МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра МО ЭВМ

ОТЧЕТ

по лабораторной работе №3

по дисциплине «Объектно-ориентированное программирование»

Студент гр. 4384 _____ Овчаренко Я.В.

Преподаватель _____ Жангиров Т.Р.

Санкт-Петербург

2025

Цель работы.

Изучить и применить принципы объектно-ориентированного программирования для разработки системы сохранения/загрузки игрового состояния, реализации механизма перехода между уровнями с прогрессирующей сложностью и создания системы улучшений характеристик игрока. Разработать надежную систему обработки исключительных ситуаций при работе с файлами и данных, обеспечивающую консистентность состояния игры между сеансами.

Задание.

На 6/3/1 баллов:

Создать класс(ы) игры, который реализует основной цикл игры, и которому передаются команды от пользователя. Игровой цикл состоит из следующих шагов:

Начало игры

Запуск уровня

Ход игрока. Ход, атака или применение заклинания.

Ход союзников - если имеются

Ход врагов

Ход вражеской базы и башни - если имеются

Условие прохождения уровня студент определяет самостоятельно. Если игрок проигрывает, то игроку должно предлагаться начать заново игру, либо выйти из программы.

Все взаимодействие должно происходить через классы игры.

Реализовать систему сохранения и загрузки игры. Пользователь должен иметь возможность сохранить игру в любой момент. Пользователь должен иметь возможность загружаться при запуске программы (или выбрать новую), либо во время игры. Сохранения должны оставаться в консистентном состоянии между запусками игры.

Добавить обработку исключительных ситуаций для загрузки/сохранения, например, невозможность записать в файл, нельзя загрузиться так как файл не существует или в нем некорректные данные.

На 8/4/1.5 баллов:

Реализовать переход на следующий уровень, после прохождения уровня. При переходе на следующий уровень создается новое поле другого размера с более сильными врагами. При переходе на следующий уровень, значение жизни игрока восстанавливается, и половина его карточек заклинаний случайным образом удаляется.

На 10/5/2 баллов:

Реализовать прокачку игрока при переходе между уровнями. Пользователь может улучшить характеристики игрока или улучшить заклинание (что и как улучшать определяет студент). Для этого нужно расширить игровой цикл.

Выполнение работы.

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

- Классы исключений для обработки ошибок сохранения/загрузки
- Классы сериализации для преобразования состояния игры
- Класс GamePersistence для управления сохранением/загрузкой
- Систему уровней с улучшением характеристик

Архитектура программы.

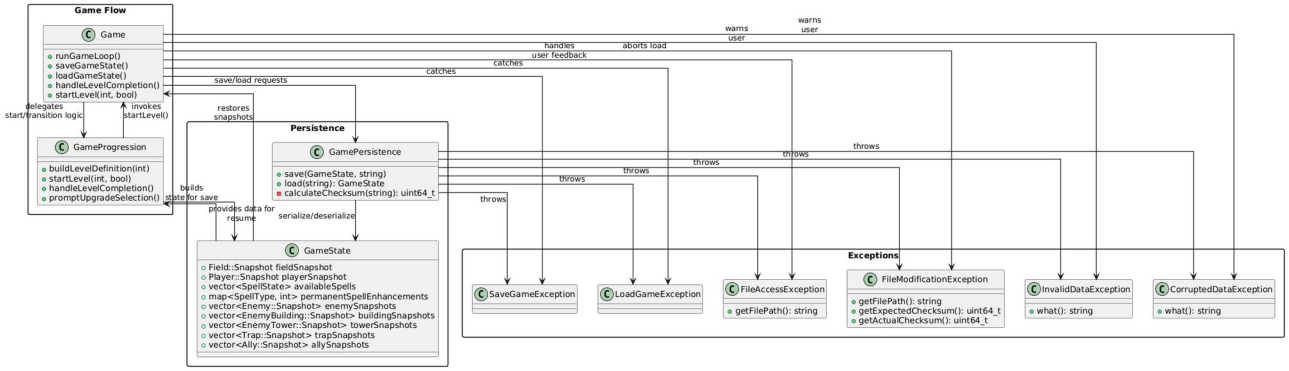

Рисунок 1 - Диаграмма взаимосвязи классов

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
- GamePersistence - класс для сохранения и загрузки игры
- GameState - структура для хранения состояния игры
- SpellSerializer - сериализатор заклинаний
- SpellContext и GameSpellContext - система контекста для заклинаний

Дополнительные enum классы:

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

Новые классы системы сохранения/загрузки:

- SaveGameException - исключение для ошибок сохранения
- LoadGameException - базовое исключение для ошибок загрузки
- FileAccessException - исключение для ошибок доступа к файлу
- CorruptedDataException - исключение для поврежденных данных
- InvalidDataException - исключение для некорректных значений данных
- SpellState - структура состояния заклинания
- GameState - структура полного состояния игры

Описание классов.

Класс GamePersistence

Назначение: Основной класс для сохранения и загрузки состояния игры в файл. Использует идиому RAII для работы с файлами.

