МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ імені Тараса Шевченка
ФАКУЛЬТЕТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ
Кафедра програмних систем і технологій

Дисципліна
« Кросплатформне програмування,»

Лабораторна робота № 8
на тему:
"Гра Морський бій"

Виконав:	Безруков Андрій Миколайович	**Перевірив**:	Васильєв Олексій Миколайович
Група	ІПЗ-33	Дата перевірки	
Форма навчання	денна	Оцінка	
Спеціальність	121		
2024			

1. Постановка задачі

Розробити Java- консольний застосунок гри «Морський бій» для двох гравців (користувач і комп’ютер) на полях розміром 10×10 із розміщенням флоту:

- 1 чотирипалубний корабель;
- 2 трипалубні кораблі;
- 3 двопалубні кораблі;
- 4 однопалубні кораблі;

Кораблі не повинні торкатися один одного навіть кутами. Перед грою випадково визначається, хто ходить першим. Гравці по черзі здійснюють постріли по координатах суперника й отримують відповіді «Мимо», «Влучив», «Потопив». Переможець — той, хто першим потопить усі кораблі противника.

2. Опис реалізації та програмний код

Застосунок реалізовано у класі `lab8` з такими ключовими компонентами:

- Ініціалізація полів та випадкове розміщення кораблів відповідно до правил;
- Метод `displayBoard` для виводу стану полігону через ASCII-символи;
- Обробка ходу гравця (`playerMove`) із валідацією координат;
- Обробка ходу комп’ютера (`computerMove`) з двома режимами роботи;
- Логіка пострілів та зменшення лічильників залишкових кораблів;
- Перевірка потоплення корабля (`checkSunk`).

