;
};

class SpellCard {
private:
    std::unique_ptr<ISpell> spell;
    bool used;

public:
    SpellCard(std::unique_ptr<ISpell> spell);
    ~SpellCard() = default;

    SpellCard(const SpellCard&) = delete;
    SpellCard& operator=(const SpellCard&) = delete;
    SpellCard(SpellCard&&) = default;
    SpellCard& operator=(SpellCard&&) = default;

    bool cast(Position target, GameField& field);
    bool isUsed() const;
    void reset();

    std::string getName() const;
    std::string getDescription() const;
    int getManaCost() const;
    int getRange() const;
    SpellType getType() const;
};

#endif
```

Файл Spell.cpp:

```
#include "Spell.h"
#include <iostream>

SpellCard::SpellCard(std::unique_ptr<ISpell>          spell)          :
spell(std::move(spell)), used(false) {}

bool SpellCard::cast(Position target, GameField& field) {
    if (used) {
        std::cout << "Spell already used this turn!\n";
        return false;
    }

    spell->cast(target, field);
    used = true;
    return true;
}

bool SpellCard::isUsed() const {
    return used;
}

void SpellCard::reset() {
    used = false;
}

std::string SpellCard::getName() const {
    return spell->getName();
}

std::string SpellCard::getDescription() const {
    return spell->getDescription();
}

int SpellCard::getManaCost() const {
    return spell->getManaCost();
}

int SpellCard::getRange() const {
    return spell->getRange();
}

SpellType SpellCard::getType() const {
    return spell->getType();
}
```

Файл AreaSpell.cpp:

```
#include "AreaSpell.h"
#include <iostream>

AreaSpell::AreaSpell(std::string name, int damage, int manaCost, int
range)
    : name(name), damage(damage), manaCost(manaCost), range(range) {
    description = "Deals " + std::to_string(damage) + " damage in 2x2
```

```
area";
}

void AreaSpell::cast(Position target, GameField& field) {
    int hitCount = 0;

    for (int dx = 0; dx < 2; ++dx) {
        for (int dy = 0; dy < 2; ++dy) {
            int x = target.x + dx;
            int y = target.y + dy;

            if (field.isValidPosition(x, y)) {
                Cell& cell = field.getCell(x, y);

                if (cell.getHasBeast()) {
                    std::cout << name << " hits beast at (" << x << ","
<< y << ") for " << damage << " damage!\n";
                    hitCount++;
                }

                if (cell.getHasBuilding()) {
                    std::cout << name << " hits building at (" << x <<
"," << y << ") for " << damage << " damage!\n";
                    hitCount++;
                }
            }
        }
    }

    if (hitCount == 0) {
        std::cout << name << " explodes but hits nothing!\n";
    } else {
        std::cout << name << " hit " << hitCount << " targets!\n";
    }
}
```

Файл AreaSpell.h:

```
#ifndef AREA_SPELL_H
#define AREA_SPELL_H

#include "Spell.h"

class AreaSpell : public ISpell {
private:
    std::string name;
    std::string description;
    int damage;
    int manaCost;
    int range;

public:
    virtual ~AreaSpell() = default;
    AreaSpell(std::string name, int damage, int manaCost = 3, int range
= 2);

    void cast(Position target, GameField& field) override;
```

```
    std::string getName() const override { return name; }
    std::string getDescription() const override { return description; }
    int getManaCost() const override { return manaCost; }
    int getRange() const override { return range; }
    SpellType getType() const override { return SpellType::AREA_DAMAGE;
}
};

#endif
```

Файл CombatManager.cpp:

```
#include "CombatManager.h"
#include <iostream>
#include <algorithm>

CombatManager::CombatManager(Wanderer&     wanderer,     BeastManager&
beastManager,
                             GameField&    field,    const    Position&
wandererPos)
    : wanderer(wanderer), beastManager(beastManager),
      field(field), wandererPosition(wandererPos), killCounter(0) {}

void CombatManager::processMeleeAttack(const Position& targetPos) {
    Beast* beast = beastManager.getBeastAt(targetPos);
    if (beast) {
        int damage = wanderer.getCurrentDamage();
        beast->takeDamage(damage);
        std::cout << "Melee attack! You hit beast for " << damage << "
damage! ";

        if (!beast->isAlive()) {
            std::cout << "Beast defeated!\n";
            field.getCell(targetPos.x, targetPos.y).setHasBeast(false);
        } else {
            std::cout << "Beast health: " << beast->getHealth() << "\n";
        }

        if (wanderer.getIsAlive()) {
            wanderer.takeDamage(beast->getDamage());
            std::cout  <<  "Beast  counterattacks  for  "  <<  beast-
>getDamage() << " damage!\n";
        }
    }
}

void CombatManager::processRangedAttack(int directionX, int directionY)
{
    bool attacked = false;

    const auto& beasts = beastManager.getBeasts();

    for (const auto& beastData : beasts) {
        int beastX = beastData.first.x;
        int beastY = beastData.first.y;
```

