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»

Студент гр. 4384 _____ Овчаренко Я.В.

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 интерфейс карточки заклинания. Заклинание должно применяться игроком. На использование заклинания игрок тратит один ход.

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.

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.

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.

На 8/4/1.5 баллов:

Реализовать интерфейс заклинания ловушки. Заклинание размещает на поле ловушку, если враг наступает на клетку с ловушкой, то ему наносится урон, и ловушка пропадает.

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.

На 10/5/2 баллов:

Реализовать интерфейс заклинания призыва. Заклинание создает союзника рядом с игроком, который перемещается самостоятельно.

Реализовать интерфейс заклинание улучшения. Заклинание улучшает следующее используемое заклинание:

Заклинание прямого урона - увеличивает радиус применения

Заклинание урона по площади - увеличивает площадь

Заклинание ловушки - увеличивает урон

Заклинание призыва - призывает больше союзников

Заклинание улучшение - накапливает усиление, то есть при применении следующего заклинания отличного от улучшения, все улучшения применяются сразу

Выполнение работы.

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

- Интерфейс карточки заклинания;
- Класс "руки" игрока;
- Класс заклинания прямого урона;
- Класс заклинания урона по площади;
- Класс заклинания ловушки;
- Класс вражеской башни;
- Класс заклинания призыва;
- Класс заклинания улучшения.

Архитектура программы.

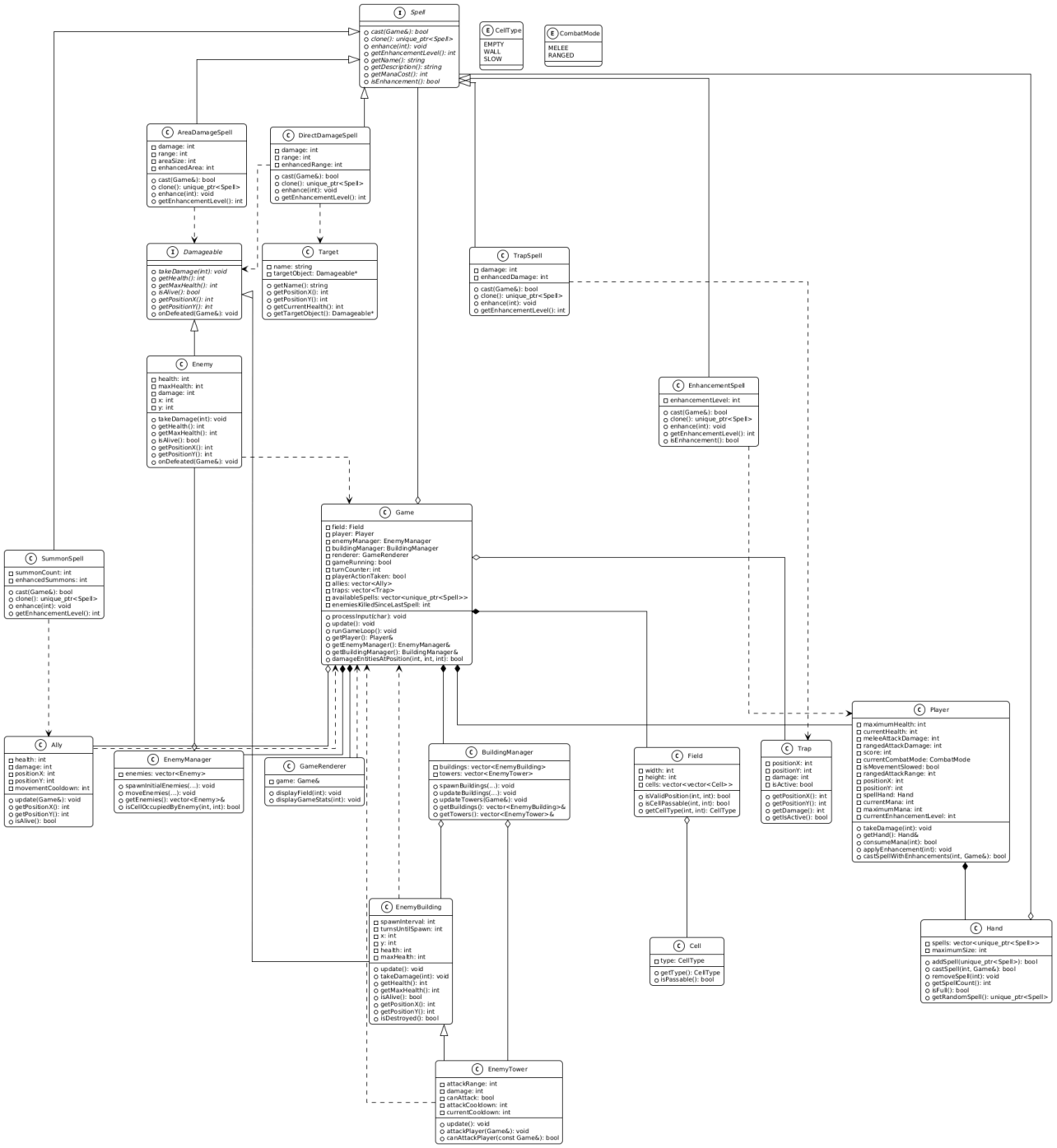

Рисунок 1 - Диаграмма взаимосвязи классов

В программе реализована иерархия классов, соответствующая принципам ООП.

