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»

Тема: Связывание классов

Студентка гр. 3383 _____ Земерова С.Н.

Преподаватель _____ Жангиров Т.Р.

Санкт-Петербург

2025

Оглавление

1. Цель работы

Связать разработанные ранее сущности (ЛР-1 — поле/корабли, ЛР-2 — способности) единым управляемым игровым циклом в классе Game, обеспечить корректную сериализацию/десериализацию состояния и подготовить архитектуру к независимой замене ввода/вывода (ЛР-4).

2. Задание

a) Создать класс игры, который реализует следующий игровой цикл:
   i. Начало игры
   ii. Раунд, в котором чередуются ходы пользователя и компьютерного врага. В свой ход пользователь может применить способность и выполняет атаку. Компьютерный враг только наносит атаку.
   iii. В случае проигрыша пользователь начинает новую игру
   iv. В случае победы в раунде, начинается следующий раунд, причем состояние поля и способностей пользователя переносятся.

Класс игры должен содержать методы управления игрой, начало новой игры, выполнить ход, и т.д., чтобы в следующей лаб. работе можно было выполнять управление исходя из ввода игрока.

b) Реализовать класс состояния игры, и переопределить операторы ввода и вывода в поток для состояния игры. Реализовать сохранение и загрузку игры. Сохраняться и загружаться можно в любой момент, когда у пользователя приоритет в игре. Должна быть возможность загружать сохранение после перезапуска всей программы.

3. Модель предметной области и инварианты

3.1. Сущности

1) Клетка (Cell) - базовый элемент игрового поля с состояниями:
   - UNKNOWN - непроверенная клетка
   - EMPTY - пустая клетка (вода)
   - SHIP - клетка с кораблем
2) Сегмент корабля - часть корабля с состояниями:
   - INTACT - неповрежденный
   - DAMAGED - поврежденный
   - DESTROYED - уничтоженный
3) Корабль (Ship) - игровая единица с характеристиками:
   - Длина: 1-4 клетки
   - Ориентация: HORIZONTAL/VERTICAL
   - Массив сегментов
4) Игровое поле (PlayingField) - прямоугольная сетка N×M для размещения кораблей
5) Менеджер кораблей (ShipManager) - управление флотом, учет состояния кораблей
6) Способности (Ability) - специальные умения игрока:
   - DoubleDamage - двойной урон
   - Scanner - сканирование области
   - Shelling – обстрел
7) Менеджер способностей (AbilityManager) - управление очередью способностей
8) Состояние игры (GameState) - сериализуемое представление всей игровой сессии
9) Игра (Game) - основной контроллер, управляющий игровым процессом
10) Игрок (Player) - участник игры (человек или ИИ)
11) Настройки игры (GameSettings) - конфигурационные параметры

3.2. Инварианты

1) Целостность игрового поля:
- Поля не могут быть nullptr
- Размеры полей согласованы
- Курсор всегда находится в пределах: x ∈ [0, width-1], y ∈ [0, height-1]

2) Корректность состояния игры**:**
- GameStatus всегда содержит допустимое значение из перечисления
- Статус и очередь хода согласованы (PLAYER_TURN ↔ ход игрока)

3) Размещение кораблей:
- Корабли не пересекаются
- Корабли не соприкасаются (включая диагонали)
- Все корабли полностью помещаются в пределах поля

4) Повреждения кораблей**:**
- Состояния сегментов изменяются строго: INTACT → DAMAGED → DESTROYED
- Корабль считается уничтоженным только когда все его сегменты в состоянии DESTROYED

5) Механика выстрелов:
- После выстрела клетка не может остаться в состоянии UNKNOWN
- Результат выстрела синхронизирован между полем и менеджером кораблей

6) Система способностей**:**
- Очередь способностей может быть пустой
- Попытка использовать способность из пустой очереди вызывает корректное исключение

7) Сохранение/загрузка:
- Сохранение возможно только в статусах без диалоговых окон и вне хода ии-игрока

- Загрузка возможна только в статусах: GAME_OVER, PAUSED, PLACING_SHIPS, PLAYER_WON, ENEMY_WON, WAITING_NEW_ROUND
- Загруженное состояние проходит валидацию на соответствие инвариантам

8) Статистика игры**:**

- Количество попаданий ≤ количество выстрелов
- Точность вычисляется как (попадания / выстрелы) × 100%
- Количество уничтоженных кораблей ≤ общее количество кораблей

4. Детальное описание классов

4.1. Класс настройки игры GameSettings

Назначение

Класс GameSettings отвечает за хранение, изменение и интерактивную конфигурацию всех параметров игры перед её запуском.
Он обеспечивает выбор интерфейса, размера поля, режима расстановки, набора кораблей, имени игрока и графических параметров.

Перечисления

InterfaceType

Определяет тип пользовательского интерфейса:

- CONSOLE — текстовый интерфейс в консоли.
- GUI — графический интерфейс.

FleetBuildMode

Определяет режим формирования флота:

- STANDARD — стандартный набор кораблей: *4, 3, 3, 2, 2, 2, 1, 1, 1, 1*
- CUSTOM — пользовательская спецификация кораблей.

PlacementMode

Определяет способ расстановки кораблей:

- AUTO — автоматическая постановка.
- MANUAL — ручная расстановка.

