Battleship (Variation B)

Code Implementation

Module Block: **CS1701** Group Project Lectures and Tutorials
Assessment Block: **CS1809** Software Design
Tutor: Lela Koulouri
Group: Yellow01

Suraj Shaw

2023424

Requirement Specifications -

Core functionalities

1. The user plays against the computer.
2. The board has 100 positions (by default).
3. The positions of the board are marked as if the board is placed on the IV quadrant of graph i.e., the board starts from x=1 and y=1 from the top left corner.
4. The computer places 5 different ships and 2 monsters on the board randomly by the computer.
5. The board contains:
    1. 5 ships of the following types:
        i. Aircraft carrier: 5 squares long
        ii. Battleship: 4 squares long
        iii. Submarine: 3 squares long iv. Destroyer: 3 squares long
        v. Patrol Boat: 2 squares long 2.

    2 sea monsters of following types:
        i. Kraken: it will consume all of the points in the user's score at the time at which it is hit.
        ii. Cetus: it will cause all un-sunk ships on the board to move to different places on the board.
6. The user can shoot a position of the board by entering x and y coordinates of the board.
7. Validate user input, x,y with x $\in [1, 10]$ and y $\in [1, 10]$.
8. The shoot can result in either hit or miss. After each shot, the board is updated by a 'H' or 'M' on the position that was shot.
9. User is shown the updated board and updated statistics of the current game after every shot.
10. The programs keep a running score of the user. The score rules are as follows:
    1. Each shot (hit or miss) deducts one point from the score.
    2. Each hit adds one point.
    3. When a ship is sunk, a number of points equal to the length of that ship multiplied by 2 is added to the score
11. A ship sinks when all of its squares have been hit.
12. The game ends when user sinks all the ships or the chooses to quit the game.
13. At the end of the game, the user must be asked if they want to play again.

Additional functionalities

1. The game will have a score board.
2. The game has a radar functionality which will help the player to detect ships on board that will help them to take more accurate shots.
3. Additional player stats are shown like hit-to-miss ratio, number of miss(es), number of shots taken and also a list of sunked and unsunked ships.

Testing – Test Plans and Test Cases

Test Title : Testing the shoot() method.
Test Description : Testing all the valid inputs for the board i.e. in range of board size [0,10]

Input	Expected Results	Actual Result	Status (Pass/Fail)	Fix
0	Accept Input	Accept Input	Pass	-
1	Accept Input	Accept Input	Pass	-
5	Accept Input	Accept Input	Pass	-
10	Accept Input	Accept Input	Pass	-
-1	Decline Input	Decline Input	Pass	-
20	Decline Input	Decline Input	Pass	-

Test Title : Testing the ship placement on board
Test Description : Testing if all the ships and monsters are in the range of board and don't overlap with anything. This is tested by counting the empty position ("N") on board.

Test no.	Expected Results	Actual Result	Status (Pass/Fail)	Fix
1	"N" = 81	"N" = 86	Fail	Add ships immediately after creating so other ships find coordinates accordingly.
2	"N" = 81	"N" = 81	Pass	
3	"N" = 81	"N" = 81	Pass	
4	"N" = 81	"N" = 81	Pass	

Test Title : Testing the output of board
Test Description : Testing the output of board that is shown to user.

Test no.	Expected Results	Actual Result	Status (Pass/Fail)	Fix
1	Expected Output String	Output String	Fail	Replace every N character with "-"
2	Expected Output String	Output String	Fail	Adjust the spacing to deal with 1 digit and 2 digit numbers on the side
3	Expected Output String	Output String	Fail	Show "H" and "M" inside brackets("[]")
4	Expected Output String	Output String	Pass	-

Test Title : Testing the working of radar
Test Description : Radar is tested against 2 cases, (1) it should return *True* when at least a ship is present in range and (2) *False* when no ship is present in range.

Test no.	Expected Results	Actual Result	Status (Pass/Fail)	Fix
1	True, False	False, False	Fail	If condition error (use && instead of ||)
2	True, False	True, False	Pass	-

Algorithm and User Interface Design -

- The core functionalities are exactly those described in my report from assignment 2(listed above). Every core functionality has been tested to work as expected.

