SeaBattle-1: Технічна Документація

Група: Комп'ютерна математика
Автор: Поштак Євген

5 грудня 2025 р.

Анотація

SeaBattle-1 — реалізація гри «Морський бій» на C++ з використанням бібліотеки ncurses для графічного інтерфейсу. Проект підтримує штучний інтелект (2 рівні складності), мережевий мультиплеєр через TCP/IP, масштабовані ігрові дошки від 10×10 до 26×26 та систему залпів. Документація описує архітектуру, модулі, алгоритми та особливості реалізації.

Зміст

1 Вступ

1.1 Огляд проекту

SeaBattle-1 — це консольна реалізація класичної гри «Морський бій» з розширеними функціями:

- Кросплатформність (Windows/Linux/macOS)
- Два режими AI: Easy (випадкові постріли) та Smart (цільовий з parity targeting)
- TCP/IP мультиплеєр (Host/Client на порту 54000)
- Автоматична та ручна розстановка кораблів
- 12 категорій автотестів для перевірки функціоналу

1.2 Архітектурна схема

```
data/       - структури даних (BoardData, GameState, GamePiece)
logic/      - ігрова логіка, AI, мережева комунікація
ui/         - візуалізація, анімації, меню
game/       - контролери режимів гри
tests/      - тестова система
```

2 Модуль Data

2.1 board_data.cpp

BoardData::BoardData() — створює дошку 10x10 заповнену водою ('w').
BoardData::BoardData(int size) — створює дошку заданого розміру.
void initialize(int size) — очищає та змінює розмір дошки.
void clear() — скидає дошку до початкового стану (вода).
void resize(int newSize) — змінює розмір ігрового поля.
int receiveShot(int x, int y) — обробляє постріл, повертає 0 (промах), 1 (влучання), 2 (затоплення).
bool isShipSunk(char symbol) — перевіряє, чи затоплений корабель з даним символом.
void markShipAsHit(char symbol) — оновлює статус корабля при влучанні.
int getRemainingShips() — повертає кількість незатоплених кораблів.
int getWoundedCount() — повертає кількість пошкоджених сегментів живих кораблів.
int getSunkCount() — повертає кількість повністю затоплених кораблів.
vector<pair<int,int» getShipCoordinates(char symbol) — отримує всі координати корабля за символом.
vector<pair<int,int» getShipOccupiedCells(int x, int y) — отримує координати всіх клітинок корабля за однією координатою.
void buildShipCellMap() — будує мапу відповідності координат символам кораблів.
void addShip(int orientation, int startPos, int length, char symbol) — розміщує корабель на дошці (1=вертикально, 0=горизонтально).

2.2 Стан клітинок

2.3 game_state.cpp

GameState::GameState() — ініціалізує стан гри з параметрами за замовчуванням.
void initialize(int size, int shots, bool host) — налаштовує параметри гри та створює дошки з "туманом війни"для ворога.

Символ	Опис
'w'	вода (water)
'o'	промах (miss)
'x'	влучання (hit)
's'	затоплений сегмент (sunk)
'A'-'Z'	цілі сегменти кораблів

void reset() — скидає гру до початкового стану зі збереженням конфігурації.
bool isGameOver() const — перевіряє, чи досягнуто ліміт влучань.
bool hasPlayerWon() const — перевіряє, чи переміг гравець.

2.4 ship_data.cpp

Клас GamePiece:
GamePiece::GamePiece() — створює порожній сегмент корабля.
GamePiece::GamePiece(int length, char symbol) — створює сегмент з параметрами.
int Get_Piece_Length() const — повертає довжину корабля.
char Get_Piece_Symbol() const — повертає символ корабля.

Допоміжні функції (в ship_data.hpp):
ShipConfiguration getShipConfig(int boardSize) — повертає конфігурацію флоту для розміру дошки.
int getTotalShips(int boardSize) — розраховує загальну кількість кораблів.
int getTotalShipCells(int boardSize) — розраховує суму довжин усіх кораблів.