Основные классы:

- Player - класс игрока
- Enemy - класс врага
- EnemyBuilding, EnemyTower - вражеские сооружения

- Cell - класс клетки поля
- Field - класс игрового поля
- EnemyManager - класс управления врагами
- BuildingManager - класс управления зданиями
- GameRenderer - класс отображения игры
- Game - основной класс, управляющий игровым циклом
- Spell и наследники - система заклинаний
- Ally - союзные персонажи
- Trap - ловушки
- Hand - управление заклинаниями игрока
- Target - система целей для заклинаний

Дополнительные enum классы:

- CellType – класс типов клеток
- CombatMode – класс режимов боя
- TargetType - типы целей для заклинаний

Описание классов.

Класс Game

Основной класс, управляющий игровым циклом и координирующий работу всех компонентов.

Поля класса:

- field - игровое поле
- player - игрок
- enemyManager - менеджер врагов
- buildingManager - менеджер зданий
- renderer - рендерер игры
- gameRunning - состояние игры
- turnCounter - счётчик ходов
- playerActionTaken - флаг совершения действия игроком
- allies - вектор союзников
- traps - вектор ловушек

- availableSpells - вектор доступных заклинаний
- enemiesKilledSinceLastSpell - счётчик убийств для получения заклинаний

Методы класса:

- processInput(char) - обработка ввода игрока
- update() - обновление игрового состояния
- runGameLoop() - запуск игрового цикла
- isGameRunning() - проверка состояния игры
- displayField() - отображение поля
- initializeSpells() - инициализация заклинаний
- giveRandomSpell() - выдача случайного заклинания
- buySpell() - покупка заклинания
- castSpell(int) - применение заклинания
- displaySpells() - отображение заклинаний
- spawnPlayer() - размещение игрока на поле
- processCellEffects(int, int) - обработка эффектов клетки
- displayHelp() - отображение помощи
- updateAllies() - обновление союзников
- checkTraps() - проверка ловушек
- addAlly(const Ally&) - добавление союзника
- addTrap(const Trap&) - добавление ловушки
- isCellOccupiedByAlly(int, int) - проверка занятости клетки союзником
- isCellOccupiedByEnemy(int, int) - проверка занятости клетки врагом

Связи с другими классами:

- **Player** (композиция) - содержит объект игрока
- **EnemyManager** (композиция) - содержит менеджер врагов
- **BuildingManager** (композиция) - содержит менеджер зданий
- **Field** (композиция) - содержит игровое поле
- **Ally** (агрегация) - содержит вектор союзников
- **Trap** (агрегация) - содержит вектор ловушек
- **Spell** (ассоциация) - взаимодействует с системой заклинаний

Класс Player

Класс содержит характеристики и методы игрока.

Поля класса:

- health, maxHealth – текущее и максимальное здоровье
- meleeDamage, rangedDamage – урон в ближнем и дальнем бою
- score – очки игрока
- combatMode – текущий режим боя
- isSlowed – состояние замедления
- rangedAttackRange – дистанция дальней атаки
- x, y – координаты на поле
- hand - рука с заклинаниями
- mana, maxMana - текущая и максимальная мана
- currentEnhancementLevel - текущий уровень улучшения заклинаний

Методы класса:

- getDamage() - возвращает урон в зависимости от текущего режима боя
- takeDamage(int) - метод получения урона
- setPosition(int, int) - метод изменения позиции игрока
- switchCombatMode() - метод смены типа атаки
- addScore(int) - метод получения очков
- setSlowed(bool) - метод установки состояния замедления
- getHand() - получение руки с заклинаниями
- consumeMana(int) - потребление маны
- restoreMana(int) - восстановление маны
- applyEnhancement(int) - применение улучшения к следующему заклинанию
- getCurrentEnhancementLevel() - получение текущего уровня улучшения
- castSpellWithEnhancements(int, Game&) - применение заклинания с учетом улучшений

Связи с другими классами:

- **Hand** (композиция) - содержит руку с заклинаниями

- **Game** (ассоциация) - взаимодействует с игровым миром
- **CombatMode** (ассоциация) - использует перечисление режимов боя
- **Enemy** (ассоциация) - атакует врагов
- **EnemyBuilding** (ассоциация) - атакует здания

Класс Enemy

Класс содержит характеристики и методы вражеских персонажей.