- Some of the additional functions have been omitted like game saving feature, NullPointException keep occurring when tried to implement the saving feature by importing Jackson dependency using maven due some error in eclipse so I decided to focus my time and effort on testing because testing has more importance than this feature.

- User Interface is exactly the same as mentioned in assignment 2 which is console based.

- The algorithm has been modified to follow more Object Oriented Programming rules to make the code much more efficient and shorter.

Source Code -

Java files list :

1. Main.java
2. Board.java
3. Ship.java
4. Monster.java
5. Radar.java
6. Score.java
7. Marshalling.java
8. GlobalMehods.java

1. Main.java

```
package battleship;

import java.io.File;
import java.io.IOException;

public class Main{
     public static void menu() {

          String menuOptions = "1) New Game\n"
                                    + "2) Continue\n"
                                    + "3) Load Game\n"
                                    + "4) Options\n"
                                    + "5) Help\n"
                                    + "6) Quit\n";
          System.out.println(menuOptions);
          System.out.print("Select an option : ");
     }

     public static void loadGame() {

     }


     public static void checks() {
          File gameJSON = new File("savedgames.json");
           if (!gameJSON.exists()) {
                try {
                     if(!gameJSON.createNewFile()){
                          System.out.println("File cannot be
created");
                     }
```

```
                    } catch (IOException e) {
                         e.printStackTrace();
                    }
               System.out.println("Does not Exists");
             }
     }


     public static void main(String[] args) {

             checks();
             boolean dontExit = true;
             boolean canCont = false;
             Game game = new Game();
             while (dontExit) {
                    GlobalMethods.heading();
                    menu();
                    int option = GlobalMethods.inputInt();
                    switch (option) {

                    case 1 :
                         game.newGame();
                         canCont = true;
                         game.play();
//                       game.saveCurrentGame();
                         break;

                    case 2 :
                         if (canCont) {
                               game.play();
                         } else {
                               System.out.println("Start a game first to
continue.");
                         }

                    case 3 :
                         // game.loadGame();
                         // game.play();
                         System.out.println("This is coming soon. Devs
are working for u. :)");
                         break;

                    case 5 :
                         System.out.println("Instructions of the game :
");
```

```
                      System.out.print("\nThe computer places the
following five types of ship of different lengths randomly on a 10 by
10 board. The types of ship and their lengths are shown below:\n"
                                 + "   1. Aircraft carrier: 5 squares
long\n"
                                 + "   2. Battleship: 4 squares
long\n"
                                 + "   3. Submarine: 3 squares long\n"
                                 + "   4. Destroyer: 3 squares long\n"
                                 + "   5. Patrol Boat: 2 squares
long\n\n"
                                 + "The player cannot see the ship
position"
                                 + "The player ‘shoots’ in order to
hit and sink all the ships of the computer. A ship sinks when all of
its squares have been hit.\n"
                                 + "The player should enter the X,Y
coordinates of a shot. And enter x = 0 and y = 0 to go back to main
menu.\n"
                                 + "The game will display the
appropriate message according to the outcome of the shot:\n"
                                 + "   - My ship was hit!\n"
                                 + "   - You missed!\n"
                                 + "   - You sank my [ship type]!\n\n"
                                 + "Scoring rules :\n"
                                 + "   1. -(1) for every shoot.\n"
                                 + "   2. +(1) for every hit.\n"
                                 + "   3. +(2 * ship_size) for sinking
every ship.\n\n"
                                 + "The board also contains two sea
monsters, Kraken and Cetus, which will be annoyed if hit."
                                 + "If Kraken is hit, it will consume
all of the points in the player’s score at the time at which it is
hit. "
                                 + "If Cetus is hit, it will cause all
un-sunk ships on the board to move to different places on the board."
                                 + "Each sea monster will be placed on
a random square (which is not occupied by a ship) immediately after
the computer has placed the ships on the board.\n"
                                 + "Tip : Annoying monsters is not
always a bad idea.\n\n");
                      System.out.println("Press enter to
continue...\n");
                      GlobalMethods.input();
                      break;
```

```
                case 6 :
                     dontExit = false;
                     break;

                default :
                     System.out.println("Please Enter a valid
option");
                     break;
                }
           }

           GlobalMethods.scan.close();
      }
}
```

