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

ОТЧЕТ

по лабораторной работе №3

по дисциплине «Объектно-ориентированное программирование»

Тема: Создание игры на C++

Студент гр. 4382 Исаев А. В.

Преподаватель Жангиров Т. Р.

Санкт-Петербург

2025

Цель работы.

Разработать игру на языке программирования C++.

Задание.

На 6/3/1 баллов:

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

a. Начало игры

b. Запуск уровня

c. Ход игрока. Ход, атака или применение заклинания.

d. Ход союзников - если имеются

e. Ход врагов

f. 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.

2. 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.

3. 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:

4. 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:

5. 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.

Примечания:

- Класс игры может знать о игровых сущностях, но не наоборот
- При работе с файлом используйте идиому RAII.
- Исключения должны обязательно обрабатываться, и программа не должна завершаться
- Исключения должны быть информативными (содержать информацию о том, что и где произошло), на разные виды исключительных ситуаций должны быть свои исключения

Выполнение работы.

Класс GameException и его производные

Назначение: Базовый класс для всех исключений игры. Производные классы предоставляют специализированные исключения для различных ситуаций.

Иерархия:

- GameException - базовый класс исключений игры
- SaveGameException - исключение при ошибке сохранения игры
- LoadGameException - исключение при ошибке загрузки игры
- FileNotFoundException - исключение при отсутствии файла
- CorruptedSaveException - исключение при поврежденном файле сохранения

Класс Position

Назначение: Представляет координаты на игровом поле. Обеспечивает базовые операции с позициями.

Приватные поля:

- int x - координата X
- int y - координата Y

Основные методы:

- getNeighbour() - возвращает соседнюю позицию в указанном направлении
- distanceTo() - вычисляет расстояние до другой позиции
- toString() / fromString() - сериализация позиции

Класс GameObject (абстрактный)

Назначение: Базовый класс для всех игровых объектов. Определяет общий интерфейс.

Абстрактные методы:

- getPosition() - возвращает позицию объекта
- isAlive() - проверяет, жив ли объект
- takeDamage() - наносит урон объекту
- serialize() - сериализует объект в строку

Класс Cell

Назначение: Представляет одну клетку игрового поля. Управляет состоянием клетки.

Приватные поля:

- Флаги занятости: is_occupied, has_player, has_enemy, has_tower, has_trap
- is_blocked - флаг блокировки клетки

Основные методы:

- Методы установки
  состояния: setPlayer(), setEnemy(), setTower(), setTrap()
- Методы проверки состояния: геттеры для всех флагов
- serialize() / deserialize() - сериализация состояния

Класс GameField

Назначение: Управляет игровым полем - сеткой клеток. Реализует "Правило пяти".

Приватные поля:

- vector<vector<Cell>> grid - двумерная сетка клеток
- int width, int height - размеры поля

Основные методы:

- Конструкторы и операторы (копирование, перемещение)
- Методы валидации: isValidPosition(), isCellFree(), isCellBlocked()
- Методы управления объектами: setPlayerPosition(), setEnemyPosition(), и т.д.
- serialize() / deserialize() - сериализация всего поля

Класс Enemy

Назначение: Представляет вражеского персонажа. Наследуется от GameObject.

Приватные поля:

- Position position - позиция врага
- int health - здоровье
- int damage - урон

- int level - уровень врага

Основные методы:

- moving() - перемещение врага
- serialize() / deserialize() - сериализация

Класс EnemyTower

Назначение: Представляет вражескую башню. Наследуется от GameObject.

Приватные поля:

- Position position - позиция башни
- int health - здоровье
- int damage - урон
- int attackRange - дальность атаки
- bool canAttack - флаг возможности атаки
- int level - уровень башни

Основные методы:

- attackIfPossible() - атакует игрока если возможно
- resetAttack() - сбрасывает флаг атаки
- serialize() / deserialize() - сериализация

Класс Spell (абстрактный) и его производные

Назначение: Базовый класс для заклинаний. Определяет интерфейс для магических способностей.

Производные классы:

- DirectDamageSpell - заклинание прямого урона
- AreaDamageSpell - заклинание площадного урона
- TrapSpell - заклинание установки ловушки

Абстрактные методы:

- canCast() - проверяет возможность применения
- cast() - применяет заклинание
- getName(), getDescription(), getRange() - информация о заклинании

Класс Trap

Назначение: Представляет ловушку на поле. Наследуется от GameObject.