Структура класса GameSettings

Приватные поля

Поле	Тип	Назначение
interface_type_	InterfaceType	Тип интерфейса (GUI по умолчанию)
field_size_	int	Размер игрового поля (NxN), по умолчанию 10
cell_size_	int	Размер клетки поля в пикселях (GUI), по умолчанию 40
player_name_	std::string	Имя игрока (по умолчанию "Player")
fleet_mode_	FleetBuildMode	Режим формирования флота
placement_mode_	PlacementMode	Режим расстановки кораблей
fleet_spec_	std::vector	Текущая спецификация флота (используемая игрой)
temp_fleet_spec_	std::vector	Временная спецификация флота
temp_field_size_	int	Временный размер поля

Конструктор GameSettings()

Конструктор инициализирует объект следующими значениями:

- **i**nterface_type_ = InterfaceType::GUI
- field_size_ = 10
- cell_size_ = 40
- player_name_ = "Player"
- fleet_mode_ = FleetBuildMode::STANDARD
- placement_mode_ = PlacementMode::AUTO
- fleet_spec_ = {4,3,3,2,2,2,1,1,1,1} — стандартный набор кораблей
- temp_fleet_spec_ = {}
- temp_field_size_ = 10

Основные публичные методы

ShowSettingsDialog()

Отображает интерактивный диалог в консоли. Позволяет настроить:

- тип интерфейса;
- размер клетки (при GUI);
- имя игрока (с обработкой UTF-8, ограничением по длине и проверкой корректности);
- размер поля;
- режим формирования флота (STANDARD/CUSTOM);
- пользовательские размеры кораблей;
- режим расстановки (AUTO/MANUAL).

Метод вызывает внутреннюю процедуру:

ConfigureGameSettings()

Пошаговая настройка:

1. Ввод размера поля (10–14).

2. Выбор FleetBuildMode.
3. Ввод списка кораблей (если CUSTOM).
4. Выбор PlacementMode.
5. Подтверждение.

Методы управления флотом

AddShipSize(int size)

Добавляет корабль во временную спецификацию temp_fleet_spec_ (до 16 кораблей).

ClearTempFleetSpec()

Очищает временный список кораблей.

ApplyFleetSpec()

Применяет temp_fleet_spec_ как текущий fleet_spec_.

set_fleet_mode(bool custom = false)

При true → устанавливает CUSTOM

При false → устанавливает STANDARD и сбрасывает fleet_spec_ в стандартный набор кораблей.

Методы управления игровым полем

ApplyFieldSize()

Применяет временный размер поля (temp_field_size_) как постоянный.

set_temp_field_size(int size)

Устанавливает временный размер поля

ResetFieldAndShipSize()

Сбрасывает поле к размеру 10×10 и активирует стандартный набор кораблей.

Вспомогательные функции

readIntOrDefaultWithWarn(default, min, max)

Безопасно считывает число с проверкой:

- диапазона,
- ошибок ввода,
- очисткой буфера.

parseFleetLine(const std::string& line)

Разбирает строку вида "4,3,3,2":

- разделитель — запятая,
- игнорирует пустые токены,
- принимает числа **1–4**,
- возвращает std::vector<int>.

4.2. Класс состояния игры GameState

Назначение

GameState представляет собой сериализуемое состояние игрового процесса. Класс отвечает за:

- сохранение и загрузку игровых данных;
- хранение параметров текущей игровой сессии;
- полное восстановление игрового раунда;
- управление состояниями, статистикой, курсором и менеджером способностей.

GameState является центральным хранилищем информации и используется как при консольном, так и при графическом интерфейсе.

Структуры данных

Структура PlayerStats

Хранит статистику игрока в пределах одного раунда.

Поля:

- name — имя игрока;
- hits — число успешных попаданий;
- shots — общее количество выстрелов;
- accuracy — точность стрельбы (hits/shots);
- destroyed — количество уничтоженных кораблей противника;
- remaining — количество оставшихся собственных кораблей.

Структура поддерживает потоковые операторы << и >> (см. 4.2.4).

Структура TotalPlayerStats

Содержит накопленную статистику за все раунды:

- total_hits — суммарные попадания,
- total_shots — суммарные выстрелы,
- accuracy — суммарная точность,
- rounds — количество сыгранных раундов,
- count_won — количество выигранных раундов.

Используется для глобального анализа прогресса игрока.

Перечисление GameStatus

Определяет текущее состояние игрового процесса.

Основные состояния:

- PLACING_SHIPS — расстановка кораблей;
- PLAYER_TURN — ход игрока;
- ENEMY_TURN — ход компьютера;
- PLAYER_WON, ENEMY_WON — завершение раунда победой одной из сторон;
- PAUSED — игра поставлена на паузу;
- GAME_OVER — игровой сеанс завершён.

Дополнительные состояния для последующей визуализации (GUI/консоль):

- SET_FIELD, SET_SIZES, SET_PLACEMENT_MODE, SETTING_SHIPS — этапы настройки;
- ASK_EXIT, ASK_SAVE — подтверждения;
- WAITING_NEXT_ROUND — ожидание следующего раунда;

- SELECT_SAVE_SLOT, SELECT_LOAD_SLOT — выбор сейва.

Приватные поля класса

Статистика

- player_stats_ — статистика игрока;
- enemy_stats_ — статистика противника;
- total_player_stats_ — суммарная статистика игрока;
- total_enemy_stats_ — суммарная статистика противника.

Состояние игры

- status_ — текущий статус (по умолчанию GAME_OVER);
- round_result_ — итог раунда: -1 (не завершён), 0 (ничья), 1/-1;
- is_player_turn_ — флаг текущего хода;
- round_number_ — номер текущего раунда.

Положение курсора

- cursor_x_, cursor_y_ — координаты указателя в интерфейсе.

Сохранения

- save_date_ — дата последнего сохранения;
- save_directory_ = "saves/" — директория для файлов сохранений.

Игровые поля и системы

- player_field_state_ — игровое поле игрока;
- enemy_field_state_ — поле компьютера;
- player_abilities_ — менеджер способностей игрока (инициализируется ссылками на оба поля);

- ship_manager_ — менеджер кораблей для переходов между раундами.