```
        int vecX = beastX - wandererPosition.x;
        int vecY = beastY - wandererPosition.y;

        int distance = std::max(std::abs(vecX), std::abs(vecY));

        if (distance >= 2 && distance <= 3) {
            if ((directionX == 0 || (vecX * directionX > 0)) &&
                (directionY == 0 || (vecY * directionY > 0))) {

                if           (!field.hasLineOfSight(wandererPosition.x,
wandererPosition.y, beastX, beastY)) {
                    continue;
                }

                int damage = wanderer.getCurrentDamage();
                Beast* beast = beastManager.getBeastAt(Position(beastX,
beastY));
                if (beast) {
                    beast->takeDamage(damage);
                    std::cout << "Ranged attack! You shoot beast for "
<< damage << " damage! ";

                    if (!beast->isAlive()) {
                        std::cout << "Beast defeated!\n";
                        field.getCell(beastX,
beastY).setHasBeast(false);
                    } else {
                        std::cout  <<  "Beast  health:  "  <<  beast-
>getHealth() << "\n";
                    }
                    attacked = true;
                    break;
                }
            }
        }
    }

    if (!attacked) {
        std::cout << "Ranged: no beasts in that direction (2-3 cells
away)\n";
    }
}

void CombatManager::checkCombat() {
    int killedBeasts = beastManager.countDeadBeasts();

    if (killedBeasts > 0) {
        int points = killedBeasts * 50;
        wanderer.addPoints(points);
        int exp = killedBeasts * 50;
        wanderer.addExp(exp);
        std::cout << "Defeated " << killedBeasts << " beasts! +" <<
points << " points!\n";
        std::cout << "Experience gained: " << exp << "\n";
```

```
         if (killCounter >= 3) {
            killCounter = 0;
            SpellFactory factory;
            auto newSpell = factory.createRandomSpell();
            if   (newSpell   &&   wanderer.getHand().getSpellCount()   <
wanderer.getHand().getMaxSize()) {
                wanderer.learnSpell(std::move(newSpell));
                std::cout << "Gained new spell for 3 kills!\n";
            }
        }
        checkSpellRewards(killedBeasts);

        beastManager.removeDeadBeasts();
    }
}

void CombatManager::checkSpellRewards(int killedBeasts) {
    killCounter += killedBeasts;
    if (killCounter >= 3) {
        killCounter = 0;

        SpellFactory factory;
        auto newSpell = factory.createRandomSpell();
        if (newSpell) {
            if           (wanderer.getHand().getSpellCount()          <
wanderer.getHand().getMaxSize()) {
                if (wanderer.learnSpell(std::move(newSpell))) {
                    std::cout << "Reward: New spell earned for defeating
3 beasts!\n";
                }
            } else {
                std::cout << "Hand is full! Can't add new spell.\n";
            }
        }
    }
}
```

Файл CombatManager.h:

```
#ifndef COMBAT_MANAGER_H
#define COMBAT_MANAGER_H

#include "GameField.h"
#include "Wanderer.h"
#include "BeastManager.h"
#include "SpellFactory.h"

class CombatManager {
private:
    Wanderer& wanderer;
    BeastManager& beastManager;
    GameField& field;
    Position wandererPosition;
    int killCounter;

public:
    CombatManager(Wanderer&   wanderer,   BeastManager&   beastManager,
GameField& field, const Position& wandererPos);
```

```
    void processMeleeAttack(const Position& targetPos);
    void processRangedAttack(int directionX, int directionY);
    void checkCombat();
    void updateWandererPosition(const Position& newPos) {
        wandererPosition = newPos;
    }
private:
    void checkSpellRewards(int killedBeasts);
};

#endif
```

Файл DirectDamageSpell.cpp:

```
#include "DirectDamageSpell.h"
#include <iostream>

DirectDamageSpell::DirectDamageSpell(std::string name, int damage, int
manaCost, int range)
    : name(name), damage(damage), manaCost(manaCost), range(range) {
    description = "Deals " + std::to_string(damage) + " damage to single
target";
}

void DirectDamageSpell::cast(Position target, GameField& field) {
    if (!field.isValidPosition(target.x, target.y)) {
        std::cout << "Invalid target position!\n";
        return;
    }

    Cell& cell = field.getCell(target.x, target.y);
    bool hitSomething = false;

    if (cell.getHasBeast()) {
        std::cout << name << " hits beast for " << damage << " damage!\n";
        hitSomething = true;
    }

    if (cell.getHasBuilding()) {
        std::cout << name << " hits building for " << damage << "
damage!\n";
        hitSomething = true;
    }

    if (!hitSomething) {
        std::cout << name << " fizzles no valid target!\n";
    }
}
```

Файл DirectDamageSpell.h:

```
#ifndef DIRECT_DAMAGE_SPELL_H
#define DIRECT_DAMAGE_SPELL_H

#include "Spell.h"
#include "BeastManager.h"
#include "BuildingManager.h"
```

```
class DirectDamageSpell : public ISpell {
private:
    std::string name;
    std::string description;
    int damage;
    int manaCost;
    int range;

public:
    DirectDamageSpell(std::string name, int damage, int manaCost = 2,
int range = 3);

    void cast(Position target, GameField& field) override;
    std::string getName() const override { return name; }
    std::string getDescription() const override { return description; }
    int getManaCost() const override { return manaCost; }
    int getRange() const override { return range; }
    SpellType     getType()     const     override     {     return
SpellType::DIRECT_DAMAGE; }
};

#endif
```

Файл EnemyTower.cpp:

```
#include "EnemyTower.h"
#include "Wanderer.h"
#include <iostream>
#include <cmath>

EnemyTower::EnemyTower(int range, int damage, int cooldown)
    : attackRange(range), damage(damage), attackCooldown(cooldown),
      currentCooldown(0), isActive(true) {}

void EnemyTower::update() {
    if (currentCooldown > 0) {
        currentCooldown--;
    }
}

bool EnemyTower::canAttack() const {
    return isActive && currentCooldown == 0;
}

bool  EnemyTower::isInRange(const  Position&  towerPos,  const  Position&
targetPos) const {
    int distance = std::max(std::abs(targetPos.x - towerPos.x),
                            std::abs(targetPos.y - towerPos.y));
    return distance <= attackRange;
}

void EnemyTower::performAttack(const Position& towerPos, const Position&
playerPos,
                               GameField& field, Wanderer& player) {
    if (!canAttack() || !isInRange(towerPos, playerPos)) {
```

```
        return;
    }

    if (!field.hasLineOfSight(towerPos.x, towerPos.y, playerPos.x,
playerPos.y)) {
        return;
    }

    player.takeDamage(damage);
    std::cout << "Enemy tower attacks you for " << damage << "
damage!\n";

    currentCooldown = attackCooldown;
}

bool EnemyTower::getIsActive() const {
    return isActive;
}

void EnemyTower::setActive(bool active) {
    isActive = active;
}

int EnemyTower::getAttackRange() const {
    return attackRange;
}
```

Файл EnemyTower.h:

```
#ifndef ENEMY_TOWER_H
#define ENEMY_TOWER_H

#include "GameField.h"

class EnemyTower {
private:
    int attackRange;
    int damage;
    int attackCooldown;
    int currentCooldown;
    bool isActive;

public:
    EnemyTower(int range = 4, int damage = 15, int cooldown = 3);

    void update();
    bool canAttack() const;
    void performAttack(const Position& towerPos, const Position&
playerPos, GameField& field, class Wanderer& player);
    bool isInRange(const Position& towerPos, const Position& targetPos)
const;

    bool getIsActive() const;
    void setActive(bool active);
    int getAttackRange() const;
```

```
};

#endif
```

Файл GameController.cpp:

```
#include "GameController.h"
#include "SpellFactory.h"
#include <iostream>
#include <random>