Поля класса:

- health – здоровье врага
- maxHealth - максимальное здоровье
- damage – урон врага
- x, y – координаты на поле

Методы класса:

- takeDamage(int) - метод получения урона
- setPosition(int, int) - метод изменения позиции
- isAlive() - проверка, жив ли враг
- getX(), getY() - получение координат
- getHealth(), getMaxHealth() - получение здоровья
- getDamage() - получение урона

Связи с другими классами:

- **EnemyManager** (ассоциация) - управляется менеджером врагов
- **Player** (ассоциация) - атакует игрока
- **Field** (ассоциация) - занимает позицию на поле
- **Spell** (ассоциация) - получает урон от заклинаний

Класс EnemyBuilding

Класс содержит характеристики и методы вражеских зданий.

Поля класса:

- spawnInterval – период спавна врага
- turnsUntilSpawn – счётчик ходов до следующего спавна
- x, y – координаты на поле
- health, maxHealth - здоровье здания

Методы класса:

- update() - уменьшение счётчика ходов
- shouldSpawnEnemy() - метод проверки возможности спавна
- resetSpawnTimer() - обнуление счётчика ходов
- takeDamage(int) - получение урона
- isDestroyed() - проверка уничтожения
- getX(), getY() - получение координат
- getHealth(), getMaxHealth() - получение здоровья

Связи с другими классами:

- **BuildingManager** (ассоциация) - управляется менеджером зданий
- **EnemyManager** (ассоциация) - спавнит врагов через менеджер
- **Field** (ассоциация) - занимает позицию на поле
- **EnemyTower** (наследование) - базовый класс для башен

Класс EnemyTower

Специализированное вражеское сооружение с дальним боем (наследуется от EnemyBuilding).

Поля класса:

- attackRange - дальность атаки
- damage - урон
- canAttack - флаг возможности атаки
- attackCooldown - перезарядка атаки
- currentCooldown - текущая перезарядка

Методы класса:

- update() - обновление состояния башни
- canAttackPlayer(const Game&) - проверка возможности атаки игрока
- attackPlayer(Game&) - атака игрока
- getAttackRange() - получение дальности атаки
- getDamage() - получение урона

Связи с другими классами:

- **EnemyBuilding** (наследование) - наследует базовый функционал

- **Game** (ассоциация) - взаимодействует с игровым миром
- **Player** (ассоциация) - атакует игрока
- **Field** (ассоциация) - проверяет видимость через поле

Класс Field

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

Поля класса:

- width – ширина поля
- height – высота поля
- grid – двумерный массив клеток поля

Методы класса:

- canMoveTo(int, int) - метод проверки возможности перемещения в клетку. Проверяет, находится ли клетка в пределах поля и проходима ли она
- canPlaceEntity(int, int, int, int) - метод проверки возможности размещения сущности. Проверяет, что клетка свободна и не занята игроком
- findPathToPlayer(int, int, int, int, int&, int&) - метод вычисления пути к игроку. Определяет направление движения врага к игроку с учетом препятствий
- hasLineOfSight(int, int, int, int) - метод проверки видимости между двумя точками
- isCellPassable(int, int) - проверка проходимости клетки
- isSlowCell(int, int) - проверка, является ли клетка замедляющей
- isValidPosition(int, int) - проверка валидности координат

Связи с другими классами:

- **Cell** (композиция) - содержит сетку клеток
- **Game** (ассоциация) - используется игровым миром
- **EnemyManager** (ассоциация) - используется для перемещения врагов
- **BuildingManager** (ассоциация) - используется для размещения зданий
- **Player** (ассоциация) - отслеживает позицию игрока

Класс EnemyManager

Класс управления вражескими персонажами.

Поля класса:

- enemies – массив вражеских персонажей

Методы класса:

- spawnInitialEnemies(Field&, BuildingManager&, Player&, int) - метод создания начальных врагов. Размещает врагов на поле в случайных позициях, избегая зданий и игрока
- moveEnemies(Field&, Player&, BuildingManager&, const Game&) - метод процесса хода врага. Передвигает каждого живого врага в направлении игрока. Если враг пытается перейти на клетку с игроком, то перемещение не происходит, и игроку наносится урон. Если враг пытается перейти на клетку с другим врагом или зданием, перемещение не происходит.
- attackEnemyAtPosition(int, int, Player&) - метод атаки врага. Если игрок пытается перейти на клетку с врагом, то перемещение не происходит, и врагу наносится урон
- performRangedAttack(Field&, Player&, int, int) - метод дальней атаки. Выпускает снаряд в указанном направлении, нанося урон первому встречному врагу или стене
- isCellOccupiedByEnemy(int, int) - проверка занятости клетки врагом

Связи с другими классами:

- **Enemy** (агрегация) - содержит вектор врагов
- **Field** (ассоциация) - для проверки перемещения и позиционирования
- **Player** (ассоциация) - для атак и проверки позиции игрока
- **BuildingManager** (ассоциация) - проверка занятости клеток зданиями
- **Game** (ассоциация) - проверка занятости клеток союзниками

Класс BuildingManager

Класс управления вражескими зданиями.