3 Модуль Logic

3.1 game_logic.cpp

void initializeGame(GameState& state, int boardSize, int shotsPerTurn, bool isHost) — ініціалізує стан гри з усіма параметрами.
void initializeGamePieces(BoardData& board, std::vector<GamePiece>& pieces) — створює набір кораблів згідно конфігурації.
void generateBoardPlacement(BoardData& board, const std::vector<GamePiece>& pieces) — автоматично розставляє кораблі випадково.
short checkStartingPeg(const BoardData& board, int orientation, int startPos, int length) — перевіряє валідність розміщення: 1=валідно, 2=за межами, 3=колізія.
bool placeShip(BoardData& board, int gridX, int gridY, int orientation, int length, char symbol) — розміщує корабель вручну за координатами.
bool isValidShipPlacement(const BoardData& board, int gridX, int gridY, int orientation, int length) — перевіряє можливість розміщення корабля.
int processShot(BoardData& targetBoard, int x, int y) — обробляє постріл і повертає результат.
void updateSunkShips(BoardData& board, int x, int y) — позначає всі клітинки затопленого корабля.
std::string generateShipSymbol(int shipId) — генерує унікальний символ для корабля.
void markShipParts(int r, int c, int size, const std::vector<std::vector<char»& board, std::vector<std::vector<bool»& visited) — рекурсивно позначає з'єднані частини корабля (DFS).
int countRemainingShips(const std::vector<std::vector<char»& boardArray, int size) — підраховує кількість незатоплених кораблів.

3.2 ai_logic.cpp

AILogic::AILogic(AIDifficulty diff, int size) — створює AI з рівнем складності та розміром дошки.
void setupBoard() — генерує дошку AI з випадковим розміщенням кораблів.
void initializeAvailableShots() — заповнює список доступних пострілів та парних цілей.
void addSmartNeighbors(int x, int y) — додає сусідні клітини до черги цілей після влучання.
AICoordinates pickAttackCoordinates() — обирає наступну ціль (Easy: випадково, Smart: пріоритетно).
void recordShotResult(int x, int y, bool isHit, bool isSunk) — записує результат пострілу та оновлює стратегію.
bool isValidCoordinate(int x, int y) — перевіряє чи координати в межах дошки.
void clearTargetQueue() — очищує чергу пріоритетних цілей.
void reset() — скидає стан AI для нової гри.

3.2.1 Стратегія Smart AI

1. **Parity Targeting** — стріляє по клітинках де $(x+y) \bmod 2 = 0$ (шахматний порядок).
2. **Hunt Mode** — після влучання додає 4 сусідні клітинки до targetQueue.
3. **Пріоритети**: targetQueue → parityShots → випадкові.

3.3 network_logic.cpp

bool initializeNetworking() — ініціалізує Winsock на Windows.
void cleanupNetworking() — звільняє мережеві ресурси.
SOCKET_TYPE createHostSocket() — створює серверний сокет на порту 12345.
SOCKET_TYPE acceptClientConnection(SOCKET_TYPE hostSocket, bool& accepted) — приймає підключення клієнта з таймаутом 60с.
SOCKET_TYPE createClientSocket(const char* hostname) — підключається до хоста.
unsigned long resolveName(const char* name) — перетворює DNS ім'я на IP адресу.
bool sendGameSettings(SOCKET_TYPE socket, int boardSize, int shotsPerTurn) — відправляє налаштування гри.
bool receiveGameSettings(SOCKET_TYPE socket, int& boardSize, int& shotsPerTurn) — отримує налаштування від опонента.
bool sendShot(SOCKET_TYPE socket, const coordinates& shot) — відправляє координати пострілу.
bool receiveShot(SOCKET_TYPE socket, coordinates& shot) — отримує координати пострілу.
bool sendShotResult(SOCKET_TYPE socket, char result) — відправляє результат ('m'/'h'/'s').
bool receiveShotResult(SOCKET_TYPE socket, char& result) — отримує результат пострілу.
bool sendShotCount(SOCKET_TYPE socket, int count) — відправляє кількість пострілів у залпі.
bool receiveShotCount(SOCKET_TYPE socket, int& count) — отримує кількість пострілів.

4 Модуль Game

4.1 game-loop.cpp

void GameLoop::runGameLoop(...) — основний ігровий цикл для AI та мережевої гри. Координує хід гравця (вибір цілей, стрільба), хід AI/опонента, оновлення дошок та перевірку умов перемоги.

Фази ходу гравця:

1. **Selection Mode**: WASD/стрілки для руху курсора, Space/Enter для вибору цілі, F для підтвердження залпу, Q для виходу.
2. **Firing Mode**: відправка пострілів, отримання результатів, відображення статистики залпу.