Конструктор и базовая инициализация

GameState()

При создании объекта выполняется:

1. Инициализация всех полей

- Игровой статус = GAME_OVER;
- Статистика игроков создаётся с именами «Игрок» и «Противник»;
- Поля player_field_state_ и enemy_field_state_ создаются размером 10×10;
- Менеджер способностей получает ссылки на созданные поля.

2. Автоматическое создание папки сохранений

std::filesystem::create_directories(save_directory_);

Гарантируется, что директория saves/ существует при первом запуске.

Потоковые операторы сериализации

PlayerStats: operator<<

1. Имя записывается в кавычках через std::quoted.
2. Далее последовательно записываются:
    - hits, shots, accuracy, destroyed, remaining.
3. В конце — \n.

PlayerStats: operator>>

1. Имя читается через std::quoted.

2. Затем — все числовые поля.

TotalPlayerStats: аналогичные операторы<< >>

GameState: operator<<

Записывает:

1. Дату сохранения (quoted)
2. Статус, результат раунда, чей ход
3. Номер раунда
4. Положение курсаора
5. Статистику игроков
6. Суммарную статистику
7. Менеджер кораблей ShipManager
8. Размеры обоих полей
9. Полные состояния полей (PlayingField)
10. Состояние способностей (AbilityManager)

Порядок строго фиксирован.

GameState: operator>>

Пошагово восстанавливает всё состояние:

1. Читает дату, статус, результаты, координаты.
2. Читает всю статистику.
3. Восстанавливает ShipManager.
4. Читает размеры полей → создаёт новые объекты.
5. Загружает состояние полей.
6. Загружает AbilityManager.

Все данные полностью заменяют текущие.

Управление сохранением и загрузкой

SaveGame(const std::string& filename)

Последовательность:

1. Формируется путь: "saves/filename.save".
2. Создаётся FileHandler с флагом std::ios::trunc.
3. Генерируется временная метка:
    - std::time, std::localtime, std::put_time.
4. Выполняется file << this;
5. Обработка ошибок через исключения.

LoadGame(const std::string& filename)

1. Открытие файла.
2. file >> this;
3. Вывод сообщения об успехе.
4. Ошибки обрабатываются исключениями.

Проверка допустимости операций

CanSave()

Разрешает сохранение, если:

- состояние НЕ относится к фазам настройки (SET_FIELD, SET_SIZES, SELECT_LOAD_SLOT, ASK_SAVE).

CanLoad()

Разрешено загружать, когда:

- игра завершена (GAME_OVER)
- игра на паузе (PAUSED)
- корабли размещаются (PLACING_SHIPS)
- раунд завершён (PLAYER_WON, ENEMY_WON)
- ожидается новый раунд.

Методы установки состояния

set_player_field_state / set_enemy_field_state

Полностью заменяют поле новым объектом PlayingField.

set_player_abilities

Передаёт AbilityManager новое состояние (копирование через потоковый оператор).

ResetForNewRound

Сбрасывает всю раундовую статистику, курсор и способности.

ResetForNewGame

Сбрасывает раунд + сбрасывает суммарную статистику.

Геттеры и сеттеры

Все геттеры возвращают копии, обеспечивая безопасность внутреннего состояния.

Основные:

- game_status(), set_game_status()

- player_stats(), set_player_stats()
- enemy_stats()
- total_player_stats()
- player_field_state()
- player_abilities()
- round_number()
- cursor_x(), cursor_y()
- is_player_turn() и др.

Управление курсором

MoveCursorBy(int dx, int dy, int maxX, int maxY)

1. Если размеры ≤ 0 → фоллбек:
    - просто смещение с ограничением по минимуму.
2. Иначе:
    - вычисляются новые координаты;
    - значения ограничиваются диапазоном [0, max - 1];
    - используется локальная clamp-функция.

Особенности реализации GameState

Целостность данных

- внутренние структуры защищены через копирование;
- AbilityManager сохраняется и загружается полностью;
- PlayingField и ShipManager всегда пересоздаются при загрузке.

Надёжные сохранения

- директория создаётся автоматически;
- потоковые операторы обеспечивают стабильность формата;
- FileHandler гарантирует закрытие файла.

Гибкость состояний

- поддерживаются состояния для меню, выбора сейва, паузы, игры, переходов;
- всё управление интерфейсом возможно через GameState.

4.3. Структуры результатов Result

Структура AttackResult

Назначение: Хранение результатов атаки для передачи между компонентами системы

Поля:

- hit - результат попадания:
  - -1: ошибка или недопустимая атака
  - 0: промах (мимо корабля)
  - 1: попадание в корабль
  - 2: потопление корабля
- x - координата X атакованной клетки
- y - координата Y атакованной клетки

Структура AbilityResult

Назначение: Хранение результатов применения способности

Поля:

- abilityName - название примененной способности
- x - координата X применения способности (если применимо)
- y - координата Y применения способности (если применимо)

4.4. Класс Game

Назначение:

Центральный контроллер игры — управляет полным жизненным циклом сессии «Морского боя»: от создания игроков и расстановки флота до хода игрока/ИИ, применения способностей, подсчёта статистики, перехода между раундами и сохранения/загрузки состояния.

Ключевые поля:

std::unique_ptr<Player> human_player_ - игрок-пользователь

std::unique_ptr<Player> ai_player_ - ИИ-противник

std::shared_ptr<ShipManager> ship_manager_; - менеджер кораблей (одинаковое для обоих)

std::shared_ptr<AbilityManager> ability_manager_; - менеджер способностей игрока-пользователя

std::string human_name_ - имя игрока-пользователя

GameState current_state_ - текущее состояние игры (поля, статистика, статус, курсор и т.д.)