GameController::GameController(int width, int height)
    :    turnCount(0),    gameRunning(true),    wandererCanMove(true),
wandererOnSlowCell(false),
      field(width, height), wanderer(), wandererPosition() {

    spawnWanderer();

    beastManager        =        std::make_unique<BeastManager>(field,
wandererPosition, wanderer);
    buildingManager     =     std::make_unique<BuildingManager>(field,
*beastManager);
    combatManager      =      std::make_unique<CombatManager>(wanderer,
*beastManager, field, wandererPosition);
    inputHandler   =   std::make_unique<InputHandler>(wanderer,   field,
wandererPosition,   *combatManager,   *beastManager,   wandererCanMove,
wandererOnSlowCell, gameRunning);

    spawnInitialBeasts();
    buildingManager->spawnBuildings();
    buildingManager->spawnTowers();
    SpellFactory factory;

    auto spell = factory.createRandomSpell();
    if (spell) {
        wanderer.learnSpell(std::move(spell));
    }
}

void GameController::spawnWanderer() {
    int x, y;
    if (field.findEmptyPosition(x, y)) {
        wandererPosition = Position(x, y);
        field.getCell(x, y).setHasWanderer(true);
    }
}

void GameController::spawnInitialBeasts() {
    std::random_device rd;
    std::mt19937 gen(rd());
    std::uniform_int_distribution<> dis(1, 2);

    int beastCount = dis(gen);
    for (int i = 0; i < beastCount; ++i) {
        int x, y;
        if (field.findEmptyPosition(x,y)) {
```

```
            Beast beast(30, 10);
            beastManager->addBeast(Position(x, y), Beast(30, 10));
        }
    }
}

void GameController::renderGame() const {
    std::cout << "\nTurn: " << turnCount << " | Health: " <<
wanderer.getHealth()
              << " | Shield: " << wanderer.getShield() << "/" <<
wanderer.getMaxShield()
              << " | Mana: " << wanderer.getMana() << "/" <<
wanderer.getMaxMana()
              << " | Score: " << wanderer.getPoints()
              << " | Mode: " << (wanderer.getCombatMode() ==
CombatMode::MELEE ? "MELEE" : "RANGED") << "\n";

    for (int y = 0; y < field.getHeight(); ++y) {
        for (int x = 0; x < field.getWidth(); ++x) {
            const Cell& cell = field.getCell(x, y);
            char symbol = '.';

            if (cell.getHasWanderer()) symbol = 'W';
            else if (cell.getHasBeast()) symbol = 'B';
            else if (cell.getHasBuilding()) {
                bool isTower = false;
                for (const auto& tower : buildingManager->getTowers())
{
                    if (tower.first.x == x && tower.first.y == y) {
                        symbol = 'X';
                        isTower = true;
                        break;
                    }
                }
                if (!isTower) {
                    symbol = 'T';
                }
            }
            else if (cell.getType() == CellType::OBSTACLE) symbol = '#';
            else if (cell.getType() == CellType::SLOW) symbol = '~';
            else if (cell.getHasTrap()) symbol = '^';

            std::cout << symbol << ' ';
        }
        std::cout << '\n';
    }

    std::cout << "Symbols: W-you, B-beast, T-building, X-tower, ^-trap,
#-wall, ~-slow\n";
    if (wanderer.getCombatMode() == CombatMode::MELEE) {
        std::cout << "MELEE: Move into beasts to attack (adjacent
cells)\n";
    } else {
        std::cout << "RANGED: Move cursor 2-3 cells away to ranged
attack\n";
    }
    std::cout << "Controls: WASD-move/attack, 1-3-spells, M-show spells,
```

```
C-switch mode, B - buy random spell, I-show status, Q-quit\n";
}

void GameController::runGame() {
    std::cout << "=== Game Started ===\n";

    while (gameRunning && wanderer.getIsAlive()) {
        turnCount++;
        renderGame();

        wanderer.getHand().resetSpells();

        std::cout << "Enter command: ";
        char input;
        std::cin >> input;

        if (!gameRunning || !wanderer.getIsAlive()) {
            break;
        }

        if (input == 'i' || input == 'I') {
            wanderer.displayStatus();
            turnCount--;
            continue;
        }

        if (input == 'm' || input == 'M') {
            wanderer.getHand().displayHand();
            std::cout << "Mana: " << wanderer.getMana() << "/" <<
wanderer.getMaxMana() << "\n";
            turnCount--;
            continue;
        }

        inputHandler->processInput(input);

        if (!wanderer.getIsAlive()) {
            std::cout << "Game Over! You died.\n";
            break;
        }

        if (!gameRunning) {
            break;
        }

        if (wanderer.getIsAlive() && turnCount % 3 == 0) {
            int oldShield = wanderer.getShield();
            wanderer.regenerateShield();
            wanderer.regenerateMana();

            if (wanderer.getShield() > oldShield) {
                std::cout << "Shield regenerated to " <<
wanderer.getShield() << "/" << wanderer.getMaxShield() << "!\n";
            }
```