Приватные поля:

- Position position - позиция ловушки
- int damage - урон ловушки
- bool active - активна ли ловушка

Основные методы:

- trigger() - активирует ловушку
- serialize() / deserialize() - сериализация

Класс SpellHand

Назначение: Управляет набором заклинаний игрока.

Приватные поля:

- vector<shared_ptr<Spell>> spells - список заклинаний
- int maxSize - максимальное количество заклинаний

Основные методы:

- addSpell() - добавляет заклинание
- addRandomSpell() - добавляет случайное заклинание
- castSpell() - применяет заклинание
- removeRandomSpells() - удаляет случайные заклинания
- serialize() / deserialize() - сериализация

Класс Player

Назначение: Представляет игрового персонажа. Наследуется от GameObject.

Приватные поля:

- Характеристики: max_health, health, score, melee_damage, ranged_damage
- AttackType current_attack - текущий тип атаки
- Position position - позиция игрока
- SpellHand spellHand - набор заклинаний
- int level - уровень игрока

Основные методы:

- switchWeapon() - переключает тип оружия
- addEnemyDefeated() - увеличивает счетчик побежденных врагов

- buySpell() - покупка заклинания
- prepareForNextLevel() - подготовка к следующему уровню
- serialize() / deserialize() - сериализация

Класс GameRenderer

Назначение: Отвечает за отрисовку игрового интерфейса.

Основные методы:

- renderHeader() - отрисовывает информационную панель
- renderField() - отрисовывает игровое поле
- renderGameOver() - отрисовывает экран завершения игры

Класс GameState

Назначение: Хранит и управляет состоянием игры.

Приватные поля:

- GameField field - игровое поле
- shared_ptr<Player> player - игрок
- vector<shared_ptr<GameObject>> gameObjects - игровые объекты
- GameStatus status - статус игры
- int currentLevel - текущий уровень

Основные методы:

- Геттеры и сеттеры для всех полей
- updateFieldState() - обновляет состояние поля
- getEnemies(), getTowers(), getTraps() - получает списки объектов

Класс LevelManager

Назначение: Управляет уровнями игры.

Основные методы:

- startLevel() - запускает уровень
- nextLevel() - переходит на следующий уровень
- initializeEnemies(), initializeTowers() - инициализация объектов уровня

Класс CombatManager

Назначение: Управляет боевой системой.

Основные методы:

- performRangedAttack() - выполняет дальнюю атаку
- processEnemyTurns() - обрабатывает ходы врагов
- movePlayer() - перемещает игрока

Класс ObjectManager

Назначение: Управляет игровыми объектами.

Основные методы:

- cleanupDeadObjects() - удаляет мертвые объекты
- checkTraps() - проверяет активацию ловушек

Класс GameStateManager

Назначение: Управляет общим состоянием игры.

Основные методы:

- updateGameState() - обновляет состояние игры

Класс SaveLoadManager

Назначение: Управляет сохранением и загрузкой игры.

Основные методы:

- saveGame() - сохраняет игру
- loadGame() - загружает игру

Класс InputHandler

Назначение: Обрабатывает пользовательский ввод.

Основные методы:

- getPlayerInput() - получает команду игрока
- getDirectionInput() - получает направление
- getSpellIndexInput() - получает индекс заклинания

Класс Game

Назначение: Основной класс, управляющий игровым процессом.

Приватные поля:

- Менеджеры: levelManager, combatManager, objectManager, stateManager, saveLoadManager
- GameState gameState - состояние игры
- GameRenderer renderer - отрисовщик

- InputHandler inputHandler - обработчик ввода

Основные методы:

- runGame() - запускает игровой цикл
- playerTurn() - обрабатывает ход игрока
- castSpell(), buySpell(), switchWeapon() - действия игрока

Класс MenuManager

Назначение: Управляет главным меню игры.

Основные методы:

- showMainMenu() - отображает главное меню
- handleMainMenu() - обрабатывает выбор в меню
- startNewGame(), loadGame() - запуск новой/загруженной игры

Перечисления (Enums)

- Direction - направления движения
- GameStatus - состояния игры
- AttackType - типы атаки

Вывод.

В ходе выполнения лабораторной работы была успешно разработана объектно-ориентированная пошаговая игра на языке C++. Основным достижением стало создание модульной и расширяемой архитектуры, демонстрирующей принципы объектно-ориентированного программирования.