Поля класса:

- buildings – массив вражеских зданий
- towers – массив вражеских башен

Методы класса:

- spawnBuildings(Field&, Player&, int) - метод добавления на поле зданий. Размещает здания на поле в случайных позициях, избегая позиции игрока
- updateBuildings(Field&, EnemyManager&) - метод процесса хода здания. Проверяет у каждого здания, прошло ли необходимое количество ходов с момента предыдущего спавна. Если да, то пытается спавнить нового врага в соседней клетке. Враги спавнятся вокруг здания, но не внутри его.
- updateTowers(Game&) - обновление состояния башен (атака игрока)
- isCellOccupiedByBuilding(int, int) - проверка занятости клетки зданием
- damageBuildingAt(int, int, int) - нанесение урона зданию в указанной клетке
- damageTowerAt(int, int, int) - нанесение урона башне в указанной клетке
- removeDestroyedBuildings() - удаление уничтоженных зданий и башен

Связи с другими классами:

- **EnemyBuilding** (агрегация) - содержит вектор обычных зданий
- **EnemyTower** (агрегация) - содержит вектор башен
- **Field** (ассоциация) - для размещения зданий на поле
- **EnemyManager** (ассоциация) - для спавна врагов из зданий
- **Game** (ассоциация) - для обновления башен

Класс Hand

Управление заклинаниями в руке игрока.

Поля класса:

- spells - вектор умных указателей на заклинания
- maximumSize - максимальный размер руки

Методы класса:

- addSpell(unique_ptr<Spell>) - добавление заклинания
- castSpell(int, Game&) - применение заклинания по индексу
- removeSpell(int) - удаление заклинания
- getSpellCount() - получение количества заклинаний
- getSpell(int) - получение заклинания по индексу

- isFull() - проверка, заполнена ли рука
- getRandomSpell() - извлечение случайного заклинания
- displaySpells() - отображение заклинаний в руке

Связи с другими классами:

- **Spell** (агрегация) - содержит вектор умных указателей на заклинания
- **Game** (ассоциация) - передает ссылку на Game для применения заклинаний
- **Player** (ассоциация) - управляется игроком
- Базовый класс Spell и его наследники
- **Абстрактный класс Spell** определяет интерфейс для всех заклинаний.

Методы класса:

- cast(Game&) - применение заклинания
- clone() - создание копии заклинания
- enhance(int) - улучшение заклинания
- getEnhancementLevel() - получение уровня улучшения
- getName() - получение имени
- getDescription() - получение описания
- getManaCost() - получение стоимости маны

Наследники:

- DirectDamageSpell - точечный урон с выбором цели
- AreaDamageSpell - площадной урон в указанном направлении
- TrapSpell - установка ловушки
- SummonSpell - призыв союзников
- EnhancementSpell - улучшение следующих заклинаний
- **Связи с другими классами:**
- **Game** (ассоциация) - все заклинания взаимодействуют с игрой
- **Hand** (ассоциация) - хранятся в руке игрока
- **Target** (ассоциация) - система целей для заклинаний

Класс Ally

Союзные персонажи, призываемые игроком.

Поля класса:

- health - здоровье
- damage - урон
- positionX, positionY - координаты
- movementCooldown - задержка между движениями

Методы класса:

- update(Game&) - обновление состояния (движение и атака)
- takeDamage(int) - получение урона
- isAlive() - проверка жизни
- getPositionX(), getPositionY() - получение координат
- getHealth(), getDamage() - получение характеристик

Связи с другими классами:

- **Game** (ассоциация) - взаимодействует с игровым миром
- **Enemy** (ассоциация) - атакует врагов
- **Field** (ассоциация) - перемещается по полю

Класс Trap

Ловушки, устанавливаемые игроком.

Поля класса:

- positionX, positionY - координаты
- damage - урон
- isActive - активна ли ловушка

Методы класса:

- getPositionX(), getPositionY() - получение координат
- getDamage() - получение урона
- getIsActive() - проверка активности
- deactivate() - деактивация ловушки

Связи с другими классами:

- **Game** (ассоциация) - размещается в игровом мире
- **Enemy** (ассоциация) - активируется врагами
- **Field** (ассоциация) - занимает позицию на поле

Класс Target

Представляет цель для заклинаний (универсальный класс для врагов, зданий, башен).

