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

ОТЧЕТ

по лабораторной работе №3

по дисциплине «Объектно-ориентированное программирование»

Студент гр. 4384 Кочкин Д. М.

Преподаватель Жангиров Т. Р.

Санкт-Петербург

2025

Цель работы.

Изучение принципов объектно-ориентированного проектирования и реализация игровой логики на языке C++. В ходе работы создаются классы для представления сущностей игрового поля (игрока, врагов, зданий, клеток и самого поля) и реализуются их взаимодействия.

Задание.

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.

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.

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.

Описание архитектуры программы.

Программа построена по принципам объектно-ориентированного программирования с чётким разделением ответственности между классами. Центральным элементом является класс Field, отвечающий за состояние игрового поля, размещение и обновление сущностей.

Основные классы:

• Entity — базовый абстрактный класс, определяющий общие свойства всех

сущностей (идентификатор, здоровье, урон).

• Player — класс игрока, наследник Entity. Реализует перемещение, режимы боя (ближний/дальний), взаимодействие с замедляющими клетками и врагами.

• Enemy — противник, движущийся к игроку. Если враг пытается шагнуть на клетку с игроком, он не перемещается, но наносит урон.

• Building — структура, периодически создающая врагов.

• Field — класс, содержащий сетку клеток, список сущностей и их координаты. Реализует механику игры: обновление, движение, спавн врагов и проверку конца игры.

• Cell — элемент поля, хранящий тип содержимого (пустая, непроходимая, замедляющая и т.д.).

Дополнительные классы:

Tower — специализированный тип здания, наследник Building. В отличие от обычных зданий не создает врагов, а атакует игрока в определенном радиусе, нанося уменьшенный урон.

Ally — дружественная сущность, наследник Entity. Автономно перемещается по полю и атакует врагов.

Trap — статическая сущность, наследник Entity. При попадании врага на клетку с ловушкой наносит урон и исчезает.

Система заклинаний:

ISpell — интерфейс, определяющий базовое поведение всех заклинаний (применение, клонирование, получение имени).

DirectDamageSpell — заклинание прямого урона по одной цели в указанном радиусе.

AreaDamageSpell — заклинание урона по области размером 2x2 клетки.

TrapSpell — заклинание, создающее ловушку на поле.

SummonSpell — заклинание призыва союзника.

BuffSpell — заклинание, временно усиливающее характеристики других заклинаний.

Вспомогательные компоненты:

Hand — класс, представляющий "руку" игрока с заклинаниями. Управляет коллекцией заклинаний, их добавлением и удалением.

CombatSystem — статический класс, отвечающий за боевую механику:

Обработка нанесения урона

Обработка смерти сущностей

Подсчет убийств и выдача наград игроку

Связи классов:

Иерархия сущностей (Entity):

Player, Enemy, Building, Ally, Trap наследуют базовый класс Entity

Tower наследует Building

Иерархия заклинаний (ISpell):

DirectDamageSpell, AreaDamageSpell, TrapSpell, SummonSpell, BuffSpell реализуют интерфейс ISpell

Композиция:

Game содержит Field

Field содержит коллекции Cell и Entity

Player содержит Hand

Hand содержит коллекцию ISpell

Зависимости:

CombatSystem использует Field и Player для реализации боевой механики

Spell классы используют Field для применения эффектов

Tower использует Field для атаки игрока

Система сохранения и загрузки

GameState — структура для сохранения полного состояния игры.

Поля:

currentLevel (int) — текущий номер уровня

playerHealth (int) — здоровье игрока

playerHealthMax (int) — максимальное здоровье

playerMeleeDamage (int) — урон в ближнем бою

playerRangedDamage (int) — урон в дальнем бою

playerScore (int) — очки игрока

fieldRows (int) — высота поля

fieldCols (int) — ширина поля

playerSpells (vector<string>) — названия заклинаний в руке

entityIds (vector<int>) — ID всех сущностей на поле

entityHealths (vector<int>) — здоровье каждой сущности

entityTypes (vector<int>) — типы сущностей (Enemy, Building, etc.)

