WARSHIPS

OOP-Final Project

GROUP 2

DEVELOPER TEAM

ITITIU22157
Hồ Thành Tiến

ITITIU22184
Nguyễn Thế Vinh

ITITIU22205
Tôn Quang Tấn

ITCSIU22276
Ngô Quang Hải

WORK DISTRIBUTION

Name	Task
Hồ Thành Tiến	Report / CPU / Tester/ Fix Bug
Nguyễn Thế Vinh	Report / Gameplay Function / UI - UX
Tôn Quang Tấn	Report / UML, Use Case diagram / Classes and Objects for location
Ngô Quang Hải	Report / UI-UX / Frame

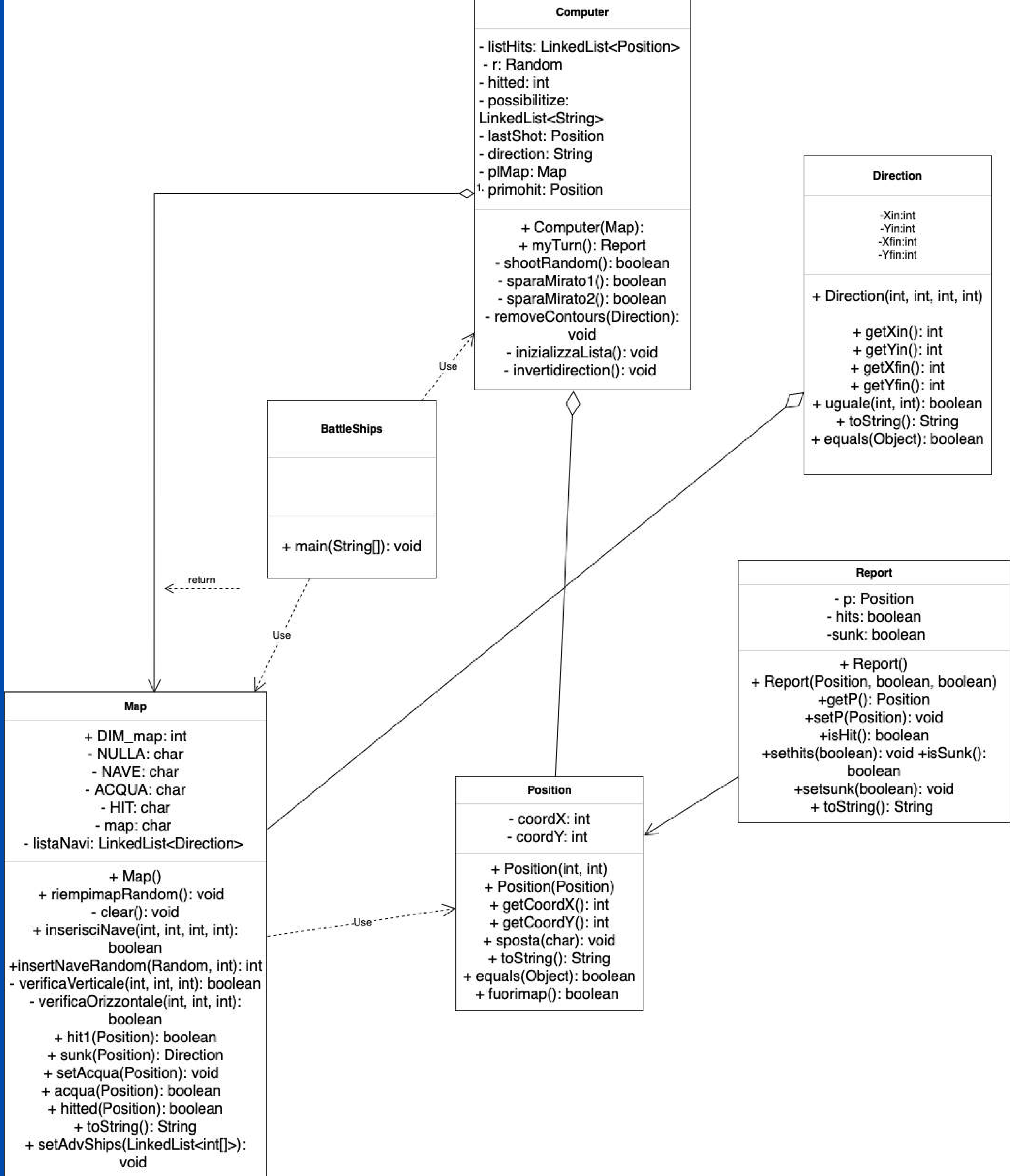
CONTENT

Overview

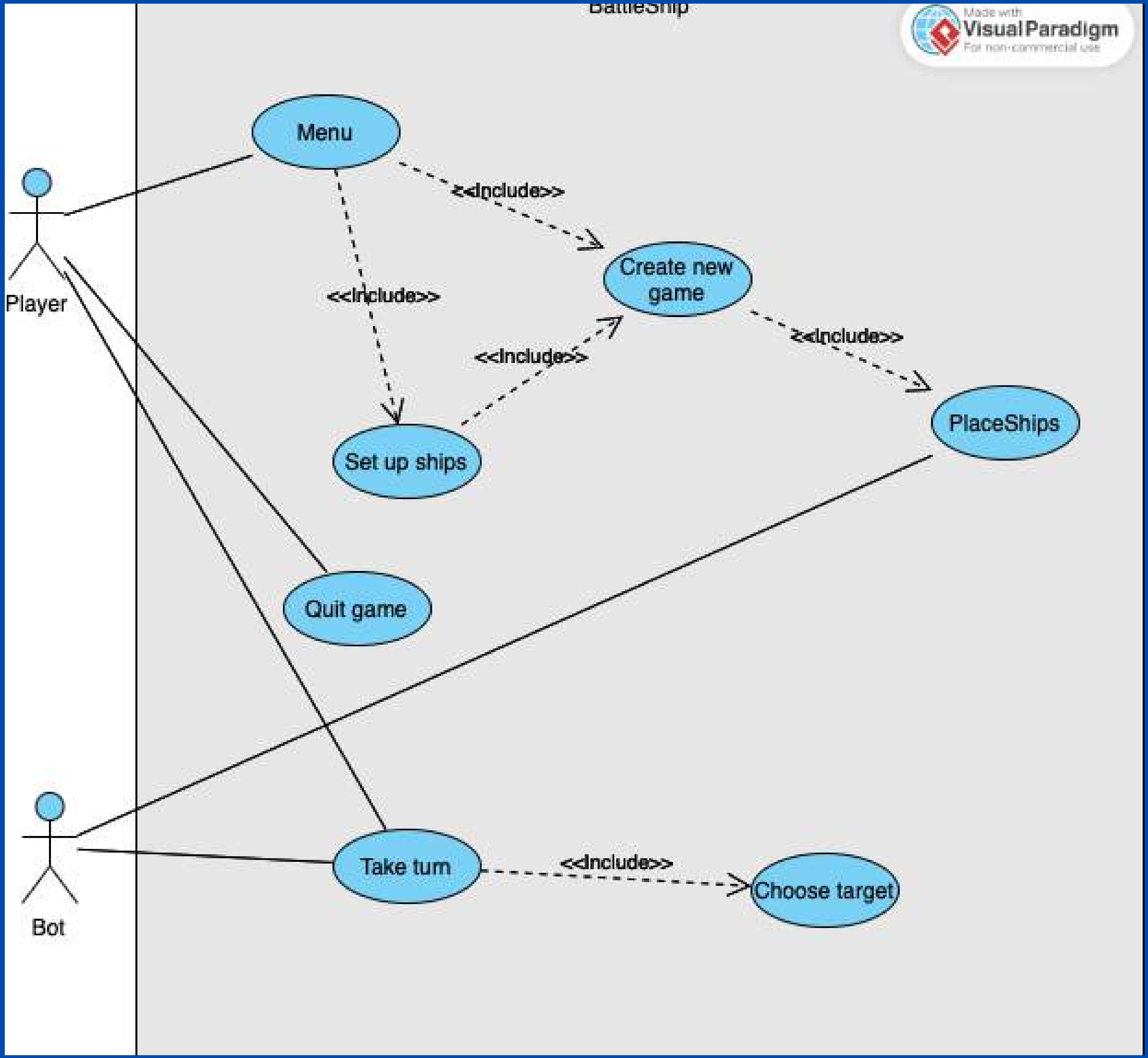
Warship is a Java-based turn-based strategy game inspired by "Battleship." Players place ships on a 10x10 grid and take turns launching attacks to sink their opponent's fleet. The game features an interactive UI, dynamic ship placement, AI opponents, and real-time status updates for an engaging gameplay experience.