Розділ "Програмний код"

```
package lab8;

import java.util.Random;
import java.util.Scanner;

public class lab8 {
   private static final int BOARD_SIZE = 10;
```

```
    private static final int EMPTY = 0;
    private static final int SHIP = 1;
    private static final int HIT = 2;
    private static final int MISS = 3;

    private static final int[] SHIP_SIZES = {4, 3, 3, 2, 2, 2, 1, 1, 1,
1};

    private static int[][] playerBoard = new
int[BOARD_SIZE][BOARD_SIZE];
    private static int[][] computerBoard = new
int[BOARD_SIZE][BOARD_SIZE];
    private static int[][] playerView = new int[BOARD_SIZE][BOARD_SIZE];
// what player sees of computer's board
    private static int[][] computerView = new
int[BOARD_SIZE][BOARD_SIZE]; // what computer sees of player's board

    private static int playerShipsRemaining = 10;
    private static int computerShipsRemaining = 10;

    private static Random random = new Random();
    private static Scanner scanner = new Scanner(System.in);

    // Tracks previous successful hits to continue targeting adjacent
cells
    private static int lastHitX = -1;
    private static int lastHitY = -1;
    private static boolean huntMode = false;
    private static int huntDirection = 0; // 0-north, 1-east, 2-south,
3-west

    public static void main(String[] args) {
        System.out.println("=== МОРСЬКИЙ БІЙ ===");

        initializeBoards();
        placeShips(playerBoard);
        placeShips(computerBoard);

        boolean playerTurn = random.nextBoolean();
        System.out.println("Визначаємо, хто ходить першим...");
        System.out.println(playerTurn ? "Ви ходите першим!" : "Комп'ютер
ходить першим!");
```

```
        while (playerShipsRemaining > 0 && computerShipsRemaining > 0) {
            if (playerTurn) {
                System.out.println("\nВаш хід:");
                displayBoard(playerBoard, false);
                System.out.println();
                displayBoard(playerView, true);

                playerTurn = playerMove();
            } else {
                System.out.println("\nХід комп'ютера:");
                playerTurn = !computerMove();

                // Add a small delay to make it feel more natural
                try {
                    Thread.sleep(1000);
                } catch (InterruptedException e) {
                    e.printStackTrace();
                }
            }
        }

        // Game over
        if (playerShipsRemaining == 0) {
            System.out.println("\nВи програли! Комп'ютер потопив усі
ваші кораблі.");
        } else {
            System.out.println("\nВи перемогли! Ви потопили всі кораблі
комп'ютера.");
        }

        // Display final boards
        System.out.println("\nВаша фінальна дошка:");
        displayBoard(playerBoard, false);
        System.out.println("\nДошка комп'ютера:");
        displayBoard(computerBoard, false);
    }

    private static void initializeBoards() {
        for (int i = 0; i < BOARD_SIZE; i++) {
            for (int j = 0; j < BOARD_SIZE; j++) {
                playerBoard[i][j] = EMPTY;
                computerBoard[i][j] = EMPTY;
                playerView[i][j] = EMPTY;
```

```
                computerView[i][j] = EMPTY;
            }
        }
    }

    private static void placeShips(int[][] board) {
        for (int shipSize : SHIP_SIZES) {
            boolean placed = false;
            while (!placed) {
                int row = random.nextInt(BOARD_SIZE);
                int col = random.nextInt(BOARD_SIZE);
                boolean horizontal = random.nextBoolean();

                if (canPlaceShip(board, row, col, shipSize, horizontal))
{
                    placeShip(board, row, col, shipSize, horizontal);
                    placed = true;
                }
            }
        }
    }

    private static boolean canPlaceShip(int[][] board, int row, int col,
int shipSize, boolean horizontal) {
        // Check if ship fits on the board
        if (horizontal) {
            if (col + shipSize > BOARD_SIZE) return false;
        } else {
            if (row + shipSize > BOARD_SIZE) return false;
        }

        // Check if ship area and surrounding area is clear
        int startRow = Math.max(0, row - 1);
        int endRow = Math.min(BOARD_SIZE - 1, horizontal ? row + 1 : row
+ shipSize);
        int startCol = Math.max(0, col - 1);
        int endCol = Math.min(BOARD_SIZE - 1, horizontal ? col +
shipSize : col + 1);

        for (int i = startRow; i <= endRow; i++) {
            for (int j = startCol; j <= endCol; j++) {
                if (board[i][j] == SHIP) {
                    return false;
```

```
                }
            }
        }

        return true;
    }

    private static void placeShip(int[][] board, int row, int col, int
shipSize, boolean horizontal) {
        if (horizontal) {
            for (int j = col; j < col + shipSize; j++) {
                board[row][j] = SHIP;
            }
        } else {
            for (int i = row; i < row + shipSize; i++) {
                board[i][col] = SHIP;
            }
        }
    }

    private static void displayBoard(int[][] board, boolean hideShips) {
        System.out.println("  A B C D E F G H I J");
        for (int i = 0; i < BOARD_SIZE; i++) {
            System.out.print((i + 1) + " ");
            if (i + 1 < 10) System.out.print(" ");

            for (int j = 0; j < BOARD_SIZE; j++) {
                char symbol;
                switch (board[i][j]) {
                    case SHIP:
                        symbol = hideShips ? '·' : '■';
                        break;
                    case HIT:
                        symbol = 'X';
                        break;
                    case MISS:
                        symbol = 'o';
                        break;
                    default:
                        symbol = '·';
                }
                System.out.print(symbol + " ");
            }
```

```
            System.out.println();
        }
    }

    private static boolean playerMove() {
        int row = 0;
        int col = 0;
        boolean validInput = false;

        do {
            System.out.print("Введіть координати пострілу (наприклад,
A5): ");
            String input = scanner.nextLine().toUpperCase();

            if (input.length() < 2 || input.length() > 3) {
                System.out.println("Неправильний формат. Спробуйте ще
раз.");
                continue;
            }

            char colChar = input.charAt(0);
            if (colChar < 'A' || colChar > 'J') {
                System.out.println("Неправильна колонка. Використовуйте
літери від A до J.");
                continue;
            }

            col = colChar - 'A';

            try {
                row = Integer.parseInt(input.substring(1)) - 1;
                if (row < 0 || row >= BOARD_SIZE) {
                    System.out.println("Неправильний рядок.
Використовуйте числа від 1 до 10.");
                    continue;
                }
            } catch (NumberFormatException e) {
                System.out.println("Неправильний рядок. Використовуйте
числа від 1 до 10.");
                continue;
            }
```

```
            if (playerView[row][col] == HIT || playerView[row][col] ==
MISS) {
                System.out.println("Ви вже стріляли в цю клітину.
Спробуйте ще раз.");
                continue;
            }

            validInput = true;
        } while (!validInput);

        return processShot(row, col, computerBoard, playerView);
    }

    private static boolean computerMove() {
        int row, col;
        boolean validTarget = false;

        do {
            if (huntMode) {
                // Target cells adjacent to previous hits
                switch (huntDirection) {
                    case 0: // North
                        row = lastHitY - 1;
                        col = lastHitX;
                        break;
                    case 1: // East
                        row = lastHitY;
                        col = lastHitX + 1;
                        break;
                    case 2: // South
                        row = lastHitY + 1;
                        col = lastHitX;
                        break;
                    case 3: // West
                        row = lastHitY;
                        col = lastHitX - 1;
                        break;
                    default:
                        row = random.nextInt(BOARD_SIZE);
                        col = random.nextInt(BOARD_SIZE);
                }
```

```
                // If target is invalid or already shot, try next
direction
                if (row < 0 || row >= BOARD_SIZE || col < 0 || col >=
BOARD_SIZE ||
                    computerView[row][col] == HIT ||
computerView[row][col] == MISS) {
                    huntDirection = (huntDirection + 1) % 4;
                    if (huntDirection == 0) {
                        // If we've tried all directions, go back to
random targeting
                        huntMode = false;
                    }
                    continue;
                }
            } else {
                // Use probability density algorithm for targeting
                // Simple version: random targeting for now
                row = random.nextInt(BOARD_SIZE);
                col = random.nextInt(BOARD_SIZE);
            }

            if (computerView[row][col] != HIT && computerView[row][col]
!= MISS) {
                validTarget = true;
            }
        } while (!validTarget);

        System.out.println("Комп'ютер стріляє в " + (char)('A' + col) +
(row + 1));

        return processShot(row, col, playerBoard, computerView);
    }

    private static boolean processShot(int row, int col, int[][]
targetBoard, int[][] viewBoard) {
        if (targetBoard[row][col] == SHIP) {
            targetBoard[row][col] = HIT;
            viewBoard[row][col] = HIT;

            // Check if ship is sunk
            boolean sunk = checkSunk(row, col, targetBoard);

            if (sunk) {
```

```
                System.out.println("Потопив!");
                if (targetBoard == computerBoard) {
                    computerShipsRemaining--;
                } else {
                    playerShipsRemaining--;
                    // Reset hunt mode if ship is sunk
                    huntMode = false;
                }
            } else {
                System.out.println("Влучив!");
                if (targetBoard == playerBoard) {
                    // Enter hunt mode when hit player's ship
                    huntMode = true;
                    lastHitX = col;
                    lastHitY = row;
                    huntDirection = 0;
                }
            }
            return true; // Keep turn
        } else {
            targetBoard[row][col] = MISS;
            viewBoard[row][col] = MISS;
            System.out.println("Мимо!");
            return false; // End turn
        }
    }

    private static boolean checkSunk(int row, int col, int[][] board) {
        // Check horizontally and vertically if there are any unsunk
ship parts
        for (int dr = -1; dr <= 1; dr += 2) {
            int r = row;
            while (r >= 0 && r < BOARD_SIZE) {
                r += dr;
                if (r < 0 || r >= BOARD_SIZE || board[r][col] != SHIP &&
board[r][col] != HIT) {
                    break;
                }
                if (board[r][col] == SHIP) {
                    return false; // Found unsunk part
                }
            }
        }
```

```
        for (int dc = -1; dc <= 1; dc += 2) {
            int c = col;
            while (c >= 0 && c < BOARD_SIZE) {
                c += dc;
                if (c < 0 || c >= BOARD_SIZE || board[row][c] != SHIP &&
board[row][c] != HIT) {
                    break;
                }
                if (board[row][c] == SHIP) {
                    return false; // Found unsunk part
                }
            }
        }

        return true; // All parts of the ship are hit
    }
}
```

