МИНОБРНАУКИ РОССИИ

Санкт-Петербургский государственный

электротехнический университет

«ЛЭТИ» им. В.И. Ульянова (Ленина)

Кафедра МО ЭВМ

отчет

по лабораторной работе №1

по дисциплине «Объектно-Ориентированное Программирование»

Студент гр. 4384 _____ Боков М.О.

Преподаватель _____ Жангиров Т.Р.

Санкт-Петербург

2025

Цель работы

Создать модель игры в которой создается игровое поле, определен игрок, враг, непроходимые поля.

Задание

1. Создать класс игрока, который должен хранить информацию об игроке (его жизни, урон, очки, и т.д. - студент сам определяет необходимые для работы характеристики). Объект класса игрока должен перемещаться по карте. Если у игрока кончаются жизни, то происходит конец игры.

2. Создать класс врага, который хранит параметры жизней и урона. Объектами класса врага управляет компьютер. При перемещении, если враг пытается перейти на клетку с игроком, то перемещение не происходит, и игроку наносится урон.

3. Создать класс квадратного/прямоугольного игрового поля, по которому перемещаются игрок и враги. Игровое поле не должно быть меньше 10 на 10 клеток, и не больше 25 на 25 клеток. Размеры поля задаются через конструктор. Рекомендуется для хранения информации об отдельных клетках поля создать отдельный класс.
Реализовать конструкторы перемещения и копирования для поля, а также соответствующие операторы присваивания с копированием и перемещением (должна происходить глубокая копия).

4. Реализовать непроходимые клетки на поле. При попытке врагов или игрока перейти на такую клетку, перемещение не происходит. Заполнение поля непроходимыми клетками происходит в момент создания поля.

5. Добавить возможность для игрока переключаться на ближний или дальний бой с изменением значения наносимого урона. Такое переключение требует один ход.

Выполнение работы

Класс Player:

НАЗНАЧЕНИЕ: Представляет управляемого игроком персонажа с системой здоровья,

урона и очков. Реализует механику переключения режимов боя.

ПРИЧИНЫ РЕАЛИЗАЦИИ:

1. Инкапсуляция состояния игрока (здоровье, очки, режим боя)

2. Разделение ответственности - игрок управляет только своими характеристиками

3. Легкость расширения (можно добавить новые характеристики)

СВЯЗИ С ДРУГИМИ КЛАССАМИ:

- GameEngine: использует Player для управления состоянием игрока

- Enemy: наносит урон Player через метод

- GameField: проверяет состояние игрока

Поля:

health: int - текущее количество здоровья игрока. Уменьшается при получении урона, не может быть отрицательным.

meleeDamage: const int - базовый урон в режиме ближнего боя. Постоянное значение, устанавливается при создании.

rangedDamage: const int - базовый урон в режиме дальнего боя. Постоянное значение, устанавливается при создании.

currentCombatMode: CombatMode - текущий активный режим боя. Может быть MELEE или RANGED.

score: int - количество набранных игроком очков. Увеличивается при выполнении игровых действий.

Методы:

Player(int initialHealth, int meleeDmg, int rangedDmg) - конструктор класса. Создает объект игрока с заданными характеристиками.

takeDamage(int damage) - метод получения урона. Уменьшает здоровье игрока на указанное количество.

switchCombatMode() - метод переключения режима боя. Циклически меняет режим между MELEE и RANGED.

isAlive() const - метод проверки жизненного состояния. Возвращает true если игрок жив (health > 0).

getCurrentDamage() const - метод получения текущего значения урона. Возвращает урон в зависимости от активного режима боя.

getHealth() const - метод получения текущего здоровья. Возвращает текущее значение переменной health.

getScore() const - метод получения текущего счета. Возвращает текущее значение переменной score.

addScore(int points) - метод добавления очков. Увеличивает счет игрока на указанное количество.

getCombatMode() const - метод получения текущего режима боя. Возвращает значение currentCombatMode.

КЛАСС: Position

НАЗНАЧЕНИЕ: представляет координаты на двумерной игровой сетке, хранение и манипуляция координатами X и Y.

