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 с использованием ООП.

Студент гр. 4384 Боков М.О..

Преподаватель Жангиров Т.Р.

Санкт-Петербург
2025

Цель работы.

Разработка игры с использованием принципов объектно-ориентированного программирования, демонстрирующей взаимодействие классов.

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.

Выполнение работы.

В ходе выполнения работы была написана программа на языке C++, реализующая простую игру логикой взаимодействия между игроком и врагами. Программа основана на применении объектно-ориентированного подхода и разделена на несколько классов, каждый из которых отвечает за отдельную часть логики:

Класс Cell

Назначение: представляет одну клетку игрового поля.

Основные методы:

- setType(CellType t) - устанавливает тип клетки (пустая, стена, башня)
- getType() - возвращает текущий тип клетки
- isPassable() - проверяет, можно ли пройти через клетку

Класс Position

Назначение: представляет координаты на игровом поле.

Основные методы:

- getX(), getY() - возвращают координаты
- setX(), setY() - устанавливают координаты
- move(deltaX, deltaY) - перемещает позицию на заданные смещения
- isValid(maxWidth, maxHeight) - проверяет, находится ли позиция в пределах поля

Класс Enemy

Назначение: базовый класс для всех врагов.

Основные методы:

- takeDamage(damage) - наносит урон врагу

- isAlive() - проверяет, жив ли враг
- getDamage() - возвращает урон врага
- getHealth() - возвращает текущее здоровье
- isBuilding() - проверяет, является ли враг зданием

Класс EnemyTower

Назначение: вражеская башня, специализированный тип врага.

Основные методы:

- isBuilding() - возвращает true (башня является зданием)
- attackRange() - атакует игрока, если он в радиусе действия

Класс Player

Назначение: представляет игрока.

Основные методы:

- takeDamage(damage) - наносит урон игроку
- switchCombatMode() - переключает режим боя (ближний/дальний)
- isAlive() - проверяет, жив ли игрок
- getCurrentDamage() - возвращает урон в текущем режиме боя
- castSpell(spellIndex, targetPosition, field) - применяет заклинание
- addMana(amount) - восстанавливает ману
- onEnemyKilled() - вызывается при убийстве врага
- canBuySpell() - проверяет возможность покупки заклинания
- buyRandomSpell() - покупает случайное заклинание

Класс Spell (абстрактный)

Назначение: интерфейс для всех заклинаний.

Основные методы:

- apply(targetPosition, field) - применяет заклинание к цели
- getName() - возвращает название заклинания
- getManaCost() - возвращает стоимость маны
- getDescription() - возвращает описание заклинания

Класс SpellHand

Назначение: "рука" игрока с заклинаниями.

Основные методы:

- addSpell(spell) - добавляет заклинание в руку
- useSpell(index, targetPosition, field) - использует заклинание по индексу
- getSpellCount() - возвращает количество заклинаний
- getMaxSize() - возвращает максимальный размер руки
- getSpellName(index) - возвращает название заклинания
- getSpellManaCost(index) - возвращает стоимость маны

Класс DirectDamageSpell

Назначение: заклинание прямого урона по одной цели.

Основные методы:

- apply(targetPosition, field) - наносит урон врагу или зданию на целевой позиции
- getName() - возвращает "Direct Damage"
- getManaCost() - возвращает стоимость маны
- getDescription() - возвращает описание с параметрами урона и радиуса

Класс AreaDamageSpell

Назначение: заклинание урона по площади 2x2 клетки.

Основные методы:

- apply(targetPosition, field) - наносит урон всем целям в области 2x2
- getName() - возвращает "Area Damage"
- getManaCost() - возвращает стоимость маны
- getDescription() - возвращает описание с параметрами урона, радиуса и площади

Класс TrapSpell

Назначение: заклинание установки ловушки на поле.

Основные методы:

apply(targetPosition, field) - размещает ловушку на указанной позиции

getName() - возвращает "Trap"

getManaCost() - возвращает стоимость маны

getDescription() - возвращает описание с параметрами урона и радиуса

Класс GameField

Взаимодействие с Position:

Хранит: playerPosition, enemyPositions, trapPositions

Использует: Position::isValid() для проверки границ

Создает: новые Position при перемещении объектов

Взаимодействие с Cell:

Хранит: grid - двумерный вектор Cell объектов

Использует: Cell::isPassable(), Cell::getType(), Cell::setType()

Управляет: изменением типов клеток через setCellType()

Взаимодействие с Enemy:

Хранит: enemies - вектор unique_ptr<Enemy>

Использует: Enemy::isAlive(), Enemy::takeDamage(), Enemy::getDamage(), Enemy::isBuilding()

Координирует: позиции врагов через enemyPositions

Взаимодействие с Player:

Принимает: ссылку на Player в moveEnemies()

Использует: Player::takeDamage() при атаках врагов

Взаимодействует: при проверке коллизий и атак

Взаимодействие с Spell системой:

Реализует: placeTrap() для TrapSpell

Обрабатывает: checkTraps() для активации ловушек

Предоставляет: методы для доступа к врагам и зданиям

Класс GameEngine

Взаимодействие с GameField:

Содержит: объект GameField как член данных

Делегирует: большинство операций с полем (движение, коллизии)

Координирует: общий игровой цикл через поле

Взаимодействие с Player:

Содержит: объект Player как член данных

Управляет: состоянием игрока (здоровье, мана, очки)

Координирует: применение заклинаний и смену режима боя

Взаимодействие с Spell системой:

Обрабатывает: castSpell() - применение заклинаний игроком

Управляет: покупкой заклинаний через buyRandomSpell()

Предоставляет: информацию о заклинаниях через get-методы

Взаимодействие с Enemy:

Создает: врагов через addEnemy()

Управляет: их поведением через processEnemyTurns()

Обрабатывает: нанесение урона врагам через damageEnemy()

Взаимодействие с Position:

Использует: для передачи координат в методах движения и заклинаний

Проверяет: валидность позиций через делегирование GameField

Взаимодействие с Cell:

Управляет: через делегирование GameField (setCellType(), getCellType())

Архитектура.