3. Алгоритм ходів комп’ютера та обґрунтування

Комп’ютер застосовує двофазний алгоритм:

1. **Режим "Полювання" (hunt mode):**
    - При успішному влучанні (стан `HIT`) комп’ютер запам’ятовує координати останнього вдалого пострілу (`lastHitX`, `lastHitY`) та послідовно перевіряє сусідні клітинки у напрямках північ, схід, південь, захід, доки не потопить корабель або не вичерпає сусідні клітини.
    - Така стратегія дозволяє швидко докласти пострілів до знайденої частини корабля та потопити його повністю, мінімізуючи марні ходи.
2. **Режим "Випадкового вогню":**
    - Коли комп’ютер не перебуває у режимі полювання, він обирає координати випадково серед клітин, у які ще не стріляв.
    - Для спрощення логіки та уникнення надмірних обчислень використовуються випадкові цілі, але можна покращити цю стратегію, застосувавши аналіз ймовірнісної щільності

(probability density) залежно від розмірів неопіслядених кораблів.

Доцільність вибору алгоритму: комбінування випадкових пострілів із режимом полювання надає необхідний баланс між простотою реалізації та ефективністю: комп'ютер швидко виявляє та потоплює кораблі суперника, водночас залишаючись несподіваним під час початкових пострілів.

4. Результати тестування

Програма протестована на:

- **Windows 10**
- **Arch Linux**

Перевірено сценарії:

- правильне розміщення флоту з дотриманням відстаней між кораблями;
- послідовність ходів і передача ходу супернику після промаху;
- коректність відповідей «Мимо», «Влучив», «Потопив»;
- робота hunt mode: виявлення та потоплення багатокорпусних кораблів;
- завершення гри при потопленні всіх кораблів противника.

5. Висновки

Створено консольний Java- застосунок гри «Морський бій» з інтелектуальним алгоритмом для комп'ютера. Програма:

- моделює розміщення флоту за класичними правилами;
- забезпечує коректну взаємодію гравців та обробку результатів пострілів;
- реалізує комбінований алгоритм комп'ютера (hunt mode + random mode), що підвищує ефективність і «людяність» гри.

Тестування на Windows та Arch Linux довело стабільність і коректність роботи.
Перспективи подальшого розвитку: застосування ймовірнісного аналізу для оптимізації випадкових пострілів, збереження статистики ігор, розширення інтерфейсу.