2. Board.java

```
     package battleship;

     import java.awt.Point;
     import java.security.NoSuchAlgorithmException;
     //import java.util.Arrays;

     public class Board {
     private String[][] boardMatrix;

     // Create a new board
     public void create(int size) {
          boardMatrix = new String[size][size];

          for (int i = 0; i < size; i++) {
               for (int j = 0; j < size; j++) {
                    boardMatrix[i][j] = "N";
               }
          }
     }

     // Print the board for debugging
     //   void print() {
     //        System.out.println("\n");
     //        for (int i = 0; i < this.boardMatrix.length; i++) {
     //
     System.out.println(Arrays.toString(boardMatrix[i]));
     //        }
     //   }

     // output board
```

```
public void print() {
      int gameBoardLength = this.boardMatrix.length;
      System.out.println("\n");
      System.out.print("    ");
      for(int i = 0; i < gameBoardLength; i++ ) {
            System.out.print(i + 1 + "   ");
      }
      System.out.println("\n");

      for(int y = 0; y <  gameBoardLength; y++) {
            if (y < 9) {
                  System.out.print(y + 1 + "  ");
            } else {
                  System.out.print(y + 1 + " ");
            }

            for(int x = 0; x <  gameBoardLength; x++) {
                  String position = this.boardMatrix[y][x];
                  if (position == "H" || position == "M") {
                        System.out.print("[" + position + "]" + "
");
                  } else {
                        System.out.print("[-]"  + " ");
                  }

            }

            System.out.println("\n");
      }
}

// Add ships to board
public void addShipToBoard(Ship s) {
      try {
            s.newPosition();
      } catch (NoSuchAlgorithmException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
      }
      if (isValidShipPosition(s)) {
            for(int i=0; i < s.getSize(); i++) {
                  // Condition to place ships only in board matrix
                  if (s.getPosition()[i].x >= 0 &&
s.getPosition()[i].y >= 0
```

```
                    && s.getPosition()[i].x <
this.boardMatrix.length && s.getPosition()[i].y <
this.boardMatrix.length ) {

boardMatrix[s.getPosition()[i].y][s.getPosition()[i].x] =
s.getName();
                }
            }
        } else {
            addShipToBoard(s);
        }
}

public void addMonsterToBoard(Monster m) {
        try {
            m.newPosition();
        } catch (NoSuchAlgorithmException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }
        if(isValidMonsterPosition(m)) {
            if (m.getPosition().x >= 0 && m.getPosition().y >= 0
                && m.getPosition().x < this.boardMatrix.length
&& m.getPosition().y < this.boardMatrix.length ) {

boardMatrix[m.getPosition().y][m.getPosition().x] = m.getName();
            }
        }
}

// Update the board
public void update(Point hitPoint, boolean attackStat) {
        if (attackStat) {
            boardMatrix[hitPoint.y - 1][hitPoint.x - 1] = "H";
        } else {
            boardMatrix[hitPoint.y - 1][hitPoint.x - 1] = "M";
        }


}

// Position is valid i.e don't overlap with other ships or go
out of board
public boolean isValidShipPosition(Ship s) {
        for (Point p : s.getPosition()) {
            if (!boardMatrix[p.y][p.x].equals("N")) {
                    return false;
                }
            }
            return true;
        }

        public boolean isValidMonsterPosition(Monster m) {
            if
        (!boardMatrix[m.getPosition().y][m.getPosition().x].equals("N"))
        {
                return false;
            }
            return true;
        }

        public void clearBoard() {
            for(int i = 0; i < this.boardMatrix.length; i++){
                for(int j = 0; j < 10; j++){
                    this.boardMatrix[i][j] = "N";
                }
            }
        }

        // getters and setters
        /**
         * @return the boardMatrix
         */
        public String[][] getBoardMatrix() {
            return boardMatrix;
        }

        /**
         * @param boardMatrix the boardMatrix to set
         */
        public void setBoardMatrix(String[][] boardMatrix) {
            this.boardMatrix = boardMatrix;
        }
        }
```