4.2 ai-game-loop.cpp

void playAIGame(AIDifficulty difficulty) — ініціалізує та запускає гру проти AI. Включає вибір розміру дошки, кількості пострілів, розстановку кораблів (авто/ручну) та запуск ігрового циклу.

Ручна розстановка: WASD/стрілки — рух, R — поворот, Space/Enter — розміщення, G — повернення до авто-генерації.

4.3 multiplayer-game-loop.cpp

void playMultiplayerHost() — створює сокет хоста, чекає клієнта, обирає налаштування, розставляє кораблі та запускає гру (хост ходить першим).
void playMultiplayerClient() — підключається до хоста, отримує налаштування, розставляє кораблі та запускає гру (клієнт ходить другим).

Керування розстановкою: аналогічно AI режиму.

4.4 game-controller.cpp

int setupPlayerBoard(BoardData& board, int size) — генерує випадкову дошку і пропонує вибір: Y (прийняти), N (регенерувати), M (перейти до ручного режиму). Повертає 1 (прийнято), 0 (ручний режим), -1 (регенерація).

5 Модуль UI

5.1 ui_renderer.cpp

void setupWindow() — ініціалізує ncurses, створює 6 кольорових пар, встановлює обробники сигналів.
void drawTitle() — малює ASCII-арт заголовок гри.
void drawGameBoards(...) — малює дві дошки з заголовками, координатами та сіткою.
void drawBoardCell(int y, int x, char cell, bool isPlayer) — малює клітинку з відповідним кольором (вода, корабель, влучання, промах).
int selectBoardSize() — інтерактивне меню вибору розміру з слайдером та перевіркою розміру терміналу.
int selectShotsPerTurn(int boardSize) — меню вибору кількості пострілів за хід (1-26).
int showMainMenu(int& selectedOption) — відображає головне меню з анімованим фоном.

5.2 ui_animation.cpp

void drawBottomShipAnimation(int frame, int startY, int maxX) — анімація морського бою внизу екрану (кораблі, постріли, вибухи).

void drawFirework(bool playerWon) — фінальна анімація: салют при перемозі або затонулі кораблі при поразці.
void drawMenuAnimation(int frame) — анімація фону меню (підводний човен, міни, риби, водорості).

5.3 ui_config.cpp

void setBoardSize(int size) — встановлює глобальний розмір дошки (10-26).
int getBoardSize() — повертає поточний розмір дошки.
bool canFitInterface(int boardSize, int maxY, int maxX) — перевіряє, чи інтерфейс вміщується у вікно терміналу.
void getRequiredTerminalSize(int boardSize, int& minY, int& minX) — розраховує необхідні розміри терміналу.
BoardLayout calculateBoardLayout(int boardSize) — розраховує координати всіх елементів UI для центрування.

5.4 ui_helpers.cpp

std::string getColumnLetter(int index) — перетворює індекс стовпця в літеру (0 → 'A').
void gridToScreen(...) — перетворює логічні координати сітки у фізичні координати екрану.
void screenToGrid(...) — зворотне перетворення: екранні координати у координати сітки.

6 Модуль Tests

6.1 SeaBattle_1_test.cpp

Містить 12 категорій автоматичних тестів, ручні інтерактивні тести та тести на основі файлу.

void runDebugTests() — головне меню тестів (автоматичні, ручні, файлові, всі).

6.1.1 Автоматичні тести (12 категорій)

1. **testBoardSizeValidation()** — перевірка ініціалізації дошки (мін/макс розміри, заповнення водою).
2. **testShipPlacementValidation()** — валідація розміщення (валідні/невалідні позиції, OOB, перекриття).
3. **testShipRotation()** — перевірка горизонтального/вертикального розміщення та блокування ротації.
4. **testShotValidation()** — логіка пострілів (промах, влучання, потоплення, дублікати, OOB).
5. **testShipCounting()** — підрахунок статистики флоту (залишилось, пошкоджені, затоплені).
6. **testVolleySystem()** — система залпів (обробка черги пострілів, підрахунок промахів).
7. **testEasyAI()** — поведінка Easy AI (випадковість, стабільність).
8. **testSmartAI()** — поведінка Smart AI (таргетинг сусідів, парність, скидання режиму полювання).