entityPositionsRow (vector<int>) — строки позиций сущностей

entityPositionsCols (vector<int>) — столбцы позиций сущностей

Методы:

saveToFile(filename) — сохраняет состояние в бинарный файл

{static} loadFromFile(filename) — загружает состояние из файла

fileExists(filename) — проверяет наличие файла сохранения

Формат файла: бинарный с проверкой версии

1. Версия (size_t длина + строка)

2. Основные данные (int values)

3. Заклинания (count + array)

4. Сущности (count + array of entity data)

Сохранение: все сущности с позициями сохраняются для полного восстановления

Загрузка: восстанавливаются все сущности с оригинальными позициями и здоровьем

Система исключений:

Иерархия исключений построена для обработки различных ошибок:

GameException — базовый класс для всех игровых исключений.

- Наследует: std::exception
- Поля: message_ (string) — описание ошибки
- Методы: what() — возвращает сообщение об ошибке
- Использование: перехватывает все игровые ошибки в try-catch блоках

SaveGameException — исключение при ошибках сохранения.

- Наследует: GameException
- Конструктор: SaveGameException(filename, details)
- Форматирование сообщения: "[Operation] (file: filename): details"
- Выбрасывается: при ошибке открытия файла, записи данных, закрытия
- Обработка: выводит сообщение об ошибке, возврат в главное меню

LoadGameException — исключение при ошибках загрузки.

- Наследует: GameException

- Конструктор: LoadGameException(filename, details)

- Форматирование: аналогично SaveGameException

- Выбрасывается: при несовместимости версии, ошибке чтения данных

- Обработка: попытка повторной загрузки или возврат в меню

FileAccessException — исключение при проблемах доступа к файлам.

- Наследует: GameException

- Конструктор: FileAccessException(filename, operation)

- Выбрасывается: файл не существует, невозможно открыть

- Обработка: логирование и сообщение пользователю

InvalidGameStateException — исключение при некорректном состоянии игры.

- Наследует: GameException

- Конструктор: InvalidGameStateException(details)

- Выбрасывается: игрок или поле не инициализированы, неверные параметры

- Обработка: возврат в главное меню и перезагрузка

LevelException — исключение при ошибках уровня.

- Наследует: GameException

- Конструктор: LevelException(level, details)

- Выбрасывается: неверные параметры уровня, ошибка инициализации

- Обработка: загрузка стандартных параметров уровня

Обработка исключений:

- GameManager::run() содержит try-catch для всех типов исключений

- GameException перехватывается специально для игровых ошибок

- std::exception перехватывается для системных ошибок

- При ошибке выводится сообщение и предлагается вернуться в меню

Система прогрессии:

LevelManager — управляет уровнями и их параметрами.

- Поля: currentLevel_ (int), maxLevels_ (int) = 5

- Методы:

  getCurrentLevelParams() — возвращает параметры текущего уровня

  nextLevel() — переход на следующий уровень

  LevelParams — структура с параметрами уровня:

  fieldRows, fieldCols — размер поля (увеличивается)

  enemyCount — количество врагов (увеличивается)

  buildingCount — количество зданий

  enemyHealth — здоровье врагов (+20% на каждом уровне)

enemyDamage — урон врагов (+10% на каждом уровне)

buildingHealth — здоровье зданий

- Масштабирование сложности:

Level 1: 12x12 поле, 5 врагов, 2 здания

Level 2: 14x14 поле, 6 врагов, 2 здания (враги +20% HP, +10% DMG)

Level 3: 16x16 поле, 7 врагов, 2 здания (враги еще +20% HP, +10% DMG)

... и так далее до Level 5

UpgradeSystem — система прокачки между уровнями.

- Методы: showUpgradeMenu(player) — показывает меню из 6 опций

- Возможные улучшения:

1. Увеличить здоровье +15

2. Увеличить урон в ближнем бою +2

3. Увеличить урон в дальнем бою +2

4. Добавить случайное заклинание

5. Уменьшить перезарядку дальнего боя (вычитается из счетчика)

6. Увеличить радиус заклинаний +1

UML-диаграмма классов.