3. Ship.java

```
package battleship;

import java.awt.Point;
import java.security.NoSuchAlgorithmException;
import java.security.SecureRandom;
import java.util.ArrayList;
import java.util.List;
import java.util.Random;


// 1. Aircraft carrier: 5 squares long : A
// 2. Battleship: 4 squares long : B
// 3. Submarine: 3 squares long : S
// 4. Destroyer: 3 squares long : D
// 5. Patrol Boat: 2 squares long : P

/**
 * This class will hold functions for ship.
*/
public class Ship {
      private String name;       // Code name of ship
      private String fullName;   // Full name of ship
      private int size;                // Size of ship
      private Point[] position;  // Coordinates of ship
      private int health;   // Health of ship = (size - number of
times the ship got shot)

      /**
       * Ship Constructor.
       *
       * @param s Size of ship
       * @param n Code name of ship
       * @param f Full name of ship
       */
      public Ship(int s, String n, String f) {
          this.size = s;
          this.setName(n);
          this.setFullName(f);
          this.health = size;
        }

      /**
       * Generates new position for the ships.
       *
       * @return Point[] - Array of ships's coordinates.
       * @throws NoSuchAlgorithmException
       */
      public Point[] newPosition() throws NoSuchAlgorithmException {
           Random rand = SecureRandom.getInstanceStrong(); //
SecureRandom is preferred to Random
           position = new Point[size];
```

```
            for(int i = 0; i < position.length; i++) {
                position[i] = new Point();
            }

            // Create the ship
            position[0].x = rand.nextInt(10);
            position[0].y = rand.nextInt(10);

            int x;
            int y;

            List<Integer> orientations =
validOrientations(position[0]);
            int r = rand.nextInt(orientations.size());
            if (orientations.get(r) == 0) {
                  x = -1;
                  y = 0;
            }

            else if (orientations.get(r) == 1) {
                  x = 0;
                  y = -1;
            }

            else if (orientations.get(r) == 2) {
                  x = 1;
                  y = 0;
            }

            else {
                  x = 0;
                  y = 1;
            }

            for (int i = 1; i<size; i++) {
                  position[i].x = position[i-1].x + x;
                  position[i].y = position[i-1].y + y;
            }

            return (position);

      }

      /**
       * Generates valid orientations for the ship.
       *
```

```
	 * Random orientation
	 * 0 = left
	 * 1 = top
	 * 2 = right
	 * 3 = bottom
	 *
	 * @param pos Current random of ship.
	 * @return List<Integer> - Array of ships's possible
orientation(s).
	 */
	List<Integer> validOrientations(Point pos) {
		List<Integer> o = new ArrayList<>();

		if(pos.x - size >= 0) o.add(0);

		if(pos.y - size >= 0) o.add(1);

		if(pos.x + size < 10) o.add(2);

		if(pos.y + size < 10) o.add(3);


		return o;
	}

	/**
	 * Print coordinates of the ship.
	 */
	void printCoord() {
		for (int i = 0; i < this.size; i++) {
			System.out.println("x -> " + position[i].x + " and y
-> " + position[i].y);
		}
	}


	public boolean exists() {
		return false;

	}

	public boolean destroyed() {
		// Return true if exists otherwise false
		if (health == 0) {
			return true;
		}
		return false;
	}

	// getters and setters
	/**
	 * @return the name
	 */
	public String getName() {
		return name;
	}

	/**
	 * @param name the name to set
	 */
	public void setName(String name) {
		this.name = name;
	}

	/**
	 * @return the fullName
	 */
	public String getFullName() {
		return fullName;
	}

	/**
	 * @param fullName the fullName to set
	 */
	public void setFullName(String fullName) {
		this.fullName = fullName;
	}

	/**
	 * @return the size
	 */
	public int getSize() {
		return size;
	}

	/**
	 * @param size the size to set
	 */
	public void setSize(int size) {
		this.size = size;
	}
	/**
	 * @return the position
	 */
	public Point[] getPosition() {
		return position;
	}

	/**
	 * @param position the position to set
	 */
	public void setPosition(Point[] position) {
		this.position = position;
	}

	/**
	 * @return the health
	 */
	public int getHealth() {
		return health;
	}

	/**
	 * @param health the health to set
	 */
	public void setHealth(int health) {
		this.health = health;
	}

}
```

