```
package Battleship;

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

public class gameboard {
	Random random = new Random();
	Scanner input = new Scanner(System.in);

	int currentPlayerIndex;
	Player[] PlayerArray = new Player[1];

	int size;
	static int length;
	String Difficulty = ""; // Difficulty Level Variable
	int PlayerCount = 1; // Player count Variable
	int BoardSize; // Score board Info

	// Difficulty Arrays (Board Size + 2, Missiles)
	private int[] BEGINNER = { 8, 30 }; // 6x6 Board
	private int[] STANDARD = { 11, 50 }; // 9x9 Board
	private int[] ADVANCED = { 14, 75 }; // 12x12 Board

	String SPACE_EMPTY = "-\t";
	String SPACE_MISS = "O\t";

	public void AskPlayerCount() {
		do {
			System.out.println("Please select number of players.");
			System.out.println("1. Single player");
			System.out.println("2. Play against the computer");

			System.out.print("Enter 1 for Single, 2 for vs Computer:
");
			String playerInput = input.nextLine();
			System.out.println();
			try {
				PlayerCount = Integer.parseInt(playerInput);
			} catch (NumberFormatException exception) {
				PlayerCount = -1;
			}
		} while (PlayerCount < 1 || PlayerCount > 2);
	}

	public void AskPlayerDifficulty() {
		// Getting Player input for Game Type.
		do {
			System.out.println("Please select your difficulty level.
\n\n   "
					+ "Level:\tGrid:\tMissles:");
			System.out.println("1. Beginner\t6x6\t30");
			System.out.println("2. Standard\t9x9\t50");
			System.out.println("3. Advanced\t12x12\t75\n");
			System.out.print("Enter 1 for Beginner, "
```

```
                        + "2 for Standard, 3 for Advanced: ");
                Difficulty = input.nextLine();
                System.out.println();

        } while ((!Difficulty.equals("1")) &&
(!Difficulty.equals("2"))
                    && (!Difficulty.equals("3")));
    }

    public void BuildPlayers() {
        for (int i = 0; i < PlayerCount; i++) {

            // Array to hold turn details
            Player player = new Player();

            // Setting Variable Size to Difficulty level
            if (Difficulty.equals("1")) {
                size = BEGINNER[0];
                player.turns = BEGINNER[1];
            }
            if (Difficulty.equals("2")) {
                size = STANDARD[0];
                player.turns = STANDARD[1];
            }
            if (Difficulty.equals("3")) {
                size = ADVANCED[0];
                player.turns = ADVANCED[1];
            }

            length = (size - 1);
            player.Player_Board = PopulateBoard(new
String[size][size]);

            player.SetupPlayer();
            player.Ship_Board = Create_Ship_Board();

            PlaceShips(player, player.Ship_Board);
            PlayerArray[i] = player;
        }
    }

    public void SetupGame() {

        //AskPlayerCount(); // add back in if AI is completed.
        AskPlayerDifficulty();
        BuildPlayers();

    }

    // Fills board with BLANK_EMPTY
    public String[][] PopulateBoard(String[][] BOARD) {
        for (int row = 0; row < length; row++) {
            for (int col = 0; col < length; col++)
                BOARD[row][col] = SPACE_EMPTY;
```

```
            }

            // Setting Labels for Top and Side Row
            for (int row = 0; row < length; row++)
                  BOARD[row + 1][0] = ((char) (((int) 'A') + row) + "\t");

            // Clearing Top-Left space.
            for (int col = 1; col < length; col++)
                  BOARD[0][col] = ((col) + "\t");
            BOARD[0][0] = " \t";
            return BOARD;
      }

      // Creates a second board to hold ship placement.
      public String[][] Create_Ship_Board() {
            String[][] Ship_Board = new String[size][size];
            Ship_Board = PopulateBoard(Ship_Board);
            return Ship_Board;
      }

      // Fills X and Y Axis with appropriate labels.
      public void PrintBoard(Player player, String BOARD[][]) {

            System.out.print("\n\n\n\t\tBATTLESHIP:\n" + "\tTotal Turns:
\t\t"
                        + player.GetTurns() + "\n" + "\tRemaining Turns:
\t"
                        + player.GetRem() + "\n" + "\tShips Remaining: \t"
                        + player.ships_remaining + "\n" + "\tAccuracy:
\t\t");
            System.out.printf("%2.2f%%", player.GetAcc());
            System.out.println("\n");
            for (int row = 0; row < length; row++) {
                  for (int col = 0; col < length; col++) {
                        System.out.print(BOARD[row][col]);
                  }
                  System.out.println();
            }
            System.out.println();
      }

      // Randomly Placing Ships
      public void PlaceShips(Player player, String[][] ShipBoard) {
            int row = 0;
            int col = 0;
            int LastRow = row;
            int LastCol = col;
            int Direction;
            int Pass = 0;

            for (int ship = 0; ship < 5; ship++) {
                  do {
                        // Determine if First and Last Cell are valid in
the array.