GameSettings settings_ - настройки (размер поля, состав флота, режим расстановки)

bool show_ships_info_ / show_help_ / show_stats_ - флаги отображения UI-элементов

Конструктор и деструктор

Game(GameSettings new_settings)

Последовательность выполнения:

1. Принимает настройки игры (имя игрока, размер поля, состав флота и т.д.).

2. Перемещает их в член settings_ для эффективного владения.

3. Вызывает Initialize() — создаёт менеджер кораблей и обоих игроков.

4. При любой ошибке инициализации выводит критическое сообщение и завершает процесс.

~Game()

Автоматически вызывает CleanUp(), освобождая все ресурсы и сбрасывая GameState.

Инициализация

Initialize()

1. Создаёт ShipManager из настроек (по умолчанию — классический флот 4-3-3-2-2-2-1-1-1-1).

2. Создаёт объекты Player для человека и ИИ с одинаковым ShipManager.

3. Передаёт владение игроками в set_players() — там же создаётся AbilityManager и связываются все компоненты.

set_players() — ключевая точка сборки:

- Устанавливает ability_manager_ (передаёт полю ИИ и человеку ссылки друг на друга).

- Инициализирует имена и статистику в current_state_.

- Переводит игру в статус PLACING_SHIPS.

Расстановка кораблей

MoveRandomShips() — автоматическая расстановка для игрока

MoveAIShips() — автоматическая расстановка для ИИ

Оба метода используют Player::PlaceShipsRandomly().

При невозможности разместить флот — сбрасывают настройки и реинициализируют игру, бросая ImpossibleFleetException.

RotateShip(), ship_orientation(), current_ship_size() — делегируют PlayingField игрока (поддержка ручной расстановки).

MoveShip(x, y, orientation) — размещает текущий корабль вручную. После завершения всех кораблей автоматически расставляет ИИ и переходит в PLAYER_TURN.

Ход игрока

AttackShipAt(int x, int y)

1. Выполняет выстрел через human_player_->MakeMove(ai_player_, x, y).

2. Обновляет статистику (UpdateScore(), UpdateTotalStats()).

3. Передаёт ход ИИ (ENEMY_TURN).

4. Проверяет условие победы.

AttackShip() — обёртка: стреляет по текущей позиции курсора.

Применение способностей

UseAbility(int x, int y)

1. Вызывает human_player_->UseAbility(x, y) → AbilityManager::ApplyNextAbility.

2. Если способность применилась успешно (координаты ≠ -1) — обновляет статистику и передаёт ход ИИ.

3. Возвращает AbilityResult с названием способности и координатами применения.

Ход ИИ

MakeAIMove()

Интеллектуальная система прицеливания:

1. Собирает три приоритета целей:

- Повреждённые сегменты (для добивания).

- Фронтир — неизвестные клетки, соседние с уже открытыми кораблями.

- Случайные неизвестные клетки (fallback).

2. Выбирает цель по убыванию приоритета с равномерным случайным выбором внутри группы.

3. Совершает выстрел через ai_player_->MakeMove.

4. Обновляет статистику и проверяет победу.

5. При валидном выстреле передаёт ход игроку.

Обновление статистики

UpdateScore() — синхронизирует текущие показатели игроков с GameState.

UpdateTotalStats() — накапливает общую статистику за все раунды.

statistics() — формирует тексn с результатами раунда и общей статистикой.

Проверка победы и завершение раунда

CheckWinCondition()

- Если все корабли ИИ уничтожены → PLAYER_WON.

- Если все корабли игрока уничтожены → ENEMY_WON.

- Вызывает EndRound().

EndRound()

- Увеличивает счётчик побед соответствующей стороне.

- Переходит в состояние WAITING_NEXT_ROUND.

Управление раундами

PrepareNextRound() → ***SaveStateForNextRound()*** + ***LoadStateFromLastRound()***

- Сохраняет текущее состояние полей и менеджера кораблей.

- Пересоздаёт игроков и поля из сохранённых данных.

- Возвращает игру в фазу расстановки (SETTING_SHIPS).

Сохранение и загрузка

SaveGame(filename)

- Сохраняет текущие поля и ShipManager в current_state_.

- Делегирует сериализацию в GameState::SaveGame().

LoadGame(filename) → LoadGameState()

- Загружает состояние в временный GameState.

- Пересоздаёт игроков, поля, AbilityManager и восстанавливает статистику.

UI-вспомогательные методы

- ***ToggleShipsInfo(), ToggleHelp(), ToggleStats()*** — переключение отображения подсказок.

- ***ShouldShow…()*** — текущие флаги видимости.

- ***human_player_ships_info()*** — список кораблей игрока с координатами и статусом размещения.

Работа с курсором

set_cursor(), MoveCursorBy() — ограничивают координаты размерами поля противника.

Управление настройками

void AddShipSize(int size)

- Добавляет один корабль заданного размера в временный список флота.

- Вызывает settings_.AddShipSize(size) → добавляет размер в temp_fleet_spec

void ClearShipSizes()

- Очищает весь кастомный флот (временный список).

- После очистки остаётся кастомный режим — игрок может заново набрать флот с нуля.

void set_auto_ship_sizes()

- Возврат к стандартному (классическому) флоту: 1×4, 2×3, 3×2, 4×1. - Происходит переключение на стандартный режим и присваивается стандартный флот.

void set_ship_fleet_spec()

- Фиксирует текущий кастомный флот как окончательный.

void ApplyFieldSize()

- Применяет выбранный временный размер поля как окончательный.

void set_temp_field_size(int size)

- Временно устанавливает размер поля (для предпросмотра).

int temp_field_size() const

- Возвращает текущий временный размер поля (для отображения в UI).

std::string fleet_spec_string(bool use_temp_fleet = false) const

- Формирует текстовое описание текущего используемого флота или временного (в зависимости от флага)

Примечание:

Game отвечает принципу единственной ответственности:

- Не знает, как рисовать интерфейс.