4. Game.java

```
package battleship;

import java.awt.Point;
import java.util.ArrayList;
import java.util.List;

//import java.io.FileInputStream;
//import java.io.FileWriter;
//import java.io.IOException;
//import java.io.InputStream;
//import java.nio.charset.StandardCharsets;
//import org.json.JSONArray;
//import org.json.JSONObject;
//import org.json.JSONTokener;
//
//import com.fasterxml.jackson.databind.ObjectMapper;
```

```
/**
 * This class holds of the main logics of the game.
 */
public class Game {
private Board cBoard;

// Initialising all the ships and monsters
private List<Ship> ships = new ArrayList<>();
private List<Monster> monsters = new ArrayList<>();

private List<Point> shootCoord = new ArrayList<>();

private List<Score> leaderboard = new ArrayList<>();

private int score;    // Score of the player
private int hits;     // Total hits
private int miss;     // Total miss

private Radar radar;

// Constructor
public Game() {
      score = 0;
      hits = 0;
      miss = 0;
}

/**
  * Creates a new game and places the ship randomly.
*/
public void newGame() {
      GlobalMethods.heading();
      intro();

      this.cBoard = new Board();
      this.cBoard.create(10);
      // cBoard.print();
      fillBoard();
      cBoard.print();

      radar = new Radar();


}
```

```
public void fillBoard() {
      // Creating ships
      this.ships.add(new Ship(5, "A", "Aircraft carrier"));  //
Aircraft carrier
      cBoard.addShipToBoard(ships.get(ships.size() - 1));

      this.ships.add(new Ship(4, "B", "Battleship"));  //
Battleship
      cBoard.addShipToBoard(ships.get(ships.size() - 1));

      this.ships.add(new Ship(3, "S", "Submarine"));   //
Submarine
      cBoard.addShipToBoard(ships.get(ships.size() - 1));

      this.ships.add(new Ship(3, "D", "Destroyer"));   //
Destroyer
      cBoard.addShipToBoard(ships.get(ships.size() - 1));

      this.ships.add(new Ship(2, "P", "Patrol Boat")); // Patrol
Boat
      cBoard.addShipToBoard(ships.get(ships.size() - 1));

      // Creating monsters
      this.monsters.add(new Monster("K", "Kraken"));
      cBoard.addMonsterToBoard(monsters.get(monsters.size() -
1));

      this.monsters.add(new Monster("C", "Cetus"));
      cBoard.addMonsterToBoard(monsters.get(monsters.size() -
1));
}

/**
  * Starts the game play.
*/
public void play() {
      List<String> sunked = new ArrayList<>();
      List<String> unsunked = new ArrayList<>();
      for (Ship s : this.ships) {
            if (s.getHealth() == 0) {
            sunked.add(s.getFullName());
            } else {
                  unsunked.add(s.getFullName());
            }
      }
```

```
	while (true) {
		System.out.print("Choose option\n"
				+ " - Enter 1 to use radar.\n"
				+ " - Enter 2 to start shooting.\n"
				+ " - Enter 3 to go back to main menu\n");
		int option = GlobalMethods.inputInt();
		if (option == 1) {
			System.out.println("Remaining radars : " +
this.radar.getRem());
			if (this.radar.getRem() > 0) {
				System.out.println("Coordinates for
radar.");
				Point p = shoot();
				if (!(p.x == 0) && !(p.x == 0)) {
					if (this.radar.use(cBoard, p)) {
						System.out.println("A ship is
detected in the range of radar.");
					} else {
						System.out.println("No ship
detected in the range of radar.");
					}
				}
			} else {
				System.out.println("You have used all the
available radars.");
			}

		} else if (option == 2) {
			while (startShooting(sunked, unsunked)) {
				this.cBoard.print();
				printStats(sunked, unsunked);
			}
		} else if (option == 3) {
			break;
		} else {
			System.out.println("Enter a valid option.");
		}

	}


}

public boolean startShooting(List<String> sunked, List<String>
unsunked) {
        Point coord = shoot();
        this.score--;   // decrement in score for every shoot

        // Keep a record of shoot coordinates
        for (Point sc : shootCoord) {
              if (coord.getX() == sc.getX() && coord.getY() ==
sc.getY()) {
                    GlobalMethods.slowPrint("You already shot this
target.", 100);
                    return true;
              }
        }

        this.shootCoord.add(coord);

        if (coord.x == 0 || coord.y == 0) {
              return false;
        } else if (finished()) {
              System.out.print("Enter player name to save on
leaderboard : ");
              String playerName = GlobalMethods.input();
              leaderboard.add(new Score(playerName, this.score,
this.hm_ratio()));
              exit();
              return false;
        }

        boolean attackStat = isHit(coord);