Поля класса:

- name - название цели
- positionX, positionY - координаты
- currentHealth, maximumHealth - здоровье
- targetObject - указатель на объект (const void*)
- type - тип цели (TargetType)

Методы класса:

Геттеры для всех полей

Связи с другими классами:

- **DirectDamageSpell** (ассоциация) - используется для выбора целей
- **Enemy** (ассоциация) - может представлять врагов
- **EnemyBuilding** (ассоциация) - может представлять здания
- **EnemyTower** (ассоциация) - может представлять башни

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: ally.h

```
#ifndef ALLY_H
#define ALLY_H

class Game;

class Ally {
private:
    int health;
    int damage;
    int positionX;
    int positionY;
    int movementCooldown;

public:
    Ally(int health, int damage, int x, int y);

    void update(Game& game);
    int getPositionX() const;
    int getPositionY() const;
    int getHealth() const;
    int getDamage() const;
    bool isAlive() const;
    void takeDamage(int damageAmount);

private:
    void moveTowardsNearestEnemy(Game& game);
    void attackEnemyInRange(Game& game);
    bool isValidMove(const Game& game, int x, int y) const;
};

#endif
```

Название файла: areadamagespell.h

```
#ifndef AREA_DAMAGE_SPELL_H
#define AREA_DAMAGE_SPELL_H


#include "spell.h"
#include <string>


class AreaDamageSpell : public Spell {
private:
     int damage;
     int range;
     int areaSize;
```

```
        int enhancedAreaSize;

        void displayAreaPreview(Game& game, int targetX, int targetY,
int currentAreaSize,
                                int directionX, int directionY) const;

    public:
        AreaDamageSpell(int damage = 15, int range = 2, int areaSize =
2);
        bool cast(Game& game) override;
        std::unique_ptr<Spell> clone() const override;
        void enhance(int enhancementLevel) override;
        int getEnhancementLevel() const override;

        std::string getName() const override { return "Area Damage"; }
        std::string getDescription() const override {
            return "Deals " + std::to_string(damage) + " damage in "
+
                               std::to_string(areaSize)   +   "x"   +
std::to_string(areaSize) + " area";
        }
        int getManaCost() const override { return 25; }
    };

    #endif
```

Название файла: buildingmanager.h

```
#ifndef BUILDING_MANAGER_H
#define BUILDING_MANAGER_H

#include <vector>
#include "enemybuilding.h"
#include "enemytower.h"
#include "gametypes.h"

class BuildingManager {
private:
    std::vector<EnemyBuilding> buildings;
```

```
        std::vector<EnemyTower> towers;

    public:
        void spawnBuildings(Field& field, Player& player, int count);
        void updateBuildings(Field& field, EnemyManager& enemyManager);
        void updateTowers(Game& game);
        bool isCellOccupiedByBuilding(int x, int y) const;

        // Damage buildings and towers
        void damageBuildingAt(int x, int y, int damage);
        void damageTowerAt(int x, int y, int damage);

        // Remove destroyed buildings
        void removeDestroyedBuildings();

         // Getters
         const    std::vector<EnemyBuilding>&    getBuildings()    const
{ return buildings; }
         std::vector<EnemyBuilding>& getBuildings() { return buildings;
}
         const  std::vector<EnemyTower>&  getTowers()  const  {  return
towers; }
         std::vector<EnemyTower>& getTowers() { return towers; }
    };

    #endif
```

Название файла: cell.h

```
    #ifndef CELL_H
    #define CELL_H

    enum class CellType {
        EMPTY,
        WALL,
        SLOW
    };

    class Cell {
```

```
private:
      CellType type;

public:
      Cell(CellType cellType = CellType::EMPTY);

      CellType getType() const;
      void setType(CellType cellType);

      bool isPassable() const;
};

#endif
```

Название файла: damageable.h

```
#ifndef DAMAGEABLE_H
#define DAMAGEABLE_H

class Damageable {
public:
      virtual ~Damageable() = default;

      virtual void takeDamage(int damageAmount) = 0;
      virtual int getHealth() const = 0;
      virtual int getMaxHealth() const = 0;
      virtual bool isAlive() const = 0;
      virtual int getPositionX() const = 0;
      virtual int getPositionY() const = 0;
      virtual void onDefeated(class Game& game) {}
};

#endif
```

Название файла: directdamagespell.h

```
#ifndef DIRECT_DAMAGE_SPELL_H
#define DIRECT_DAMAGE_SPELL_H

#include "spell.h"
```

```
#include <string>
#include <vector>

class Target;

class DirectDamageSpell : public Spell {
private:
	int damage;
	int range;
	int enhancedRange;

public:
	DirectDamageSpell(int damage = 20, int range = 5);
	bool cast(Game& game) override;
	std::unique_ptr<Spell> clone() const override;
	void enhance(int enhancementLevel) override;
	int getEnhancementLevel() const override;

	std::string getName() const override { return "Direct Damage";
}
	std::string getDescription() const override {
		return "Deals " + std::to_string(damage) + " damage to
selected target in range";
	}
	int getManaCost() const override { return 15; }

private:
	std::vector<Target> getTargetsInRange(Game& game, int playerX,
int playerY, int currentRange);
	void displayTargets(const std::vector<Target>& targets) const;
	bool applyDamageToTarget(Game& game, const Target& target, int
damageAmount);
};
#endif
```