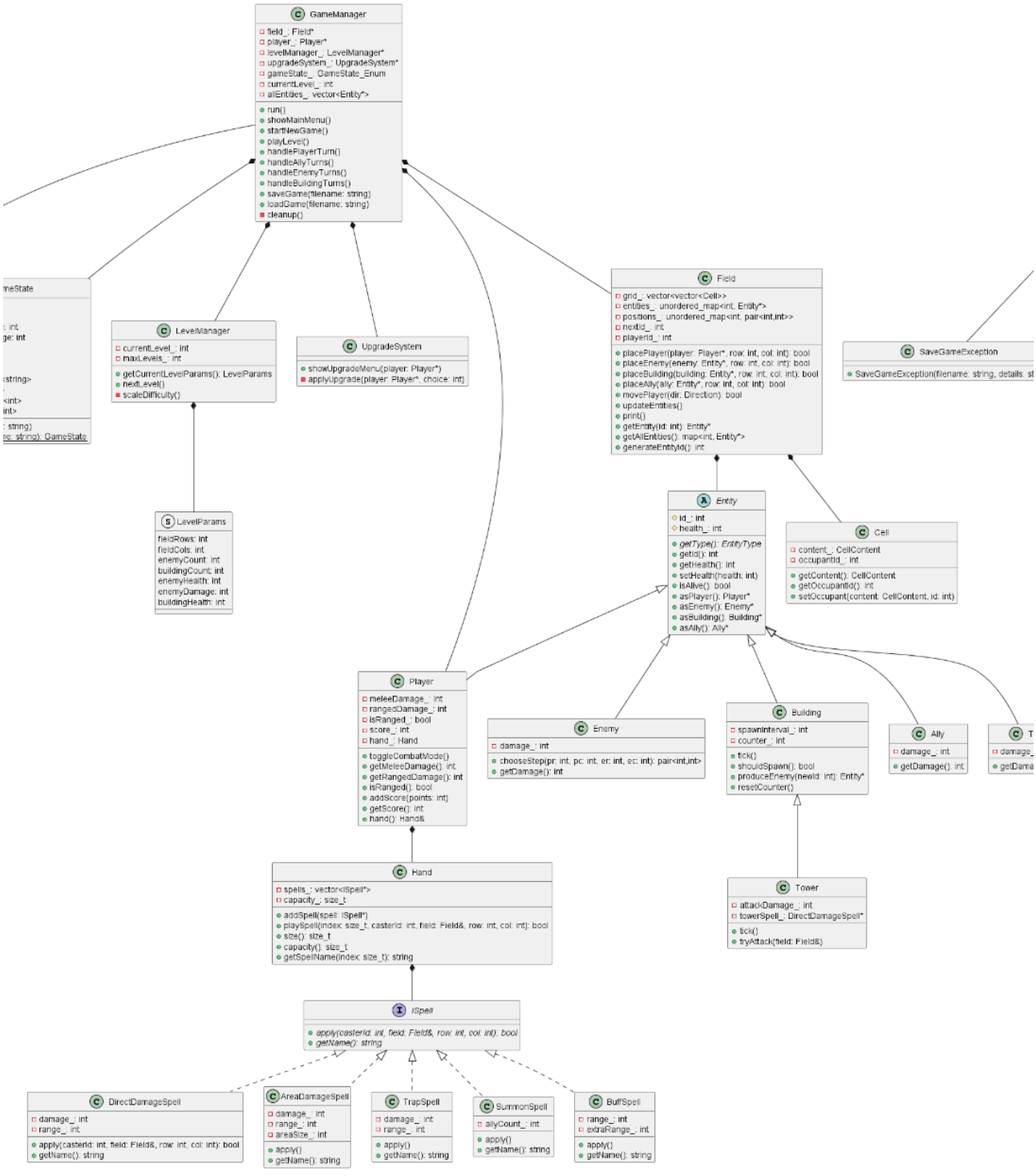

Рисунок 1 - UML-диаграмма

Проверка работы программы.