9. **testGameState()** — управління станом (ініціалізація, перевірка завершення/перемоги).
10. **testShipConfiguration()** — конфігурація флоту (10x10, 15x15, масштабування).
11. **testBoardGeneration()** — генерація поля (створення фігур, розміщення, стабільність).
12. **testCoordinateSystem()** — система координат (конвертація позиції, отримання клітинок корабля).

void runManualTests() — інтерактивні тести з консольним вводом (12 категорій).
void runFileTests() — тести на основі файлу `SeaBattle_1_test.dat`.

7 Конфігурація кораблів

Табл. 1: Конфігурація флоту за розміром дошки

Розмір	4-п.	3-п.	2-п.	1-п.	Клітинки	Shots
10×10	1	2	3	4	20	5
15×15	2	4	6	8	40	6
20×20	3	5	8	12	55	7
26×26	4	7	11	18	77	9

8 Встановлення та Компіляція

8.1 Системні Вимоги

- **Windows:** MinGW-w64, PDCurses
- **Linux/macOS:** GCC/Clang, ncurses
- **Термінал:** мінімум 80x30 символів

8.2 Встановлення Залежностей

8.2.1 Windows (MSYS2)

Відкрийте термінал MSYS2 та введіть наступні команди:

```
# Update system
pacman -Syu

# Install compiler and build tools
pacman -S mingw-w64-x86_64-gcc mingw-w64-x86_64-make

# Install PDCurses library
pacman -S mingw-w64-x86_64-pdcurses
```

8.2.2 Linux (Ubuntu/Debian)

```
sudo apt update
sudo apt install build-essential libncurses5-dev libncursesw5-dev
```

8.2.3 macOS

```
# Using Homebrew
brew install ncurses
```

8.3 Компіляція та Запуск

8.3.1 Базова Збірка

```
# Build project
make

# Run game
./battleship        # Linux/macOS
./battleship.exe    # Windows
```

8.3.2 Інші Команди Makefile

```
make clean    # Remove object files
make rebuild  # Full rebuild (clean + all)
make debug    # Build with debug symbols
make release  # Optimized build
```

8.4 Перевірка Встановлення

Після збірки можна запустити автоматичні тести:

```
./battleship
```

9 Стратегії Гри

9.1 Базові Принципи

9.1.1 Розстановка Кораблів

Оптимальні патерни:

- **Краї дошки:** розміщуйте великі кораблі (3-4 палуби) біля країв — AI рідше перевіряє ці зони першими.
- **Кластеризація:** уникайте розміщення кораблів впритул — після влучання Smart AI обстрілює сусідні клітини.
- **Діагональне розміщення:** чергуйте орієнтацію кораблів (горизонтально/вертикально) для ускладнення передбачення патерну.
- **Розсіювання:** розташовуйте 1-палубні кораблі в шаховому порядку, щоб ускладнити їх пошук.

Приклад оптимальної розстановки (10x10):

```
   A   B   C   D   E   F   G   H   I   J
1 |   |   | A |   |   |   |   |   | B |   |
2 |   |   | A |   | C | C | C |   | B |   |
3 |   |   | A |   |   |   |   |   |   |   |
4 | D |   | A |   |   |   |   |   | E |   |
```

```
   5 |   |   |   |   |   |   |   |   |   |   |
   6 |   | F | F |   |   |   | G | G |   |   |
   7 |   |   |   |   | H |   |   |   |   | I |
   8 |   |   |   |   |   |   |   |   |   |   |
   9 |   | J | J | J |   |   |   |   |   |   |
  10 |   |   |   |   |   |   |   |   |   |   |
```

9.2 Тактики Обстрілу

9.2.1 Проти Easy AI

Easy AI стріляє випадково, тому:

- Використовуйте агресивну стратегію — максимізуйте швидкість пошуку кораблів.
- **Шахова дошка:** стріляйте по клітинках де $(x+y) \bmod 2 = 0$ — це гарантує влучання в будь-який корабель довжиною 2+ клітинки.
- **Швидкий фініш:** після влучання негайно обстрілюйте 4 сусідні клітини для знищення корабля.

9.2.2 Проти Smart AI

Smart AI використовує таргетинг та parity strategy:

- **Уникайте передбачуваності:** не розміщуйте кораблі в очевидних місцях (центр, рівномірний розподіл).
- **Використовуйте залпи ефективно:** при багатьох пострілах за хід фокусуйтесь на знищенні вже підбитих кораблів замість пошуку нових.
- **Випереджайте AI:** Smart AI полює по шаховому порядку, тому розміщуйте кораблі так, щоб вони займали клітинки обох кольорів.
- **Контратака:** використовуйте агресивну стратегію пошуку — чим швидше ви знищите флот AI, тим менше часу він матиме на пошук ваших кораблів.