- Не содержит «интерфейсов»-посредников.

- Работает напрямую с Player, PlayingField, AbilityManager.

- Поддерживает многократные раунды, сохранение/загрузку, гибкие настройки.

4.5. Класс FileHandler

RAII-обертка для безопасной работы с файловыми потоками

Назначение: Обеспечение безопасного управления файловыми ресурсами с автоматическим освобождением и обработкой ошибок

Последовательность выполнения:

1. Конструктор пытается открыть файл по указанному пути в заданном режиме

2. При неудачном открытии генерируется исключение std::runtime_error

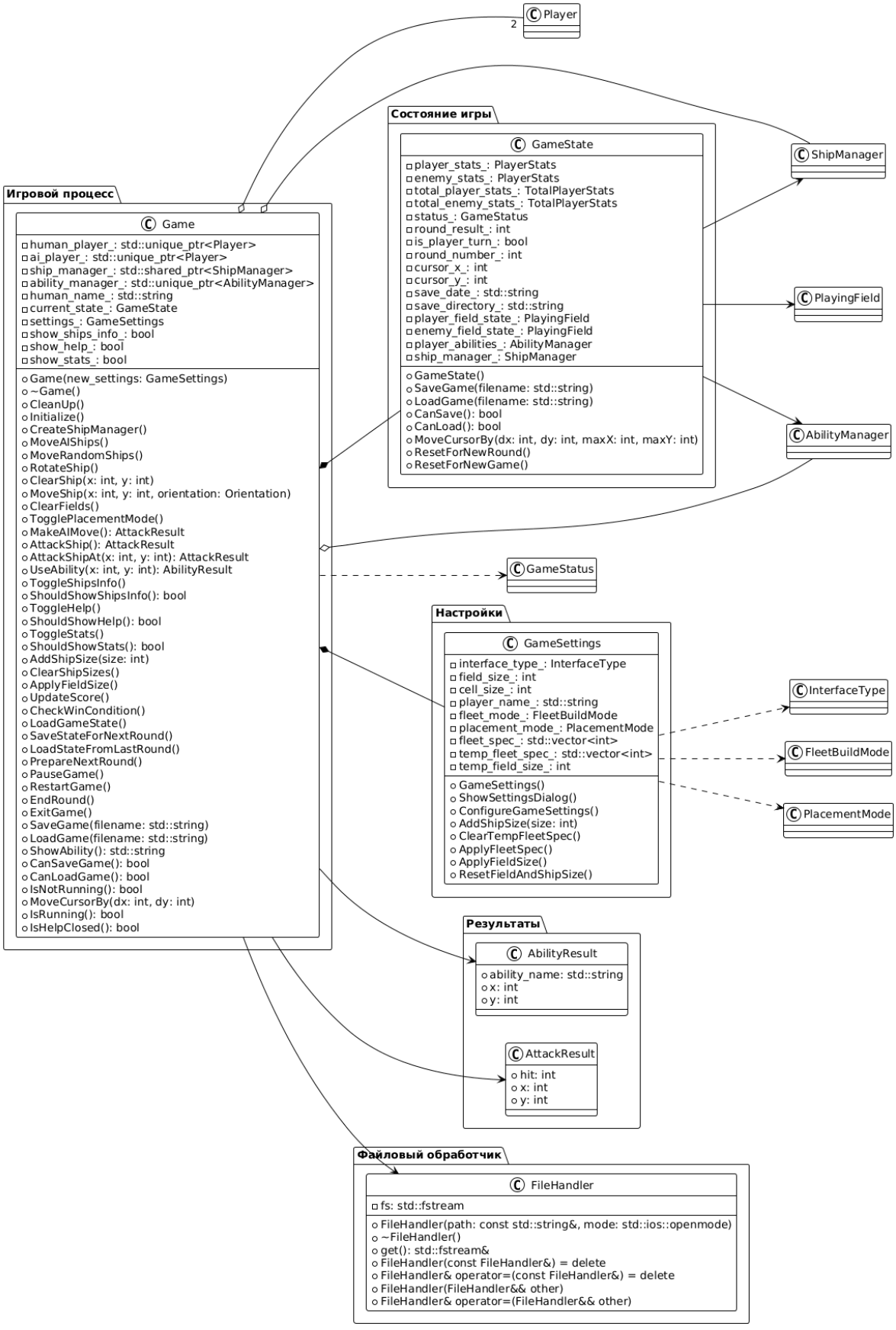
3. Метод get() предоставляет доступ к базовому файловому потоку

4. Деструктор автоматически закрывает файл при разрушении объекта

5. Запрещено копирование для исключения проблем с владением ресурсом

6. Разрешено перемещение для передачи владения файловым потоком

5. Диаграмма классов



Обоснование типов связей:

Композиция (Composition):

Game → GameState: Игра полностью владеет своим состоянием, время жизни GameState зависит от Game

Game → GameSettings: Игра полностью владеет настройками, настройки не существуют без игры

Агрегация (Aggregation):

Game → Player: Игра использует игроков через unique_ptr<Player>, игроки могут быть заменены

Game → ShipManager: Игра использует менеджер кораблей через shared_ptr<ShipManager>, менеджер может использоваться несколькими игроками

Game → AbilityManager: Игра использует менеджер способностей через unique_ptr<AbilityManager>, менеджер может быть пересоздан

Ассоциации:

Game →AttackResult/AbilityResult: Игра временно создает объекты результатов для передачи данных

GameState → PlayingField: Состояние хранит копии игровых полей как часть сериализуемых данных

GameState → AbilityManager/ShipManager: Состояние хранит менеджеры для сохранения/восстановления игры

Game → FileHandler: Игра использует FileHandler для операций с файлами

Зависимости:

Game → GameStatus: Игра зависит от перечисления состояний для управления игровым процессом

GameSettings → InterfaceType/FleetBuildMode/PlacementMode: Настройки зависят от перечислений для определения режимов работы

5. Сохранение и загрузка

Система реализована через класс GameState — единый контейнер всего состояния игры.