ПРИЧИНЫ РЕАЛИЗАЦИИ:

1. Инкапсуляция координат - централизованное управление позициями

2. Повторное использование - единый интерфейс для работы с координатами во всей системе

3. Безопасность - проверка валидности позиций перед использованием

СВЯЗИ С ДРУГИМИ КЛАССАМИ:

- GameField: использует Position для хранения позиций игрока, врагов и клеток

- Player: имеет Position для отслеживания местоположения на поле

- Enemy: имеет Position для отслеживания местоположения на поле

- Cell: используется в сочетании с Position для доступа к конкретным клеткам поля

Поля:

x: int - горизонтальная координата на игровом поле.

y: int - вертикальная координата на игровом поле

Методы:

Position(int xCoord, int yCoord) - конструктор класса. Создает объект позиции с заданными координатами

getX() const: int - метод получения координаты X. Возвращает значение переменной x.

getY() const: int - метод получения координаты Y. Возвращает значение переменной y.

setX(int newX): void - метод установки координаты X. Присваивает переменной x новое значение.

setY(int newY): void - метод установки координаты Y. Присваивает переменной y новое значение.

move(int deltaX, int deltaY): void - метод перемещения позиции. Изменяет координаты на заданные смещения.

operator==(const Position& other) const: bool - оператор сравнения на равенство. Возвращает true если обе координаты равны.

isValid(int maxWidth, int maxHeight) const: bool - метод проверки валидности позиции. Возвращает true если позиция в пределах поля.

КЛАСС: Cell

НАЗНАЧЕНИЕ: Представляет одну клетку игрового поля с определенным типом и свойствами, хранение типа клетки и определение ее свойств (проходимость)

ПРИЧИНЫ РЕАЛИЗАЦИИ:

1. Инкапсуляция свойств клетки - централизованное управление типами клеток

2. Гибкость - легко добавить новые типы клеток без изменения основной логики

3. Простота - минимальный интерфейс для выполнения необходимых функций

СВЯЗИ С ДРУГИМИ КЛАССАМИ:

- GameField: содержит двумерный массив Cell объектов для представления игрового поля

- Position: используется для идентификации конкретной клетки на поле

- Player/Enemy: взаимодействуют с клетками через GameField при перемещении

Методы:

Cell(CellType cellType) – конструктор, инициализирует переменную type заданным значением cellType

isPassable() const: bool - метод проверки проходимости клетки. Возвращает true если клетка пустая.

getType() const: CellType - метод получения типа клетки. Возвращает значение переменной type.

Класс GameEngine

НАЗНАЧЕНИЕ: Центральный координирующий класс, управляющий всем игровым процессом, координация взаимодействия между игроком, врагами и полем, управление игровой логикой

Причины реализации:

1. Простой интерфейс для сложной подсистемы

2. Разделение ответственности - отделяет игровую логику от представления

3. Централизация управления - все игровые события обрабатываются в одном месте

Связь с другими классами:

- GameField: управляет игровым полем и позициями объектов

- Player: управляет состоянием игрока (здоровье, урон, очки)

- Enemy: управляет врагами через GameField

- Position/Cell: используются для навигации и взаимодействия с полем

Поля:

field: GameField - игровое поле, содержащее клетки, позиции объектов и врагов

player: Player - объект игрока с характеристиками и состоянием

gameActive: bool - флаг активности игры (true когда игра продолжается)

Методы:

GameEngine(int fieldWidth, int fieldHeight) - конструктор, создает игровой движок с полем заданного размера

processPlayerMovement(int deltaX, int deltaY): bool - обрабатывает движение игрока, возвращает успешность перемещения

switchPlayerCombatMode(): void - переключает режим боя игрока между ближним и дальним

processEnemyTurns(): int - обрабатывает ход всех врагов, возвращает полученный игроком урон

addEnemy(unique_ptr<Enemy>, Position): void - добавляет врага на указанную позицию поля

isGameActive(): bool - проверяет активна ли игра (игрок жив и игра не завершена)

getPlayerHealth(): int - возвращает текущее здоровье игрока

getPlayerScore(): int - возвращает текущий счет игрока

getPlayerCombatMode(): CombatMode - возвращает текущий режим боя

getPlayerPosition(): Position - возвращает позицию игрока на поле

getAliveEnemyCount(): int - возвращает количество живых врагов

setPlayerPosition(Position): bool - устанавливает позицию игрока

setCellType(Position, CellType): void - устанавливает тип клетки

getCellType(Position): CellType - возвращает тип клетки

getEnemyPositions(): vector<Position> - возвращает позиции всех врагов

getEnemyHealths(): vector<int> - возвращает здоровье всех врагов

getEnemyDamages(): vector<int> - возвращает урон всех врагов

damageEnemy(int, int): void - наносит урон врагу по индексу

SetPlayerDamage(): int - возвращает текущий урон игрока

awardPointsForKill(int): void - начисляет очки за убийство врага

Класс GameField

Назначение: Представляет игровое поле, содержащее клетки, позиции игрока и врагов. Управляет пространственным расположением объектов, проверкой столкновений и перемещением.