9.2.3 Мультиплеєр

Проти людини стратегія відрізняється:

- **Психологія:** люди схильні до патернів — розміщуйте кораблі нетипово (не по краях, не в центрі).
- **Адаптація:** спостерігайте за стилем противника — якщо він обстрілює центр, розмістіть кораблі по периметру.
- **Залпи:** координуйте постріли так, щоб покрити максимальну площу або зосередитися на знищенні конкретних кораблів.
- **Темп гри:** швидкість прийняття рішень впливає на результат — балансуйте між швидкістю та точністю.

9.3 Просунуті Техніки

9.3.1 Математичний Підхід

Ймовірнісний аналіз:

- Для кожної невідомої клітинки розрахуйте ймовірність розміщення корабля.
- Формула: $P(cell) = \sum_{ship} \frac{valid\_positions(ship,cell)}{total\_valid\_positions(ship)}$
- Стріляйте по клітинках з найвищою ймовірністю.

Оптимізація залпів:

- При n пострілах за хід, максимізуйте інформаційний виграш.
- Для дошки 10×10 з 5 пострілами: 2 на пошук + 3 на добивання.
- Формула ефективності: $E = \frac{hits+sunk\times 2}{shots}$

9.3.2 Адаптивна Стратегія

Фази гри:

1. **Рання гра (0-30% втрат):**
   - Фокус на пошуку великих кораблів (3-4 палуби).
   - Використовуйте широкий пошук по шаховому порядку.
   - Уникайте країв дошки — там зазвичай менше кораблів.
2. **Середня гра (30-70% втрат):**
   - Переключайтеся на знищення виявлених кораблів.
   - Використовуйте залпи для швидкого добивання.
   - Аналізуйте залишкові можливі позиції для невиявлених кораблів.
3. **Пізня гра (70-100% втрат):**
   - Систематичний пошук останніх кораблів.
   - Обстрілюйте всі ймовірні позиції послідовно.
   - Враховуйте розміри невиявлених кораблів.

9.3.3 Спеціальні Прийоми

Техніка "Хрест":

- Після влучання стріляйте хрестом (4 напрямки).
- Як тільки визначено орієнтацію — добивайте вздовж осі.

Техніка "Спіраль":

- Починайте з центру та рухайтеся по спіралі назовні.
- Ефективно для дошок 15×15 та більше.

Техніка "Сектори":

- Розділіть дошку на 4 квадранти.
- Очищуйте квадранти по черзі для кращого контролю.

9.4 Статистика та Аналіз

9.4.1 Ключові Метрики

- **Accuracy:** $\frac{hits}{total\_shots} \times 100\%$ — бажано $> 40\%$
- **Efficiency:** $\frac{sunk\_ships}{total\_shots}$ — вище = краще
- **Average shots per kill:** $\frac{total\_shots}{sunk\_ships}$ — нижче = краще
- **Optimal for 10x10:** 35-50 пострілів для повної перемоги

9.4.2 Типові Помилки

1. **Хаотичний обстріл:** відсутність системи призводить до дублювання зон.
2. **Надмірна фокусування:** витрата всіх пострілів на один корабель замість пошуку нових.
3. **Ігнорування статистики:** не врахування розмірів знищених/залишених кораблів.
4. **Передбачуваність:** використання однакових патернів в кожній грі.

10 Особливості Реалізації

10.1 Кросплатформність

Проект забезпечує кросплатформну роботу на системах **Windows** та **POSIX-сумісних** (Linux/macOS) за допомогою умовної компіляції (`#ifdef _WIN32`).

- **Системні виклики (Sleep):** Використовується макрос `SLEEP_MS(x)` для забезпечення паузи у мілісекундах:

```
#ifdef _WIN32
    #define SLEEP_MS(x) Sleep(x)
#else
    #define SLEEP_MS(x) usleep((x)*1000)
#endif

```

- **Мережа (Sockets):** Тип сокета визначається за допомогою `typedef`:

```
#ifdef _WIN32
    typedef SOCKET SOCKET_TYPE; // Winsock2
#else
    typedef int SOCKET_TYPE; // POSIX sockets
#endif

```

- **GUI:** Використовуються бібліотеки **PDCurses** для Windows та **ncurses** для Unix-подібних систем.