Сохранение и загрузка — бинарные, высокопроизводительные, с минимальными накладными расходами.

Никаких текстовых .ini, JSON или EBNF-парсеров — используется прямая сериализация через operator<< и operator>>.

Формат файла сохранения

- Текстовый

- Путь: ./saves/<filename>.save

- Автоматическое создание папки saves/ при первом запуске

- Поддержка до 4 слотов (slot1 – slot4, exit_save)

Порядок и состав сохраняемых данных

При записи (operator<<) данные выводятся строго в следующем порядке:

1. save_date_ → std::string (в кавычках, с датой и временем)

2. status_ → int (приведённый GameStatus)

3. round_result_ + is_player_turn_ → два значения через пробел

4. round_number_ → int

5. cursor_x_ cursor_y_ → два int через пробел

6. player_stats_ → PlayerStats (сериализуется через свой operator<<)

7. enemy_stats_ → PlayerStats

8. total_player_stats_ → TotalPlayerStats

9. total_enemy_stats_ → TotalPlayerStats

10. ship_manager_ → ShipManager (полная структура флота)

11. player_field_size (x y) → два int

12. enemy_field_size (x y) → два int

13. player_field_state_ → PlayingField (полная сетка + корабли)

14. enemy_field_state_ → PlayingField

15. player_abilities_ → AbilityManager (очередь способностей)

При загрузке — точно в том же порядке через operator>>.

Методы сохранения и загрузки (более точно)

void GameState::SaveGame(const std::string& filename)

- Формирует путь: saves/filename.save

- Создаёт/перезаписывает файл

- Генерирует текущую дату и время (DD.MM.YYYY HH:MM)

- Записывает весь объект через file << this

void GameState::LoadGame(const std::string& filename)

- Открывает файл только на чтение

- Загружает данные через file >> this

- Если файл не открылся — молча завершает (без исключения)

std::string GameState::slot_date(const std::string& slot_name) const

- Открывает файл сохранения

- Читает только первую строку (дату)

- Убирает кавычки и возвращает дату

- Используется в меню выбора слота

Условия возможности сохранения/загрузки

bool CanSave() const

Разрешено сохранять только если игра не в меню настроек:

```
return status_ != ASK_SAVE
    && status_ != SET_FIELD
    && status_ != SET_SIZES
    && status_ != SELECT_LOAD_SLOT;
```

bool CanLoad() const

Разрешена загрузка только в «стабильных» состояниях:

- GAME_OVER

- PAUSED

- PLACING_SHIPS

- PLAYER_WON / ENEMY_WON

- WAITING_NEXT_ROUND

Восстановление игры после загрузки

После LoadGame() вызывается Game::LoadGameState():

1. Пересоздаёт ShipManager из сохранённого

2. Создаёт новых Player с правильными размерами поля

3. Копирует сохранённые PlayingField через присваивание

4. Пересоздаёт AbilityManager с правильными ссылками на поля

5. Восстанавливает статистику и имена

→ Полное восстановление состояния, включая:

- Кастомные размеры поля

- Кастомный флот

- Способности в очереди

- Текущий ход, курсор, раунд

6. Архитектурные решения

6.1. Принципы

- *Single Responsibility Principle* - Каждый класс имеет четкую зону ответственности:

  - Game - управление игровым процессом

  - GameState - хранение и сериализация состояния

  - GameSettings - конфигурация параметров игры

  - FileHandler - безопасная работа с файлами

- *Dependency Inversion* - Классы ядра (Game, GameState) не зависят от конкретных реализаций UI, что позволяет легко заменять интерфейсы в будущем.

- *Open/Closed Principle* - Архитектура позволяет добавлять новые способности, состояния игры и режимы настройки без изменения существующего кода.

6.2. Пакетная структура

src/

├── core/ Базовая логика игры (из ЛР №1)

├── abilities/ Система способностей (ЛР №2)

├── additional/ Вспомогательные компоненты

└── controlGame/ Классы для процесса игры

  ├── Game.h/cpp Игровой процесс

  ├── GameState.h/cpp Состояние игра

  ├── Result.h Структуры для результатов атак и способностей

  ├── FileHandler.h Для безопасной работы с файлами

  └── GameSettings.h/cpp Настройки игры

7. Алгоритмы и сложности

7.1. Размещение кораблей

- Проверка клетки + 8-соседей (4 стороны + 4 диагонали) на занятость.
- Сложность постановки одного корабля длины L: O(L) проверок × O(1) на соседей → O(L).

7.2. Атака

- Доступ к клетке — O(1). Обновление статусов в менеджере кораблей — O(1) (прямая адресация по индексу корабля/сегмента).
- Проверка завершения игры — O(кораблей) или поддерживаем счётчик живых кораблей → O(1).

7.3. Способности

- Scanner(2×2): читает до 4 клеток O(1).
- Shelling: выбор корабля/сегмента случайно O(1).
- DoubleDamage: просто флаг на «следующий выстрел».

8. Обоснование ключевых решений

- Единый класс Game как центральный контроллер - Класс Game объединяет все ранее созданные компоненты (игровые поля, корабли, способности) в единый управляемый игровой цикл. Это обеспечивает централизованное управление состоянием игры и координацию между различными системами.

- Разделение Game и GameState - Game отвечает за игровую логику и процесс, а GameState - за сериализуемое состояние. Это позволяет независимо изменять механику игры и формат сохранений.