Название файла: enemy.h

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

```
#include "damageable.h"

class Game;

class Enemy : public Damageable {
private:
	int health;
	int maxHealth;
	int damage;
	int x;
	int y;

public:
	Enemy(int health = 30, int damage = 10, int x = -1, int y =
-1);

	int getHealth() const override;
	int getMaxHealth() const override;
	int getDamage() const;
	int getX() const;
	int getY() const;
	int getPositionX() const override;
	int getPositionY() const override;

	void takeDamage(int damageAmount) override;
	void setPosition(int newX, int newY);
	bool isAlive() const override;
	void onDefeated(Game& game) override;
};

#endif
```

Название файла: enemybuilding.h

```
#ifndef ENEMY_BUILDING_H
#define ENEMY_BUILDING_H

#include "damageable.h"
```

```
class EnemyBuilding : public Damageable {
private:
	int spawnInterval;
	int turnsUntilSpawn;
	int x;
	int y;
	int health;
	int maxHealth;

public:
	EnemyBuilding(int spawnInterval = 5, int x = -1, int y = -1,
int health = 100);

	virtual void update();
	bool shouldSpawnEnemy();
	void resetSpawnTimer();
	int getTurnsUntilSpawn() const;
	int getX() const;
	int getY() const;
	int getPositionX() const override;
	int getPositionY() const override;
	void setPosition(int newX, int newY);

	int getHealth() const override;
	int getMaxHealth() const override;
	void takeDamage(int damageAmount) override;
	bool isDestroyed() const;
	bool isAlive() const override;
};

#endif
```

Название файла: enemymanager.h

```
#ifndef ENEMY_MANAGER_H
#define ENEMY_MANAGER_H

#include <vector>
```

```
#include "enemy.h"
#include "gametypes.h"

class Game; // Forward declaration

class EnemyManager {
private:
    std::vector<Enemy> enemies;

public:
        void spawnInitialEnemies(Field& field, BuildingManager& buildingManager, Player& player, int count);
        void moveEnemies(Field& field, Player& player, BuildingManager& buildingManager, const Game& game);
        bool attackEnemyAtPosition(int x, int y, Player& player);
        void performRangedAttack(Field& field, Player& player, int directionX, int directionY);
        bool isCellOccupiedByEnemy(int x, int y) const;

        // Геттеры
        const std::vector<Enemy>& getEnemies() const { return enemies; }
        std::vector<Enemy>& getEnemies() { return enemies; }
};

#endif
```

Название файла: enemytower.h

```
#ifndef ENEMY_TOWER_H
#define ENEMY_TOWER_H

#include "enemybuilding.h"

class Game;

class EnemyTower : public EnemyBuilding {
private:
    int attackRange;
```

```
        int damage;
        bool canAttack;
        int attackCooldown;
        int currentCooldown;

    public:
         EnemyTower(int spawnInterval = 5, int x = -1, int y = -1, int
range = 4, int dmg = 8, int health = 150);

        void update() override;
        void attackPlayer(Game& game);
        bool canAttackPlayer(const Game& game) const;
        int getAttackRange() const;
        int getDamage() const;
    };

    #endif
```

Название файла: enhancementspell.h

```
    #ifndef ENHANCEMENT_SPELL_H
    #define ENHANCEMENT_SPELL_H

    #include "spell.h"
    #include <string>

    class EnhancementSpell : public Spell {
    private:
         int enhancementLevel;

    public:
         EnhancementSpell(int level = 1);
         bool cast(Game& game) override;
         std::unique_ptr<Spell> clone() const override;
         void enhance(int enhancementLevel) override;
         int getEnhancementLevel() const override;
         bool isEnhancement() const override { return true; }

         std::string getName() const override { return "Enhancement"; }
```

```
        std::string getDescription() const override {
            return "Enhances next spell by " +
std::to_string(enhancementLevel) + " level(s)";
        }
        int getManaCost() const override { return 20; }
    };

    #endif
```

Название файла: entity.h

```
    #ifndef ENTITY_H
    #define ENTITY_H

    class Game;  // Forward declaration

    class Entity {
    protected:
        int health;
        int x;
        int y;

    public:
        Entity(int health = 100, int x = -1, int y = -1);
        virtual ~Entity() = default;

        virtual void update(Game& game) = 0;

        int getHealth() const;
        int getX() const;
        int getY() const;
        void setPosition(int newX, int newY);
        void takeDamage(int damage);
        bool isAlive() const;
    };

    #endif
```