10.2 Алгоритми AI

Реалізовано два рівні складності штучного інтелекту.

1. **Easy AI:**
   - **Складність:** $O(1)$ вибір.
   - **Метод:** Вибір випадкової невідвіданої клітинки через перемішування списку.
   - **Ефективність:** $\sim 20\%$ точність влучань.
2. **Smart AI:**
   - **Стратегія Parity Targeting:** Пошук зменшено на $\sim 50\%$ за рахунок обстрілу клітинок $(x + y) \bmod 2 = 0$.
   - **Режим Hunt Mode:** Після влучання додає 4 сусідні клітини до черги цілей.
   - **Ефективність:** $\sim$ 45% точність влучань, на 30-40% швидше за випадковий пошук.
   - **Складність:** $O(1)$ вибір наступної цілі завдяки попередньо підготовленим чергам.

10.3 Мережева Надійність

Надійність TCP/IP мультиплеєра забезпечується наступними механізмами:

- **Таймаут `recv`:** Встановлено ліміт очікування у **60 секунд** для запобігання зависання.
- **Гарантоване отримання:** Використовується прапор `MSG_WAITALL` для отримання повних пакетів.
- **Обробка помилок:** Кожна операція перевіряється з автоматичним закриттям сокету при збої.
- **Протокол обміну:**
  1. Хост відправляє налаштування (розмір дошки, кількість пострілів).
  2. Клієнт підтверджує отримання.
  3. Обмін координатами та результатами пострілів.
  4. Синхронізація ходів через підтвердження.

10.4 Адаптивний UI

Інтерфейс користувача адаптується до розміру терміналу та ігрового поля.

- **Динамічний Layout:** Позиції елементів розраховуються для центрування:

```
int totalWidth = 2 * (boardSize * 4 + 8) + 5;
layout.board1StartX = (maxX - totalWidth) / 2;

```

- **Масштабування заголовків:** Текст адаптується до розміру дошки (10x10: "Your Board 20x20: "You").
- **Валідація розміру:** Функція `canFitInterface()` перевіряє відповідність:

$$\text{minY} = \text{boardSize} + 20; \quad \text{minX} = 2 \times (8 + \text{boardSize} \times 4) + 5$$

- **Анімації:** Використовується неблокуючий ввід (`nodelay`) для плавності 10 FPS.

10.5 Тестування

Система тестування є модульною та комплексною.

- **Структура:** 12 категорій автотестів (60+ кейсів), ручні тести, файлові тести.
- **Покриття:** Логіка розміщення, валідація пострілів, статистика флоту, залпи, AI, стан гри.
- **Звітність:** Результати у консоль та файл `test_results.txt` з відсотком успішності.
- **Автоматизація:** Тести запускаються одною командою з детальним логуванням.

11 Висновки

Проект **SeaBattle-1** — це успішна, багатофункціональна та кросплатформна реалізація гри "Морський бій"у консольному середовищі.

11.1 Ключові Досягнення

- **Модульність та Кросплатформність:** Архітектура забезпечує легкість підтримки та роботу на Windows/Linux/macOS.
- **Інтелектуальний AI:** Smart AI з Parity Targeting та Hunt Mode суттєво підвищує складність гри.
- **Мережева Стабільність:** Реалізація TCP/IP з таймаутами та гарантованим отриманням забезпечує надійну комунікацію.
- **Адаптивність UI:** Інтерфейс динамічно підлаштовується під розмір дошки (10x10 до 26x26).
- **Якість Коду:** Комплексна система тестування забезпечує стабільність та надійність.

11.2 Технічні Характеристики

- **Мова:** C++11
- **Бібліотеки:** ncurses/PDCurses, Winsock2/BSD sockets
- **Розміри дошок:** 10x10 до 26x26
- **Режими гри:** AI (Easy/Smart), Мультиплеєр (Host/Client)
- **Система залпів:** 1-26 пострілів за хід
- **Тести:** 12 категорій, 60+ тест-кейсів

11.3 Можливості Розширення

- **Додаткові рівні AI:** Експертний режим з машинним навчанням.
- **Турнірний режим:** Система рейтингів та статистики гравців.
- **Варіації правил:** Спеціальні здібності, туман війни, обмеження часу.
- **Графічний інтерфейс:** Портування на SDL/Qt для повноцінного GUI.
- **Онлайн-режим:** Інтеграція з сервером для гри через Інтернет.