Причины реализации:

1. Инкапсуляция игрового поля - все операции с полем сосредоточены в одном классе

2. Управление памятью - умные указатели для врагов обеспечивают безопасное управление памятью

3. Производительность - быстрый доступ к клеткам по координатам

4. Гибкость - возможность легко модифицировать структуру поля

Связи с другими классами:

GameField Cell:

GameField содержит двумерный массив Cell объектов

Каждая клетка определяет можно ли через нее перемещаться

Связь через композицию - поле состоит из клеток

GameField Position:

GameField использует Position для идентификации местоположения

Все объекты на поле имеют свои позиции

Связь через использование - Position используется как координата

GameField Player:

GameField хранит позицию игрока

GameField проверяет возможность перемещения игрока

Связь через ассоциацию - GameField знает о Player только через позицию

GameField Enemy:

GameField хранит врагов и их позиции

GameField управляет перемещением врагов и их атаками

Связь через агрегацию - GameField содержит врагов

Поля:

grid: vector<vector<Cell>> - двумерный массив клеток, представляющий поле.

width: int - ширина поля.

height: int - высота поля.

playerPosition: Position - текущая позиция игрока на поле.

enemyPositions: vector<Position> - вектор позиций врагов.

enemies: vector<unique_ptr<Enemy>> - вектор умных указателей на врагов.

playerDamage: int – хранит урон игрока

Методы:

GameField(int fieldWidth, int fieldHeight) - конструктор, создающий поле заданного размера и инициализирующий его.

Конструкторы копирования и перемещения, операторы присваивания (правило пяти) для корректного управления ресурсами.

initializeField() - инициализирует поле, заполняя его клетками (случайным образом расставляя стены).

movePlayer(int deltaX, int deltaY) - перемещает игрока на заданное смещение, если это возможно.

moveEnemies(Player& player) - перемещает всех живых врагов в случайных направлениях, а если враг натыкается на игрока, то атакует его.

addEnemy(unique_ptr<Enemy> enemy, const Position& position) - добавляет врага на поле в указанную позицию.

isValidPosition(const Position& pos) const - проверяет, находится ли позиция в пределах поля.

isCellPassable(const Position& pos) const - проверяет, можно ли пройти через клетку в данной позиции.

getPlayerPosition() const - возвращает текущую позицию игрока.

getAliveEnemyCount() const - возвращает количество живых врагов.

hasEnemyAtPosition(const Position& pos) const - проверяет, есть ли враг в указанной позиции.

setPlayerPosition(const Position& pos) - устанавливает позицию игрока (для тестирования).

setCellType(const Position& pos, CellType type) - устанавливает тип клетки в указанной позиции (для тестирования).

getCellType(const Position& pos) const - возвращает тип клетки в указанной позиции (для тестирования).

getEnemyPositions() const - возвращает вектор позиций врагов (для тестирования).

getEnemyHealths() const - возвращает вектор здоровья врагов (для тестирования).

getEnemyDamages() const - возвращает вектор урона врагов (для тестирования).

damageEnemy(int index, int damage) - наносит урон врагу по индексу (для тестирования).

copyFrom(const GameField& other) - вспомогательный метод для копирования.

moveFrom(GameField&& other) - вспомогательный метод для перемещения.

setPlayerDamage(int damage): void - устанавливает урон игрока

getCell(const Position& pos): Cell& - получает ссылку на клетку

Класс Enemy:

Назначение:

Класс Enemy представляет врага в игре. Он хранит параметры здоровья и урона, а также методы для взаимодействия с врагом. Причины реализации:

1. Простота и эффективность - минимальный набор методов для выполнения необходимых функций

2. Неизменяемость урона - урон задается при создании и не меняется, что обеспечивает баланс

3. Защита данных - здоровье защищено от отрицательных значений

4. Легкость использования - интуитивно понятный интерфейс для взаимодействия

Связи с другими классами:

Enemy GameField:

GameField содержит коллекцию Enemy объектов

GameField управляет перемещением врагов и их атаками на игрока

Связь через агрегацию - GameField содержит врагов через умные указатели

Enemy Player:

Enemy наносит урон Player при атаке через метод takeDamage()

Player может атаковать Enemy через систему боя

Связь через взаимодействие - Enemy влияет на состояние Player

Enemy GameEngine:

GameEngine координирует действия врагов через GameField

GameEngine предоставляет методы для нанесения урона врагам

Связь через делегирование - GameEngine работает с врагами через GameField

Поля:

health: int - текущее здоровье врага, уменьшается при получении урона

damage: const int - базовый урон врага, константное значение устанавливается при создании