Название файла: field.h

```
#ifndef FIELD_H
#define FIELD_H

#include <vector>
#include <memory>
#include <utility>
#include "cell.h"

class Field {
private:
    std::vector<std::vector<Cell>> grid;
    int width;
    int height;


public:
    Field(int width, int height);

    Field(const Field& other);
    Field(Field&& other) noexcept;
    Field& operator=(const Field& other);
    Field& operator=(Field&& other) noexcept;
    ~Field() = default;

    int getWidth() const;
    int getHeight() const;
    Cell getCell(int x, int y) const;
    CellType getCellType(int x, int y) const;

    bool canMoveTo(int x, int y) const;

      bool canPlaceEntity(int x, int y, int playerX, int playerY)
const;

    bool isCellPassable(int x, int y) const;
    bool isSlowCell(int x, int y) const;
    bool isValidPosition(int x, int y) const;
```

```
        void findPathToPlayer(int fromX, int fromY, int playerX, int
playerY, int& moveX, int& moveY) const;

        bool hasLineOfSight(int fromX, int fromY, int toX, int toY)
const;
    };

    #endif
```

Название файла: game.h

```
    #ifndef GAME_H
    #define GAME_H

    #include "field.h"
    #include "player.h"
    #include "enemymanager.h"
    #include "buildingmanager.h"
    #include "gamerenderer.h"
    #include "ally.h"
    #include "trap.h"
    #include <iostream>
    #include <vector>
    #include <memory>

    class Spell;

    class Game {
    private:
        Field field;
        Player player;
        EnemyManager enemyManager;
        BuildingManager buildingManager;
        GameRenderer renderer;
        bool gameRunning;
        int turnCounter;
        bool playerActionTaken;
        std::vector<Ally> allies;
        std::vector<Trap> traps;
```

```
        std::vector<std::unique_ptr<Spell>> availableSpells;
        int enemiesKilledSinceLastSpell;
        const int ENEMIES_PER_SPELL = 3;
        const int SPELL_COST = 50;
        void spawnPlayer();
        void processCellEffects(int x, int y);
        void displayHelp() const;
        void checkTraps();
        void updateAllies();

    public:
        Game(int fieldWidth, int fieldHeight);

        bool isGameRunning() const;
        void displayField() const;
        void processInput(char input);
        void update();
        void runGameLoop();

        // Геттеры для доступа к приватным полям
        Field& getField() { return field; }
        const Field& getField() const { return field; }

        Player& getPlayer() { return player; }
        const Player& getPlayer() const { return player; }

        EnemyManager& getEnemyManager() { return enemyManager; }
            const EnemyManager& getEnemyManager() const { return
enemyManager; }

        BuildingManager& getBuildingManager() { return buildingManager;
}
          const BuildingManager& getBuildingManager() const { return
buildingManager; }

        std::vector<Ally>& getAllies() { return allies; }
        const std::vector<Ally>& getAllies() const { return allies; }

        std::vector<Trap>& getTraps() { return traps; }
```

```
    const std::vector<Trap>& getTraps() const { return traps; }

    void addAlly(const Ally& ally);
    void addTrap(const Trap& trap);
    void removeTrap(int index);
    void initializeSpells();
    void displaySpells() const;
    bool castSpell(int spellIndex);
    void giveRandomSpell();
    void giveSpellForEnemyKill();
    bool buySpell();
     bool isCellOccupiedByAlly(int x, int y) const;
     bool isCellOccupiedByEnemy(int x, int y) const;
     bool damageEntitiesAtPosition(int x, int y, int damageAmount);
};

#endif
```

Название файла: gamerenderer.h

```
#ifndef GAME_RENDERER_H
#define GAME_RENDERER_H

class Game;

class GameRenderer {
public:
     GameRenderer(Game& game);

     void displayField(int turnCounter) const;
     void displayGameStats(int turnCounter) const;

private:
     Game& game;
};

#endif
```

Название файла: gametypes.h

```
#ifndef GAME_TYPES_H
#define GAME_TYPES_H

class Field;
class Player;
class EnemyManager;
class BuildingManager;

#endif
```

Название файла: hand.h

```
#ifndef HAND_H
#define HAND_H

#include "spell.h"
#include <vector>
#include <memory>
#include <random>

class Game;

class Hand {
private:
	std::vector<std::unique_ptr<Spell>> spells;
	int maximumSize;

public:
	explicit Hand(int size);

	bool addSpell(std::unique_ptr<Spell> newSpell);
	bool castSpell(int spellIndex, Game& game);
	void removeSpell(int spellIndex);

	int getSpellCount() const;
	const Spell* getSpell(int spellIndex) const;
	std::vector<std::unique_ptr<Spell>>& getSpells();
	const std::vector<std::unique_ptr<Spell>>& getSpells() const;
	int getMaximumSize() const;
```

```
    bool isFull() const;

    std::unique_ptr<Spell> getRandomSpell();
    void displaySpells() const;
};

#endif
```