The classic Battleship game

UML CLASS DIAGRAM


Computer
- listHits: LinkedList<Position>
- r: Random
- hitted: int
- possibilitize: LinkedList<String>
- lastShot: Position
- direction: String
- plMap: Map
- primohit: Position
+ Computer(Map):
+ myTurn(): Report
- shootRandom(): boolean
- sparaMirato1(): boolean
- sparaMirato2(): boolean
- removeContours(Direction): void
- inizializzaLista(): void
- invertidirection(): void
1.
Direction
-Xin:int
-Yin:int
-Xfin:int
-Yfin:int
+ Direction(int, int, int, int)
+ getXin(): int
+ getYin(): int
+ getXfin(): int
+ getYfin(): int
+ uguale(int, int): boolean
+ toString(): String
+ equals(Object): boolean
Use
BattleShips
+ main(String[]): void
return
Use
Report
- p: Position
- hits: boolean
-sunk: boolean
+ Report()
+ Report(Position, boolean, boolean)
+getP(): Position
+setP(Position): void
+isHit(): boolean
+sethits(boolean): void +isSunk(): boolean
+setsunk(boolean): void
+ toString(): String
Map
+ DIM_map: int
- NULLA: char
- NAVE: char
- ACQUA: char
- HIT: char
- map: char
- listaNavi: LinkedList<Direction>
+ Map()
+ riempimapRandom(): void
- clear(): void
+ inserisciNave(int, int, int, int): boolean
+insertNaveRandom(Random, int): int
- verificaVerticale(int, int, int): boolean
- verificaOrizzontale(int, int, int): boolean
+ hit1(Position): boolean
+ sunk(Position): Direction
+ setAcqua(Position): void
+ acqua(Position): boolean
+ hitted(Position): boolean
+ toString(): String
+ setAdvShips(LinkedList<int[]>): void
Position
- coordX: int
- coordY: int
+ Position(int, int)
+ Position(Position)
+ getCoordX(): int
+ getCoordY(): int
+ sposta(char): void
+ toString(): String
+ equals(Object): boolean
+ fuorimap(): boolean
Use