```
        }


        if (wanderer.getIsAlive()) {
            beastManager->moveBeasts();
            buildingManager->updateBuildings();
            buildingManager->updateTowers(wandererPosition, wanderer);
            combatManager->checkCombat();
        }
    }

    std::cout << "Final Score: " << wanderer.getPoints() << "\n";
}
```

Файл GameController.h:

```
#ifndef GAMECONTROLLER_H
#define GAMECONTROLLER_H

#include "GameField.h"
#include "Wanderer.h"
#include "BeastManager.h"
#include "BuildingManager.h"
#include "CombatManager.h"
#include "InputHandler.h"
#include <memory>

class GameController {
private:
    int turnCount;
    bool gameRunning;
    bool wandererCanMove;
    bool wandererOnSlowCell;

    GameField field;
    Wanderer wanderer;
    Position wandererPosition;

    std::unique_ptr<BeastManager> beastManager;
    std::unique_ptr<BuildingManager> buildingManager;
    std::unique_ptr<CombatManager> combatManager;
    std::unique_ptr<InputHandler> inputHandler;

    void spawnWanderer();
    void spawnInitialBeasts();
    void renderGame() const;

public:
    GameController(int width = 15, int height = 15);
    void runGame();
};

#endif
```

Файл Hand.cpp:

```
#include "Hand.h"
```

```
#include <iostream>

Hand::Hand(int maxSize) : maxSize(maxSize), currentSize(0) {
    spells.reserve(maxSize);
}

bool Hand::addSpell(std::unique_ptr<SpellCard> spell) {
    if (!spell) {
        return false;
    }

    if (currentSize >= maxSize) {
        return false;
    }

    spells.push_back(std::move(spell));
    currentSize++;
    return true;
}

bool Hand::removeSpell(int index) {
    if (index < 0 || index >= currentSize) {
        return false;
    }

    spells.erase(spells.begin() + index);
    currentSize--;
    return true;
}

bool Hand::castSpell(int index, Position target, GameField& field) {
    if (index < 0 || index >= currentSize) {
        std::cout << "Invalid spell index!\n";
        return false;
    }

    return spells[index]->cast(target, field);
}

void Hand::resetSpells() {
    for (auto& spell : spells) {
        spell->reset();
    }
}

int Hand::getSpellCount() const {
    return currentSize;
}

int Hand::getMaxSize() const {
    return maxSize;
}

SpellCard* Hand::getSpell(int index) {
```

```
    if (index < 0 || index >= currentSize) return nullptr;
    return spells[index].get();
}

const SpellCard* Hand::getSpell(int index) const {
    if (index < 0 || index >= currentSize) return nullptr;
    return spells[index].get();
}

void Hand::displayHand() const {
    std::cout << "=== Spell Hand ===\n";
    for (int i = 0; i < currentSize; ++i) {
        std::cout << i + 1 << ". " << spells[i]->getName()
                  << " (Cost: " << spells[i]->getManaCost()
                  << ", Range: " << spells[i]->getRange() << ")\n";
        if (spells[i]->isUsed()) {
            std::cout << " [USED]";
        } else {
            std::cout << " [READY]";
        }

        std::cout << "   " << spells[i]->getDescription() << "\n";
    }
    if (currentSize == 0) {
        std::cout << "Empty\n";
    }
    std::cout << "==================\n";
}
```

Файл Hand.h:

```
#ifndef HAND_H
#define HAND_H

#include "Spell.h"
#include <vector>
#include <memory>

class Hand {
private:
    std::vector<std::unique_ptr<SpellCard>> spells;
    int maxSize;
    int currentSize;

public:
    Hand(int maxSize = 3);
    ~Hand() = default;

    Hand(const Hand&) = delete;
    Hand& operator=(const Hand&) = delete;

    Hand(Hand&&) = default;
    Hand& operator=(Hand&&) = default;

    bool addSpell(std::unique_ptr<SpellCard> spell);
    bool removeSpell(int index);
```

```
    bool castSpell(int index, Position target, GameField& field);
    void resetSpells();

    int getSpellCount() const;
    int getMaxSize() const;
    SpellCard* getSpell(int index);
    const SpellCard* getSpell(int index) const;

    void displayHand() const;
};