Была проведена проверка работы игры. Игрок может перемещаться по полю, сталкиваться с врагами и зданиями, а также переключаться между ближним и дальним боем. При переходе на клетку с врагом он атакует противника, но теряет возможность хода на следующий раунд. При переходе на замедляющую клетку — пропускает ход. Враги корректно двигаются к игроку и атакуют при попытке занять его клетку. Создана механика заклинаний. Реализована возможность возможность добавить союзника, атакующего врагов. Добавлен класс башни, которая применяет заклинания и наносит урон игроку, который стоит рядом с ней.

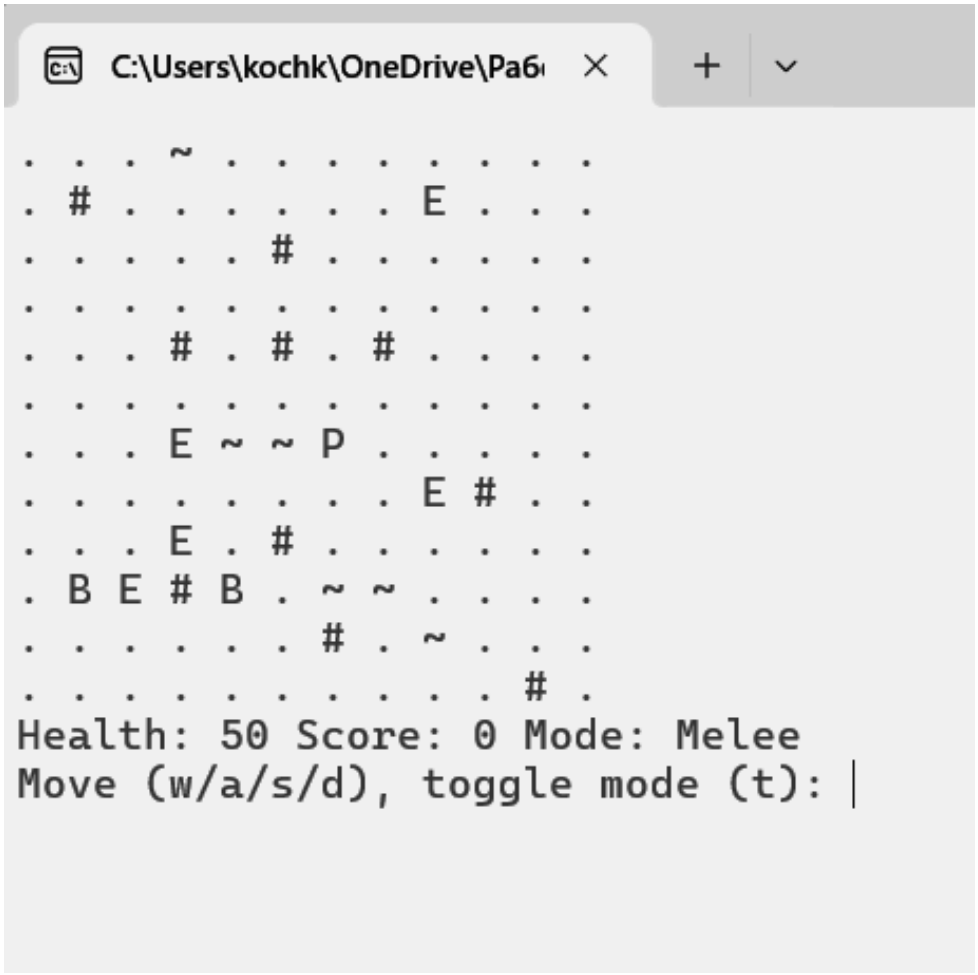

Рисунок 2 - Игровое поле

Выводы.

В результате выполнения лабораторной работы были изучены принципы ООП: наследование, инкапсуляция и полиморфизм. Разработана архитектура игры с несколькими типами сущностей и взаимодействиями между ними. Программа успешно проходит тестирование и демонстрирует корректное поведение объектов на игровом поле.