- Полная сериализация через потоковые операторы - Переопределение operator<< и operator>> для всех игровых сущностей обеспечивает простую и

эффективную систему сохранения/загрузки без зависимостей от внешних библиотек.

- RAII-обертка FileHandler - Гарантирует безопасную работу с файлами, автоматическое освобождение ресурсов и обработку ошибок при операциях сохранения/загрузки.

- Интеллектуальная система приоритетов ИИ - Алгоритм выбора целей с тремя уровнями приоритета (добивание поврежденных кораблей, исследование фронтира, случайный выбор) обеспечивает реалистичное поведение компьютерного противника.

- Гибкая система состояний GameStatus - Расширенное перечисление состояний покрывает не только игровой процесс, но и UI-состояния (меню, диалоги, выбор слотов), что подготавливает архитектуру к интеграции с различными интерфейсами.

- Кастомная настройка флота и поля - Класс GameSettings предоставляет гибкую систему конфигурации, позволяющую изменять размеры поля и состав флота, что расширяет вариативность игрового процесса.

10. Выводы

В ходе выполнения лабораторной работы №3 была успешно разработана система управления игровым процессом "Морской бой", которая объединила ранее созданные компоненты в единую архитектуру. Основные результаты:

1) **Реализован полнофункциональный игровой цикл** - класс Game корректно управляет чередованием ходов игрока и ИИ, обрабатывает применение способностей, обновляет статистику и проверяет условия победы.
2) **Создана система сохранения/загрузки** - класс GameState обеспечивает полную сериализацию состояния игры.
3) **Обеспечена модульность архитектуры** - четкое разделение ответственности между компонентами позволяет независимо заменять UI (консольный/GUI) без изменения игровой логики.
4) **Реализованы продвинутые алгоритмы ИИ** - компьютерный противник использует интеллектуальную стратегию с приоритетом добивания поврежденных кораблей и исследованием фронтира.
5) **Соблюдены принципы ООП** - инкапсуляция, наследование, полиморфизм и композиция применяются для создания расширяемой и поддерживаемой системы.
6) **Обеспечена безопасность и надежность** - обработка исключений, валидация входных данных и инварианты гарантируют стабильную работу приложения.

Архитектура готова к интеграции с системами ввода/вывода в следующей лабораторной работе.

11. Приложения А

Пример файла сохранения

"23.11.2025 19:20"
13
1 1
1
6 3
"1" 40 78 51.2821 10 0
"AI" 38 77 49.3507 9 1
"1" 40 78 51.2821 1 1
"AI" 38 77 49.3507 1 0
10
10
4 3 3 2 2 2 1 1 1 1
10 10
10 10
10 10
10 0
1 -1 -1 1 -1 -1 1 -1 -1 1 -1 -1 1 -1 -1 1 -1 -1 2 8 0 1 -1 -1 1 -1 -1 2 3 0
1 -1 -1 2 5 0 2 5 1 1 -1 -1 1 -1 -1 1 -1 -1 1 -1 -1 1 -1 -1 2 3 1
1 -1 -1 1 -1 -1 1 -1 -1 1 -1 -1 1 -1 -1 1 -1 -1 1 -1 -1 1 -1 -1 1 -1 -1 1 -1 -1
1 -1 -1 2 0 0 1 -1 -1 2 1 0 1 -1 -1 1 -1 -1 1 -1 -1 2 6 0 1 -1 -1 2 2 0
1 -1 -1 2 0 1 1 -1 -1 2 1 1 1 -1 -1 2 4 0 1 -1 -1 1 -1 -1 1 -1 -1 2 2 1
1 -1 -1 2 0 2 1 -1 -1 2 1 2 1 -1 -1 2 4 1 1 -1 -1 1 -1 -1 1 -1 -1 2 2 2
1 -1 -1 2 0 3 1 -1 -1 1 -1 -1 1 -1 -1 1 -1 -1 1 -1 -1 1 -1 -1 1 -1 -1 1 -1 -1
1 -1 -1 1 -1 -1 1 -1 -1 1 -1 -1 1 -1 -1 1 -1 -1 2 9 0 1 -1 -1 2 7 0 1 -1 -1
1 -1 -1 1 -1 -1 1 -1 -1 1 -1 -1 1 -1 -1 1 -1 -1 1 -1 -1 1 -1 -1 1 -1 -1 1 -1 -1
1 -1 -1 1 -1 -1 1 -1 -1 1 -1 -1 1 -1 -1 1 -1 -1 1 -1 -1 1 -1 -1 1 -1 -1 1 -1 -1
1 -1 -1 1 -1 -1 1 -1 -1 1 -1 -1 1 -1 -1 1 -1 -1 0 -1 -1 0 -1 -1 1 -1 -1 2 3 0
1 -1 -1 2 5 0 2 5 1 1 -1 -1 1 -1 -1 1 -1 -1 1 -1 -1 1 -1 -1 2 3 1
1 -1 -1 1 -1 -1 1 -1 -1 1 -1 -1 1 -1 -1 0 -1 -1 1 -1 -1 1 -1 -1 1 -1 -1 1 -1 -1
1 -1 -1 2 0 0 1 -1 -1 2 1 0 1 -1 -1 1 -1 -1 1 -1 -1 2 6 0 1 -1 -1 2 2 0
1 -1 -1 2 0 1 1 -1 -1 2 1 1 1 -1 -1 2 4 0 1 -1 -1 1 -1 -1 1 -1 -1 2 2 1
1 -1 -1 2 0 2 1 -1 -1 2 1 2 1 -1 -1 2 4 1 1 -1 -1 0 -1 -1 1 -1 -1 2 2 2
1 -1 -1 2 0 3 1 -1 -1 1 -1 -1 1 -1 -1 1 -1 -1 1 -1 -1 1 -1 -1 1 -1 -1 1 -1 -1
1 -1 -1 1 -1 -1 1 -1 -1 1 -1 -1 1 -1 -1 1 -1 -1 2 9 0 1 -1 -1 2 7 0 1 -1 -1
1 -1 -1 1 -1 -1 1 -1 -1 1 -1 -1 0 -1 -1 1 -1 -1 1 -1 -1 1 -1 -1 1 -1 -1 1 -1 -1
1 -1 -1 0 -1 -1 1 -1 -1 1 -1 -1 1 -1 -1 1 -1 -1 0 -1 -1 0 -1 -1 1 -1 -1 1 -1 -1
0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0