Методы:

Enemy(int initialHealth, int enemyDamage) - Конструктор создает врага с заданным здоровьем и уроном

takeDamage(int damageTaken): void - уменьшает здоровье на указанное значение, защищает от отрицательных значений

isAlive(): bool - проверяет жив ли враг

getDamage(): int - возвращает урон врага

getHealth(): int - возвращает текущее здоровье врага

Uml-диаграмма классов(рис. 1)

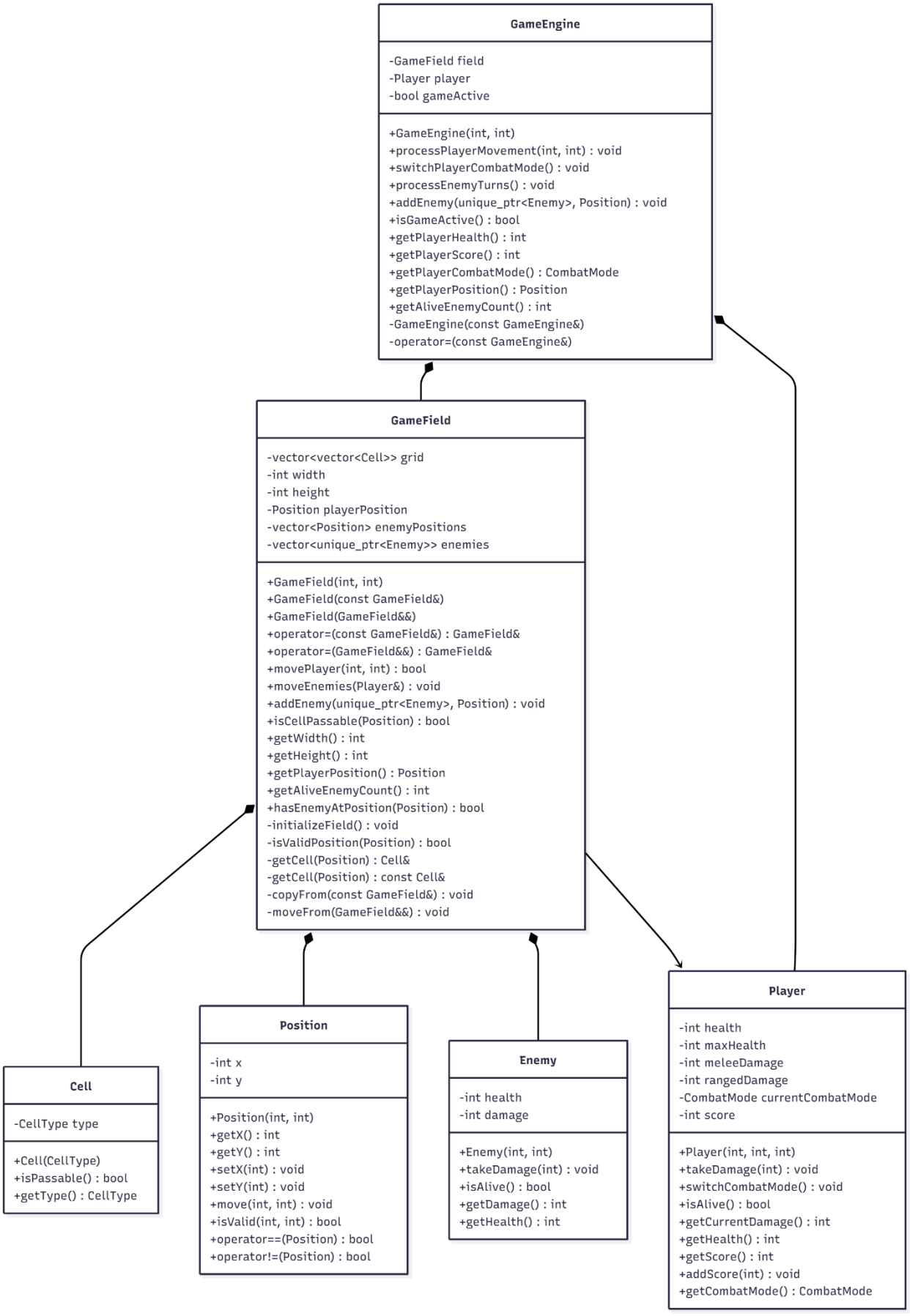

Рисунок 1 – UML-диаграмма

Вывод: была создана модель игры в которой создается игровое поле, определен игрок, враг, непроходимые поля. Реализованы классы: Player, Enemy, Cell, GameEngine, GameField, Position, которые обрабатывают разные сценарии.