        String hitStat;

        if(attackStat) {
              if (this.cBoard.getBoardMatrix()[coord.y - 1][coord.x
- 1] == "K") {
                    hitStat = "You have annoyed Kraken. It will take
all your score.";
                    this.score = 0;
                    this.cBoard.update(coord, attackStat);
              } else if (this.cBoard.getBoardMatrix()[coord.y -
1][coord.x - 1] == "C") {
                    hitStat = "You have annoyed Cetus. It will
unsunk and reposition all your ships.";
                    this.cBoard.clearBoard();
                    fillBoard();
                    shootCoord.clear();
              } else {
                    hitStat = "My ship was hit!";
```

```
                for (Ship s : this.ships) {
                     if (s.getName() != null &&
s.getName().equals(this.cBoard.getBoardMatrix()[coord.y -
1][coord.x - 1])) {
                   s.setHealth(s.getHealth() - 1);
                   if (s.destroyed()) {
                        this.score += s.getSize() * 2;
                        sunked.add(s.getFullName());
                        System.out.println(String.format("You
sank my %s!", s.getFullName()));
                   } else {
                        unsunked.add(s.getFullName());
                   }
                   return true;
                   }
                }
                this.cBoard.update(coord, attackStat);
           }

      } else {
           hitStat = "You missed!";
           this.cBoard.update(coord, attackStat);
      }

      GlobalMethods.slowPrint("\n" + hitStat, 100);

      return true;
}

/**
  * To check if shoot was a hit or miss.
  *
  * @param hitPoint  the coordinates of the shoot
  * @return boolean - hit or miss
*/
public boolean isHit(Point hitPoint) {
      if (cBoard.getBoardMatrix()[hitPoint.y - 1][hitPoint.x -
1].equals("N")) {
           this.miss++;
           return false;
      } else{
           this.score++;
           this.hits++;
           return true;
      }
}
```

```
/**
  * Take input of the shoot coordinates form user.
  *
  * @return Point - shoot coordinates
*/
public Point shoot() {
     Point hitPoint = new Point(1,1);

     // Input within the range of [1,10]
     do {
          if (hitPoint.x < 0 || hitPoint.x > 10 || hitPoint.y <
0 || hitPoint.y > 10) {
               System.out.println("Please enter valid
coordinates i.e within range [1,10]");
          }

          System.out.println("Enter x and y coordinates to
shoot. Enter x = 0 or y = 0 to exit shooting.");
          System.out.print("Enter x:  ");
          hitPoint.x = GlobalMethods.inputInt();
//             scan.nextLine(); // It consumes the \n character

          System.out.print("Enter y:  ");
          hitPoint.y = GlobalMethods.inputInt();
//             scan.nextLine(); // It consumes the \n character

     } while (hitPoint.x < 0 || hitPoint.x > 10 || hitPoint.y <
0 || hitPoint.y > 10);
     return hitPoint;
}

/**
  * Introduction of a new game.
*/
void intro() {
     String stat = "Hello, welcome to Battleship. This is a
game of just "
                    + "luck and little bit of logic. If you
think you are lucky, "
                   + "and have some common sense. PLAY THIS
GAME NOW.\n\n"
                   + "The computer will randomly place the
ships on the board,"
                   + "vertically or horizontally, taking care
that no ship "
```

```
                    + "overlaps with another ship or is out of
the bounds of the 10 by 10 board."
                    + "You have ‘shoots’ in order to hit and
sink all the ships of the computer.\n"
                    + "A ship sinks when all of its squares have
been hit.\n";
     GlobalMethods.slowPrint(stat, 0);

}

/**
  * Hit-to-miss ratio.
  *
  * @return float - hit-to-miss ratio
*/
float hm_ratio() {
     if (this.hits == 0 || this.miss == 0) {
          return 0;
     }

     return (this.hits/this.miss);
}

/**
  * Exit
*/
void exit() {
     if (finished()) {
          System.out.println("You have successfully completed
the game. ");
     } else {
          System.out.println("Do you want to save the progress?
(Y/N)");
          String doSave = GlobalMethods.inputOnlyString();
          if (doSave != null && (doSave.equals("Y") ||
doSave.equals("y"))) {
//                   saveGameToJSON();
               System.out.println("Your progress have beenn
saved. ");
          }
     }
     System.out.println("Thank you for playing the game.");
}

public void printStats(List<String> sunked, List<String>