```

```
                        do {
                              Direction = 0;
                              Pass = 0;
                              row = random.nextInt(length);
                              col = random.nextInt(length);

                              LastRow = row;
                              LastCol = col;

                              // Direction Modifier
                              // 0 = UP, 1 = RIGHT, 2 = DOWN, 3 = LEEFT
                              Direction = random.nextInt(4);

                              // Set Last Space Based on Direction
                              if (Direction == 0)
                                    LastRow -=
player.ShipsArray[ship].spaces;
                              if (Direction == 1)
                                    LastCol +=
player.ShipsArray[ship].spaces;
                              if (Direction == 2)
                                    LastRow +=
player.ShipsArray[ship].spaces;
                              if (Direction == 3)
                                    LastCol -=
player.ShipsArray[ship].spaces;

                        } while (OutOfBounds(row) || OutOfBounds(col)
                                    || OutOfBounds(LastRow) ||
OutOfBounds(LastCol));

                        if (row > LastRow) // Assigning the smaller value
to row/col.
                        {
                              int a = row;
                              row = LastRow;
                              LastRow = a;
                        }

                        if (col > LastCol) {
                              int b = col;
                              col = LastCol;
                              LastCol = b;
                        }

                        // Checking if any part of the ship will be
position in an
                        // occupied space.
                        if (Direction == 0 || Direction == 2) {
                              Direction = 0;
                              if (CheckIfVertArrayEmpty(ShipBoard, row,
col,
                                          player.ShipsArray[ship].spaces)
== true)
```

```
                              Pass = 1;
                        }

                        if (Direction == 1 || Direction == 3) {
                              Direction = 1;
                              if (CheckIfSideArrayEmpty(ShipBoard, row,
col,
                                          player.ShipsArray[ship].spaces)
== true)
                              Pass = 1;
                        }

                  } while (Pass == 0);

                  // Add Ship to Ship_Board
                  if (Direction == 0)
                        AddShipToBoard(player, ShipBoard, row, col, ship);
                  if (Direction == 1)
                        AddShipToBoardSide(player, ShipBoard, row, col,
ship);
            }
      }

      // Determine is space is empty
      public boolean CheckEmpty(String[][] ShipBoard, int row, int col) {
            if (ShipBoard[row][col] == SPACE_EMPTY)
                  return true;
            else
                  return false;
      }

      // Cycles through array to see if any spaces are occupied.
      public boolean CheckIfVertArrayEmpty(String[][] ShipBoard, int row,
                  int col, int Ship_Length) {
            int check = 0;
            for (int i = 0; i < Ship_Length; i++)
                  if ((CheckEmpty(ShipBoard, row, col) == true)) {
                        row++;
                        check = i + 1;
                  } else
                        break;
            if (check == Ship_Length)
                  return true;
            else
                  return false;
      }

      public boolean CheckIfSideArrayEmpty(String[][] ShipBoard, int row,
                  int col, int Ship_Length) {
            int check = 0;
            for (int i = 0; i < Ship_Length; i++)
                  if ((CheckEmpty(ShipBoard, row, col) == true)) {
                        col++;
                        check = i + 1;
```

```
                } else
                        break;

        if (check == Ship_Length)
                return true;
        else
                return false;
    }

    // Adds ship to Ship_Board
    public void AddShipToBoard(Player player, String[][] ShipBoard, int
row,
                int col, int ship) {

        for (int i = 1; i <= player.ShipsArray[ship].spaces; i++) {
                ShipBoard[row][col] = player.ShipsArray[ship].icon;
                row++;
        }
    }

    public void AddShipToBoardSide(Player player, String[][] ShipBoard,
                int row, int col, int ship) {
        for (int i = 1; i <= player.ShipsArray[ship].spaces; i++) {
                ShipBoard[row][col] = (player.ShipsArray[ship].icon);
                col++;
        }
    }

    // Checks if first/last ship point is off the Array.
    public boolean OutOfBounds(int a) {
        return (a > (length - 1) || a <= 0);

    }

}
```