#endif
```

Файл InputHandler.cpp:

```
#include "InputHandler.h"
#include <iostream>
#include <cmath>

InputHandler::InputHandler(Wanderer&   wanderer,   GameField&   field,
Position& wandererPos,
                           CombatManager&  combatManager,  BeastManager&
beastManager, bool& canMove, bool& onSlowCell, bool& running)
    : wanderer(wanderer), field(field), wandererPosition(wandererPos),
      combatManager(combatManager),         beastManager(beastManager),
wandererCanMove(canMove),
      wandererOnSlowCell(onSlowCell), gameRunning(running),
      selectingSpellTarget(false), selectedSpellIndex(-1) {}

void InputHandler::updateAllManagersPosition(const Position& newPos) {
    wandererPosition = newPos;
    beastManager.updateWandererPosition(newPos);
    combatManager.updateWandererPosition(newPos);
}

void InputHandler::processInput(char input) {
    if (selectingSpellTarget) {
        processSpellTargetSelection(input);
        return;
    }

    if (!wandererCanMove || !wanderer.getIsAlive()) {
        if (!wanderer.getIsAlive()) {
            gameRunning = false;
            return;
        }
        std::cout << "You are slowed and cannot move this turn!\n";
        wandererCanMove = true;
        return;
    }

    int newX = wandererPosition.x;
    int newY = wandererPosition.y;
    bool moved = false;

    switch (input) {
```

```
        case 'w': case 'W': newY--; moved = true; break;
        case 's': case 'S': newY++; moved = true; break;
        case 'a': case 'A': newX--; moved = true; break;
        case 'd': case 'D': newX++; moved = true; break;
        case 'c': case 'C':
            wanderer.switchCombatMode();
            std::cout << "Switched to " <<
                (wanderer.getCombatMode()    ==    CombatMode::MELEE    ?
"MELEE" : "RANGED") << " combat\n";
            return;
        case 'b': case 'B':
            processSpellPurchase();
            return;
        case 'q': case 'Q':
            gameRunning = false;
            std::cout << "Game ended by wanderer.\n";
            return;
    }

    if (input >= '1' && input <= '3') {
        processSpellSelection(input);
        return;
    }

    if (moved) {
        CombatMode mode = wanderer.getCombatMode();

        if (mode == CombatMode::MELEE) {
            if       (field.isValidPosition(newX,       newY)       &&
field.getCell(newX, newY).getHasBeast()) {
                combatManager.processMeleeAttack(Position(newX, newY));
                if (wanderer.getIsAlive()) {
                    field.getCell(wandererPosition.x,
wandererPosition.y).setHasWanderer(true);
                }
                return;
            }
        } else {
            int directionX = newX - wandererPosition.x;
            int directionY = newY - wandererPosition.y;

            combatManager.processRangedAttack(directionX, directionY);
            if (wanderer.getIsAlive()) {
                field.getCell(wandererPosition.x,
wandererPosition.y).setHasWanderer(true);
            }
            return;
        }

        if   (field.moveEntity(wandererPosition.x,   wandererPosition.y,
newX, newY, true)) {
            Cell& newCell = field.getCell(newX, newY);
            if (newCell.getHasTrap()) {
                wanderer.takeDamage(newCell.getTrapDamage());
                std::cout   <<   "You   triggered   a   trap   for   "   <<
newCell.getTrapDamage() << " damage!\n";
                newCell.triggerTrap();
```

```
            }
            updateAllManagersPosition(Position(newX, newY));
            applyCellEffects(newX, newY);
        } else {
            std::cout << "Cannot move there!\n";
            if (wanderer.getIsAlive()) {
                field.getCell(wandererPosition.x,
wandererPosition.y).setHasWanderer(true);
            }
        }
    }
}

void InputHandler::applyCellEffects(int x, int y) {
    CellType cellType = field.getCell(x, y).getType();

    if (cellType == CellType::SLOW && !wandererOnSlowCell) {
        wandererOnSlowCell = true;
        wandererCanMove = false;
        std::cout << "Stepped on slow cell! Next turn skipped.\n";
    } else if (cellType != CellType::SLOW && wandererOnSlowCell) {
        wandererOnSlowCell = false;
    }
}

void InputHandler::processSpellPurchase() {
    Hand& hand = wanderer.getHand();
    if (hand.getSpellCount() >= hand.getMaxSize()) {
        std::cout << "Spell hand is full! Max size: " <<
hand.getMaxSize() << "\n";
        return;
    }

    if (wanderer.getPoints() < 50) {
        std::cout << "Not enough points! Need 50, but have " <<
wanderer.getPoints() << "\n";
        return;
    }