Название файла: player.h

```
#ifndef PLAYER_H
#define PLAYER_H

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

class Spell;
class Game;

enum class CombatMode {
    MELEE,
    RANGED
};

class Player {
private:
    int maximumHealth;
    int currentHealth;
    int meleeAttackDamage;
    int rangedAttackDamage;
    int score;
    CombatMode currentCombatMode;
    bool isMovementSlowed;
    int rangedAttackRange;
    int positionX;
    int positionY;
    Hand spellHand;
    int currentMana;
```

```
        int maximumMana;
        int currentEnhancementLevel;

    public:
        Player(int health = 100, int meleeDamage = 15, int rangedDamage
= 8,
               int rangedRange = 3, int handSize = 5, int mana = 100);

        int getX() const;
        int getY() const;
        void setPosition(int newX, int newY);

        int getHealth() const;
        int getMaxHealth() const;
        int getDamage() const;
        int getScore() const;
        CombatMode getCombatMode() const;
        bool isSlowed() const;
        int getRangedAttackRange() const;

        void takeDamage(int damageAmount);
        void addScore(int points);
        void switchCombatMode();
        void setSlowed(bool slowedStatus);
        void heal(int healAmount);
        bool isAlive() const;

        Hand& getHand();
        const Hand& getHand() const;
        int getMana() const;
        int getMaxMana() const;
        bool consumeMana(int manaAmount);
        void restoreMana(int manaAmount);

        void applyEnhancement(int enhancementLevel);
        void clearEnhancements();
        int getCurrentEnhancementLevel() const;
        bool hasActiveEnhancement() const;
        bool castSpellWithEnhancements(int spellIndex, Game& game);
```

```
};

#endif
```

Название файла: spell.h

```
#ifndef SPELL_H
#define SPELL_H

#include <memory>
#include <string>

class Game;

class Spell {
public:
     virtual ~Spell() = default;

     virtual bool cast(Game& game) = 0;
     virtual std::unique_ptr<Spell> clone() const = 0;
     virtual void enhance(int enhancementLevel) = 0;
     virtual int getEnhancementLevel() const { return 0; }

     virtual std::string getName() const = 0;
     virtual std::string getDescription() const = 0;
     virtual int getManaCost() const = 0;

     virtual bool isEnhancement() const { return false; }
};

#endif
```

Название файла: summonspell.h

```
#ifndef SUMMON_SPELL_H
#define SUMMON_SPELL_H

#include "spell.h"
#include <string>
```

```
class SummonSpell : public Spell {
private:
	int summonCount;
	int enhancedSummons;

public:
	SummonSpell(int count = 1);
	bool cast(Game& game) override;
	std::unique_ptr<Spell> clone() const override;
	void enhance(int enhancementLevel) override;
	int getEnhancementLevel() const override;

	std::string getName() const override { return "Summon Ally"; }
	std::string getDescription() const override {
		return "Summons " + std::to_string(summonCount) + " ally
to fight for you";
	}
	int getManaCost() const override { return 30; }
};

#endif
```

Название файла: target.h

```
#ifndef TARGET_H
#define TARGET_H

#include "damageable.h"
#include <string>

class Target {
private:
	std::string name;
	Damageable* targetObject;

public:
	Target(const std::string& targetName, Damageable* object);

	std::string getName() const;
```

```
    int getPositionX() const;
    int getPositionY() const;
    int getCurrentHealth() const;
    int getMaximumHealth() const;
    Damageable* getTargetObject() const;
};

#endif
```

Название файла: trap.h

```
#ifndef TRAP_H
#define TRAP_H

class Trap {
private:
    int positionX;
    int positionY;
    int damage;
    bool isActive;

public:
    Trap(int x, int y, int trapDamage);

    int getPositionX() const;
    int getPositionY() const;
    int getDamage() const;
    bool getIsActive() const;
    void deactivate();
};

#endif
```

Название файла: trapspell.h

```
#ifndef TRAP_SPELL_H
#define TRAP_SPELL_H

#include "spell.h"
#include <string>
```

```
class TrapSpell : public Spell {
private:
	int damage;
	int enhancedDamage;

public:
	TrapSpell(int damage = 30);
	bool cast(Game& game) override;
	std::unique_ptr<Spell> clone() const override;
	void enhance(int enhancementLevel) override;
	int getEnhancementLevel() const override;

	std::string getName() const override { return "Trap"; }
	std::string getDescription() const override {
		return "Places trap dealing " + std::to_string(damage) +
" damage";
	}
	int getManaCost() const override { return 10; }
};

#endif
```