10
1 3 4 0 0 4 8
2 2 2 2
3 3 3 0 1 3 6
2 2 2
9 3 3 0 2 3 6
2 2 2
9 0 2 0 3 2 4
2 2
5 4 2 0 4 2 4
2 2
1 1 2 1 5 2 4
2 2
7 3 1 0 6 1 2
2
8 7 1 0 7 1 2
2
6 0 1 1 8 0 0
0
6 7 1 1 9 1 2
2
0
10 10
10 0
1 -1 -1 1 -1 -1 1 -1 -1 1 -1 -1 1 -1 -1 1 -1 -1 1 -1 -1 1 -1 -1 1 -1 -1 1 -1 -1 1 -1 -1
1 -1 -1 1 -1 -1 2 5 0 2 5 1 1 -1 -1 1 -1 -1 1 -1 -1 1 -1 -1 1 -1 -1 1 -1 -1 1 -1 -1
1 -1 -1 1 -1 -1 1 -1 -1 1 -1 -1 1 -1 -1 1 -1 -1 1 -1 -1 1 -1 -1 1 -1 -1 1 -1 -1 1 -1 -1
1 -1 -1 2 2 0 1 -1 -1 2 4 0 1 -1 -1 1 -1 -1 2 8 0 1 -1 -1 1 -1 -1 1 -1 -1
1 -1 -1 2 2 1 1 -1 -1 2 4 1 1 -1 -1 1 -1 -1 1 -1 -1 1 -1 -1 1 -1 -1 1 -1 -1 2 6 0
1 -1 -1 2 2 2 1 -1 -1 1 -1 -1 1 -1 -1 2 1 0 2 1 1 2 1 2 1 -1 -1 1 -1 -1
1 -1 -1 1 -1 -1 1 -1 -1 2 3 0 1 -1 -1 1 -1 -1 1 -1 -1 1 -1 -1 1 -1 -1 1 -1 -1 1 -1 -1
1 -1 -1 2 9 0 1 -1 -1 2 3 1 1 -1 -1 1 -1 -1 1 -1 -1 1 -1 -1 1 -1 -1 1 -1 -1 2 7 0
1 -1 -1 1 -1 -1 1 -1 -1 1 -1 -1 1 -1 -1 1 -1 -1 1 -1 -1 1 -1 -1 1 -1 -1 1 -1 -1 1 -1 -1
1 -1 -1 1 -1 -1 1 -1 -1 2 0 0 2 0 1 2 0 2 2 0 3 1 -1 -1 1 -1 -1 1 -1 -1
1 -1 -1 1 -1 -1 1 -1 -1 1 -1 -1 1 -1 -1 0 -1 -1 1 -1 -1 1 -1 -1 1 -1 -1 1 -1 -1 0 -1 -1
0 -1 -1 1 -1 -1 2 5 0 2 5 1 1 -1 -1 0 -1 -1 1 -1 -1 0 -1 -1 0 -1 -1 1 -1 -1
1 -1 -1 1 -1 -1 1 -1 -1 1 -1 -1 1 -1 -1 1 -1 -1 1 -1 -1 1 -1 -1 1 -1 -1 1 -1 -1 0 -1 -1
1 -1 -1 2 2 0 1 -1 -1 2 4 0 1 -1 -1 1 -1 -1 2 8 0 1 -1 -1 1 -1 -1 1 -1 -1
1 -1 -1 2 2 1 1 -1 -1 2 4 1 1 -1 -1 1 -1 -1 1 -1 -1 1 -1 -1 1 -1 -1 1 -1 -1 2 6 0
1 -1 -1 2 2 2 1 -1 -1 1 -1 -1 1 -1 -1 2 1 0 2 1 1 2 1 2 1 -1 -1 1 -1 -1
1 -1 -1 1 -1 -1 1 -1 -1 2 3 0 1 -1 -1 1 -1 -1 1 -1 -1 1 -1 -1 1 -1 -1 1 -1 -1 1 -1 -1
1 -1 -1 2 9 0 1 -1 -1 2 3 1 1 -1 -1 1 -1 -1 1 -1 -1 0 -1 -1 1 -1 -1 1 -1 -1 2 7 0
1 -1 -1 1 -1 -1 1 -1 -1 1 -1 -1 1 -1 -1 1 -1 -1 1 -1 -1 1 -1 -1 1 -1 -1 1 -1 -1 1 -1 -1
0 -1 -1 1 -1 -1 1 -1 -1 2 0 0 2 0 1 2 0 2 2 0 3 1 -1 -1 0 -1 -1 1 -1 -1
0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0

0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0
10
3 9 4 1 0 4 8
2 2 2 2
5 5 3 1 1 3 6
2 2 2
1 3 3 0 2 3 6
2 2 2
3 6 2 0 3 2 4
2 2
3 3 2 0 4 2 4
2 2
2 1 2 1 5 2 4
2 2
9 4 1 1 6 1 2
2
9 7 1 1 7 1 2
2
6 3 1 1 8 1 2
2
1 7 1 0 9 1 2
2
0
3
Shelling
Scanner
Double Damage