    SpellFactory factory;
    auto newSpell = factory.createRandomSpell();
    if (newSpell) {
        wanderer.addPoints(-50);
        if (hand.addSpell(std::move(newSpell))) {
            std::cout << "Purchased new spell for 50 points!\n";
        } else {
            std::cout << "Failed to add spell to hand!\n";
            wanderer.addPoints(50);
        }
    }
}

void InputHandler::processSpellSelection(char input) {
    int spellIndex = input - '1';

    if (spellIndex < wanderer.getHand().getSpellCount()) {
```

```
        SpellCard* spell = wanderer.getHand().getSpell(spellIndex);
        if (spell && !spell->isUsed()) {
            if (wanderer.spendMana(spell->getManaCost())) {
                startSpellTargetSelection(spellIndex);
            } else {
                std::cout << "Not enough mana! Need " << spell->getManaCost()
                          << ", but have " << wanderer.getMana() << "\n";
            }
        } else {
            std::cout << "Spell not available or already used!\n";
        }
    } else {
        std::cout << "No spell in slot " << (spellIndex + 1) << "!\n";
    }
}

void InputHandler::processSpellTargetSelection(char input) {
    if (!selectingSpellTarget) return;

    SpellCard* spell = wanderer.getHand().getSpell(selectedSpellIndex);
    if (!spell) {
        cancelSpellTargetSelection();
        return;
    }

    int targetX = wandererPosition.x;
    int targetY = wandererPosition.y;
    bool moved = false;

    switch (input) {
        case 'w': case 'W': targetY--; moved = true; break;
        case 's': case 'S': targetY++; moved = true; break;
        case 'a': case 'A': targetX--; moved = true; break;
        case 'd': case 'D': targetX++; moved = true; break;
        case 'x': case 'X':
            cancelSpellTargetSelection();
            std::cout << "Spell casting cancelled.\n";
            wanderer.addMana(spell->getManaCost());
            return;
    }

    if (moved) {
        int distance = std::max(std::abs(targetX - wandererPosition.x),
                                std::abs(targetY - wandererPosition.y));

        if (distance <= spell->getRange()) {
            spell->cast(Position(targetX, targetY), field);
            selectingSpellTarget = false;
            selectedSpellIndex = -1;
            wandererCanMove = false;
        } else {
            std::cout << "Target out of range! Max range: " << spell->getRange() << "\n";
        }
```

```
    }
}

bool InputHandler::isSelectingSpellTarget() const {
    return selectingSpellTarget;
}

void InputHandler::startSpellTargetSelection(int spellIndex) {
    selectingSpellTarget = true;
    selectedSpellIndex = spellIndex;
    SpellCard* spell = wanderer.getHand().getSpell(spellIndex);
    std::cout << "Select target for " << spell->getName()
              << " (WASD to move cursor, X to cancel)\n";
}

void InputHandler::cancelSpellTargetSelection() {
    selectingSpellTarget = false;
    selectedSpellIndex = -1;
}
```

Файл InputHandler.h:

```
#ifndef INPUT_HANDLER_H
#define INPUT_HANDLER_H

#include "GameField.h"
#include "Wanderer.h"
#include "CombatManager.h"
#include "BeastManager.h"
#include "SpellFactory.h"

class InputHandler {
private:
    Wanderer& wanderer;
    GameField& field;
    Position& wandererPosition;
    CombatManager& combatManager;
    BeastManager& beastManager;
    bool& wandererCanMove;
    bool& wandererOnSlowCell;
    bool& gameRunning;
    bool selectingSpellTarget;
    int selectedSpellIndex;

public:
    InputHandler(Wanderer&   wanderer,   GameField&   field,   Position&
wandererPos,
                 CombatManager&      combatManager,      BeastManager&
beastManager, bool& canMove, bool& onSlowCell, bool& running);

    void processInput(char input);
    void applyCellEffects(int x, int y);
    void processSpellSelection(char input);
    void processSpellTargetSelection(char input);
    void processSpellPurchase();
    bool isSelectingSpellTarget() const;
    void startSpellTargetSelection(int spellIndex);
    void cancelSpellTargetSelection();
```