Архитектура построена на принципах ООП с четким разделением ответственности. GameEngine выступает главным координатором, управляя игровым процессом и связывая все компоненты системы. GameField отвечает за состояние игрового поля, включая клетки (Cell), позиции объектов (Position) и коллизии. Система персонажей включает Player для управления игроком и Enemy как базовый класс для врагов, с EnemyTower как наследником. Это демонстрирует принцип открытости/закрытости - система легко расширяется новыми типами врагов. Система заклинаний построена вокруг абстрактного класса Spell, конкретными реализациями (DirectDamageSpell, AreaDamageSpe ll, TrapSpell. SpellHand управляет коллекцией заклинаний игрока с ограниченным размером. Инкапсуляция защищает состояние объектов,

композиция и умные указатели обеспечивают безопасное управление памятью.

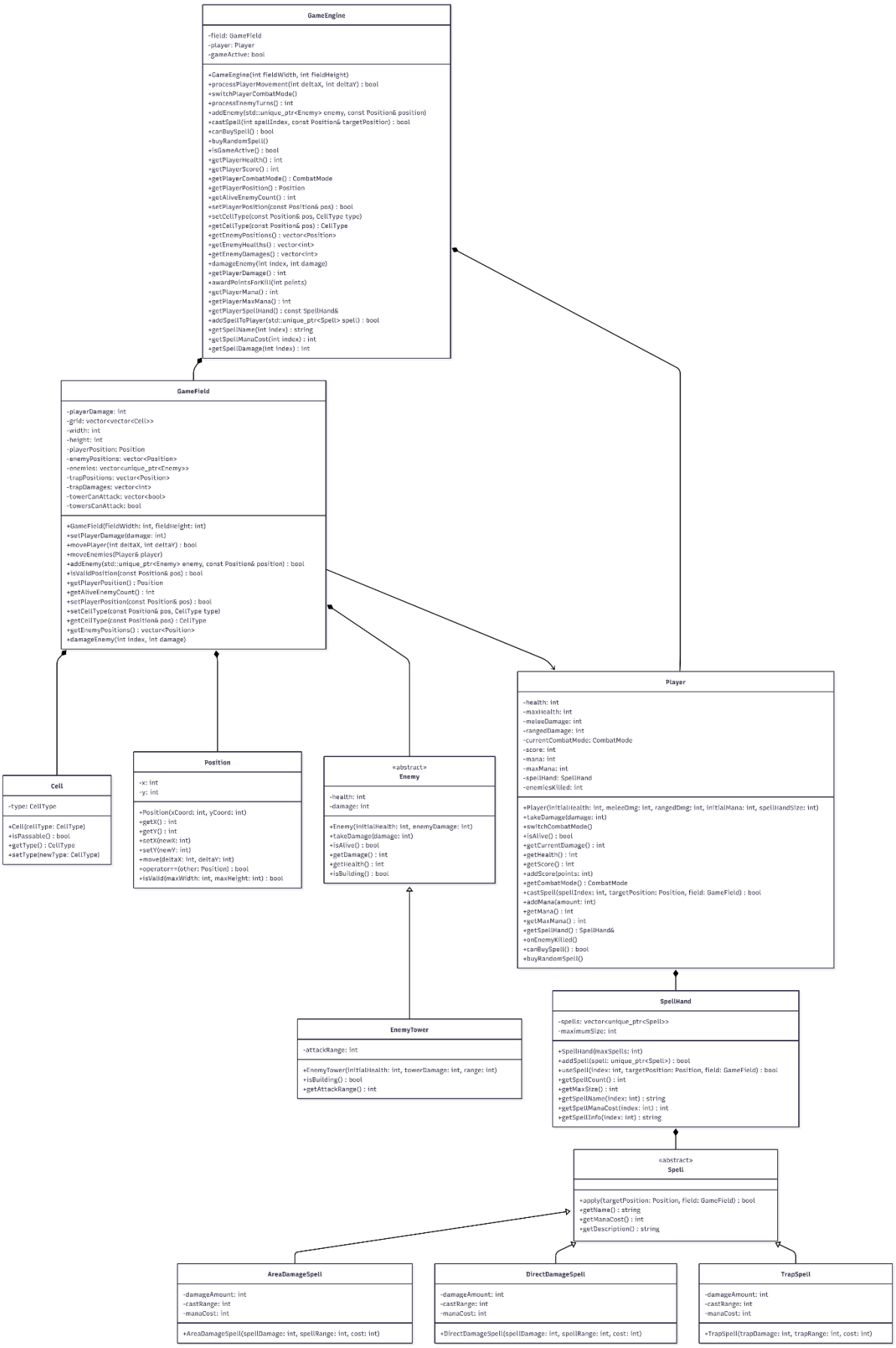

Рис. 1 UML-диаграмма классов.

Выводы.

В ходе лабораторной работы была разработана игра на языке C++, написанная с применением объектно-ориентированного программирования. В процессе выполнения были созданы и связаны между собой, каждый из которых выполняет определенную роль в архитектуре программы.

Разработанная структура демонстрирует принципы инкапсуляции, наследования и слабой связанности компонентов, что облегчает поддержку и масштабирование кода.