Поля класса:

Не содержит полей, все методы работают с передаваемыми параметрами

Методы класса:

- save(const GameState& state, const std::string& path) - сохраняет состояние игры в файл
- load(const std::string& path): GameState - загружает состояние игры из файла

Логика работы:

При сохранении последовательно записывает все компоненты GameState

При загрузке выполняет валидацию всех данных

Использует сигнатуру файла для проверки версии

Обрабатывает различные сценарии ошибок

Связи с другими классами:

- GameState (ассоциация) - работает с состоянием игры
- Исключения (ассоциация) - выбрасывает соответствующие исключения

Класс GameState

Назначение: Структура для хранения полного состояния игры при сохранении/загрузке.

Поля класса:

- levelIndex - текущий уровень
- turnCounter - счетчик ходов
- playerActionTaken - флаг действия игрока
- enemiesKilledSinceLastSpell - счетчик убийств для получения заклинаний
- baseFieldWidth, baseFieldHeight - базовые размеры поля
- currentEnemyHealth, currentEnemyDamage - параметры врагов на текущем уровне
- fieldSnapshot - снимок состояния поля
- playerSnapshot - снимок состояния игрока
- enemySnapshots - вектор снимков врагов
- buildingSnapshots - вектор снимков зданий
- towerSnapshots - вектор снимков башен
- trapSnapshots - вектор снимков ловушек
- allySnapshots - вектор снимков союзников
- availableSpells - вектор состояний заклинаний

- permanentSpellEnhancements - карта постоянных улучшений заклинаний

Связи с другими классами:

- Field::Snapshot (композиция) - содержит снимок поля
- Player::Snapshot (композиция) - содержит снимок игрока
- SpellState (композиция) - содержит состояния заклинаний

Класс SpellState

Назначение: Структура для хранения состояния заклинания при сериализации.

Поля класса:

- type: SpellType - тип заклинания
- enhancementLevel: int - уровень улучшения

Связи с другими классами:

- SpellSerializer (ассоциация) - используется для преобразования
- GameState (ассоциация) - хранится в состоянии игры

Класс SpellSerializer

Назначение: Преобразует объекты Spell в SpellState и обратно.

- Методы класса:
- SpellState serialize(const Spell& spell) — фиксирует тип заклинания и его текущий уровень усиления.
- std::unique_ptr<Spell> deserialize(const SpellState& state) — создаёт конкретный подкласс Spell и применяет сохранённый уровень. Особый случай для EnhancementSpell: уровень передаётся в конструктор, чтобы избежать двойного усиления при загрузке.

Связи с другими классами: используется Player (для руки), Game (для пула доступных заклинаний) и GamePersistence (для записи/чтения).

Классы исключений (SaveGameException, LoadGameException, FileAccessException, FileModificationException, CorruptedDataException, InvalidDataException)

- SaveGameException / LoadGameException — общие ошибки сохранения/загрузки.
- FileAccessException — невозможность открыть файл.
- FileModificationException — несоответствие контрольной суммы.
- CorruptedDataException — повреждённые данные внутри файла.
- InvalidDataException — значения выходят за допустимые пределы.

Связи: генерируются GamePersistence, обрабатываются в Game::saveGameState/Game::loadGameState с выводом понятных сообщений игроку.

Класс Game (расширен)

Новые поля:

- currentLevelIndex - текущий уровень
- campaignActive - активность кампании
- levelInProgress - флаг уровня в процессе
- persistence: GamePersistence - объект для сохранения/загрузки
- std::map<SpellType, int> permanentSpellEnhancements - хранит накопленные бонусы на типы заклинаний и используется при выдаче новых карт или восстановлении руки после загрузки.
- awaitingUpgradeSelection - флаг ожидания выбора улучшения
- GamePersistence persistence - объект, через который выполняются saveGameState() и loadGameState().

Новые методы:

- startLevel(int levelIndex) - запуск уровня
- saveGameState() - сохранение игры
- loadGameState() - загрузка игры
- buildGameState(): GameState - создание снимка состояния
- restoreFromState(const GameState& state) - восстанавливает игру из снимка, правильно применяя сохранённые уровни заклинаний и карту постоянных усилений.
- restoreAvailableSpells(const std::vector<SpellState>& states) — десериализует пул доступных спеллов без повторного наложения перманентных бонусов.
- applyPermanentEnhancementsToSpell(Spell& spell) — единая точка применения постоянных апгрейдов к любому создаваемому заклинанию.
- isLevelCleared(): bool - проверка завершения уровня
- handleLevelCompletion() - обработка завершения уровня
- promptUpgradeSelection() - выбор улучшения
- applyUpgradeChoice(int choice) - применение улучшения
- removeHalfOfPlayerSpells() - удаление половины заклинаний

Разработанный программный код см. в приложении А.

Выводы.

Была успешно реализована система сохранения и загрузки игры, обеспечивающая консистентность состояния между сеансами. Реализована система уровней с прогрессирующей сложностью и улучшением характеристик игрока. Добавлена обработка исключительных ситуаций с информативными сообщениями об ошибках.

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
```