```
    void updateAllManagersPosition(const Position& newPos);
};

#endif
```

Файл SpellFactory.cpp:

```
#include "SpellFactory.h"
#include <iostream>

SpellFactory::SpellFactory() : gen(rd()) {}

std::unique_ptr<SpellCard> SpellFactory::createRandomSpell() {
    std::uniform_int_distribution<> dis(0, 3);
    int choice = dis(gen);

    std::unique_ptr<SpellCard> spell = nullptr;
    switch (choice) {
        case 0:
            spell = createFireball();
            std::cout << "Created Fireball spell\n";
            break;
        case 1:
            spell = createPunishmentOfZeus();
            std::cout << "Created Lightning Bolt spell\n";
            break;
        case 2:
            spell = createFrosty();
            std::cout << "Created Frost Nova spell\n";
            break;
        case 3:
            spell = createTrapSpell();
            std::cout << "Created Beast Trap spell\n";
            break;
        default:
            spell = createFireball();
            std::cout << "Created default Fireball spell\n";
            break;
    }
    return spell;
}

std::unique_ptr<SpellCard> SpellFactory::createFireball() {
    auto directSpell = std::make_unique<DirectDamageSpell>("Fireball",
30, 3, 4);
    return std::make_unique<SpellCard>(std::move(directSpell));
}

std::unique_ptr<SpellCard> SpellFactory::createPunishmentOfZeus() {
    auto directSpell = std::make_unique<DirectDamageSpell>("Lightning
Bolt", 25, 2, 5);
    return std::make_unique<SpellCard>(std::move(directSpell));
}

std::unique_ptr<SpellCard> SpellFactory::createFrosty() {
    auto areaSpell = std::make_unique<AreaSpell>("Frosty", 15, 4, 3);
    return std::make_unique<SpellCard>(std::move(areaSpell));
```

```
}

std::unique_ptr<SpellCard> SpellFactory::createTrapSpell() {
    auto trapSpell = std::make_unique<TrapSpell>("Beast Trap", 25, 3,
3);
    return std::make_unique<SpellCard>(std::move(trapSpell));
}
```

Файл SpellFactory.h:

```
#ifndef SPELL_FACTORY_H
#define SPELL_FACTORY_H

#include "DirectDamageSpell.h"
#include "AreaSpell.h"
#include "TrapSpell.h"
#include <random>
#include <memory>

class SpellFactory {
private:
    std::random_device rd;
    std::mt19937 gen;

public:
    SpellFactory();
    std::unique_ptr<SpellCard> createRandomSpell();
    std::unique_ptr<SpellCard> createFireball();
    std::unique_ptr<SpellCard> createPunishmentOfZeus();
    std::unique_ptr<SpellCard> createFrosty();
    std::unique_ptr<SpellCard> createTrapSpell();
};

#endif
```

Файл TrapSpell.cpp:

```
#include "TrapSpell.h"
#include <iostream>

TrapSpell::TrapSpell(std::string name, int damage, int manaCost, int
range)
    : name(name), damage(damage), manaCost(manaCost), range(range) {
    description = "Places a trap that deals " + std::to_string(damage)
+ " damage when triggered";
}

void TrapSpell::cast(Position target, GameField& field) {
    if (!field.isValidPosition(target.x, target.y)) {
        std::cout << "Invalid target position for trap!\n";
        return;
    }

    Cell& cell = field.getCell(target.x, target.y);
    if (!cell.isEmpty()) {
        std::cout << "Cannot place trap on occupied cell!\n";
        return;
```

```
    }

    cell.setHasTrap(true);
    cell.setTrapDamage(damage);
    std::cout << name << " placed at position (" << target.x << ", " <<
target.y << ")\n";
}
```

Файл TrapSpell.h

```
#ifndef TRAP_SPELL_H
#define TRAP_SPELL_H

#include "Spell.h"
#include "GameField.h"
#include <string>

class TrapSpell : public ISpell {
private:
    std::string name;
    std::string description;
    int damage;
    int manaCost;
    int range;

public:
    TrapSpell(std::string name, int damage, int manaCost = 3, int range
= 2);

    void cast(Position target, GameField& field) override;
    std::string getName() const override { return name; }
    std::string getDescription() const override { return description; }
    int getManaCost() const override { return manaCost; }
    int getRange() const override { return range; }
    SpellType getType() const override { return SpellType::TRAP; }
};

#endif
```