unsunked) {
```

```
System.out.println(String.format("******************************
***********************************************\n"
                            + "[Score : %d] [Shot(s) : %d]\n"
                            + "[Hit(s) : %d] [Miss(es) : %d]
[hit-to-miss ratio : %.2f]\n\n"
                            + "Ships sunked : " + sunked + "\n"
                            + "Ships unsunked : " + unsunked +
"\n"
                            +
"**************************************************************
**************"
                            , this.score, this.hits+this.miss,
this.hits, this.miss, hm_ratio()));
}

public void showLeaderboard() {
     if (this.leaderboard.size() > 0) {
           System.out.print("Scoreboard : \n");

System.out.print("*********************************************
******************************\n");
           for (Score l : leaderboard) {
                System.out.println(String.format("Player  ---
Score"
                                                            +
"%d  ---  %s"
                                                            ,
l.getPlayerName(), l.getScore()));
           }

System.out.print("*********************************************
******************************\n");
     }

}

/**
 *
 *
 * @param
 * @return
 */
public boolean finished() {
     int totalHealth = 0;
     for (Ship s : ships) {
            totalHealth += s.getHealth();
        }

        if (totalHealth == 0) return true;
        return false;
    }

    // COMING SOON ---- CODE NEED FIXING BELOW THIS LINE ---

    /**
     * Save games details
     */
//    public void saveGameToJSON(String gameName) {
//         JSONArray games = new JSONArray();
//        JSONObject obj = new JSONObject();
//
////        System.out.println("The game will be saved by name :
");
////        String gameName = scan.nextLine();
//
//        obj.put("name", gameName);
//
//        // Translating board data into JSON format
//        JSONArray board = new JSONArray();
//        for (String[] i : this.cBoard.getBoardMatrix()) {
//          JSONArray arr = new JSONArray();
//          for (String j : i) {
//            arr.put(j); // or some other conversion
//          }
//          board.put(arr);
//        }
//        obj.put("board", board);
//
//        // Translating ships data into JSON format
//        JSONArray shipArray = new JSONArray();
//        for (Ship s : this.ships) {
//         JSONObject ship = new JSONObject();
//         ship.put("name", s.getName());
//         ship.put("fullName", s.getFullName());
//         ship.put("size", s.getSize());
//         JSONArray positionArray = new JSONArray();
//         for (Point p : s.getPosition()) {
//              JSONArray pointArray = new JSONArray();
//              pointArray.put(p.x);
//              pointArray.put(p.y);
//              positionArray.put(pointArray);
//          }
//          ship.put("position", positionArray);
//          ship.put("health", s.getHealth());
//          shipArray.put(ship);
//         }
//         obj.put("ships", shipArray);
//
//         // Translating player stats into JSON format
//         obj.put("score", this.score);
//         obj.put("hits", this.hits);
//         obj.put("miss", this.miss);
//
//         games.put(obj);
//
//         try (FileWriter file = new
FileWriter("savedgames.json")) {
//             file.write(games.toString());
//         } catch (IOException e) {
//             e.printStackTrace();
//         }
//     }

//     public void saveGameToJSON(String gameName) {
//          ObjectMapper mapper = new ObjectMapper();
//
//     }
//     public void saveCurrentGame() {
//          saveGameToJSON("lastGame");
//
//     }
//
//     public void loadGame() throws IOException {
//          InputStream is = new
FileInputStream("savedgames.json");
//          String output = new String(is.readAllBytes(),
StandardCharsets.UTF_8);
//
//
//         JSONTokener tokener = new JSONTokener(is);
//         JSONObject object = new JSONObject(tokener);
//
//         JSONArray games = new JSONArray(tokener);
//
//         System.out.println("Id  : " + object.getLong("id"));
//         System.out.println("Name: " +
object.getString("name"));
```

```
//        System.out.println("Age : " + object.getInt("age"));
//
//        System.out.println("Courses: ");
//        JSONArray courses = object.getJSONArray("courses");
//        for (int i = 0; i < courses.length(); i++) {
//            System.out.println("  - " + courses.get(i));
//        }
//
//    }
}
```