USE CASE DIAGRAM



Objective

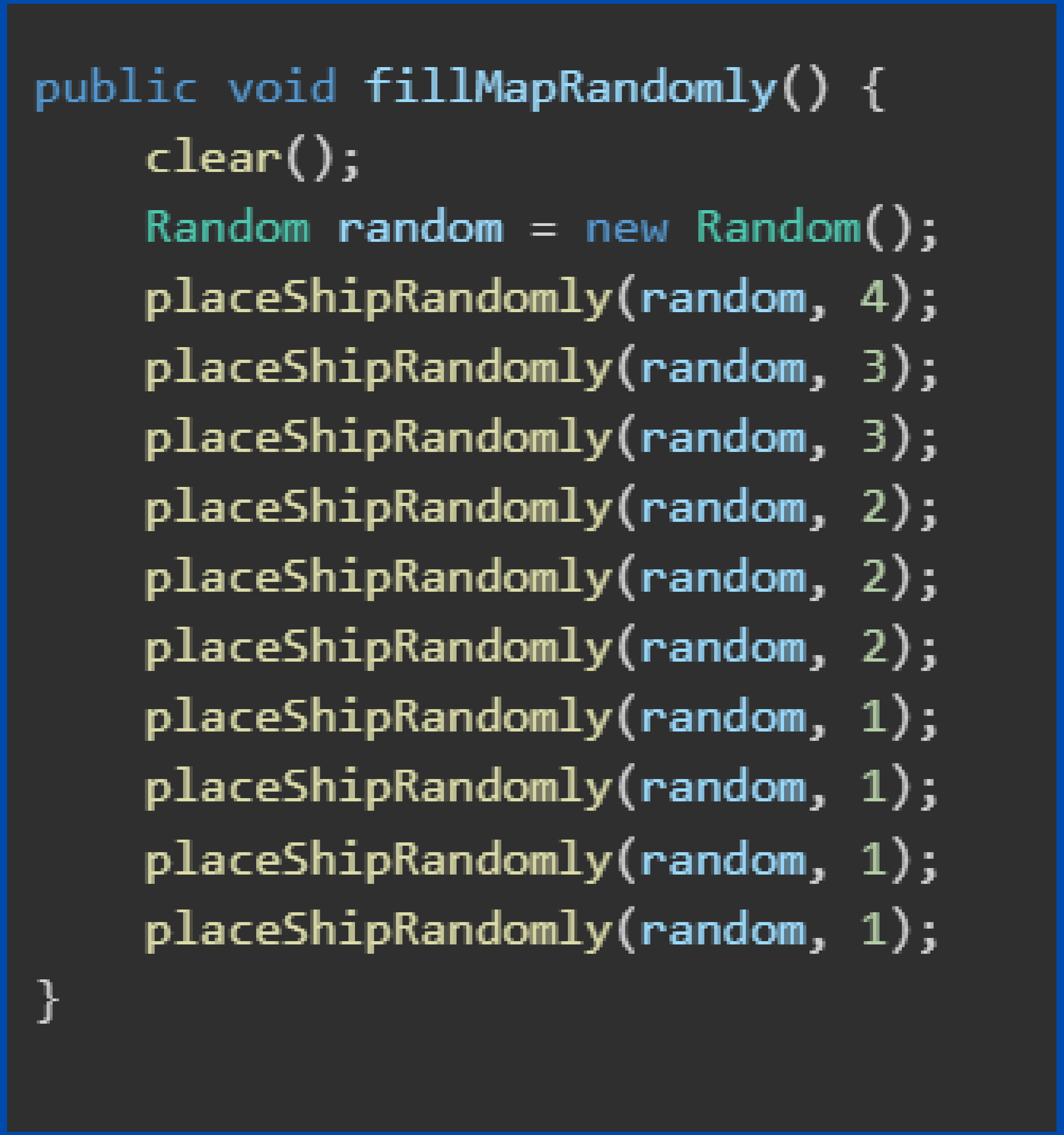
- **Demonstrate OOP Concepts: Show how OOP principles are applied in the game.**
- **Highlight Learning: Demonstrate the understanding of OOP through the creation and development of the game.**
- **Present Engaging Gameplay: Showcase a fun and playable game that effectively utilizes OOP principles.**

NOTABLE CLASSES

Computer class

Map class

MAP CLASS



```java
public void fillMapRandomly() {
    clear();
    Random random = new Random();
    placeShipRandomly(random, 4);
    placeShipRandomly(random, 3);
    placeShipRandomly(random, 3);
    placeShipRandomly(random, 2);
    placeShipRandomly(random, 2);
    placeShipRandomly(random, 2);
    placeShipRandomly(random, 1);
    placeShipRandomly(random, 1);
    placeShipRandomly(random, 1);
    placeShipRandomly(random, 1);
}
```

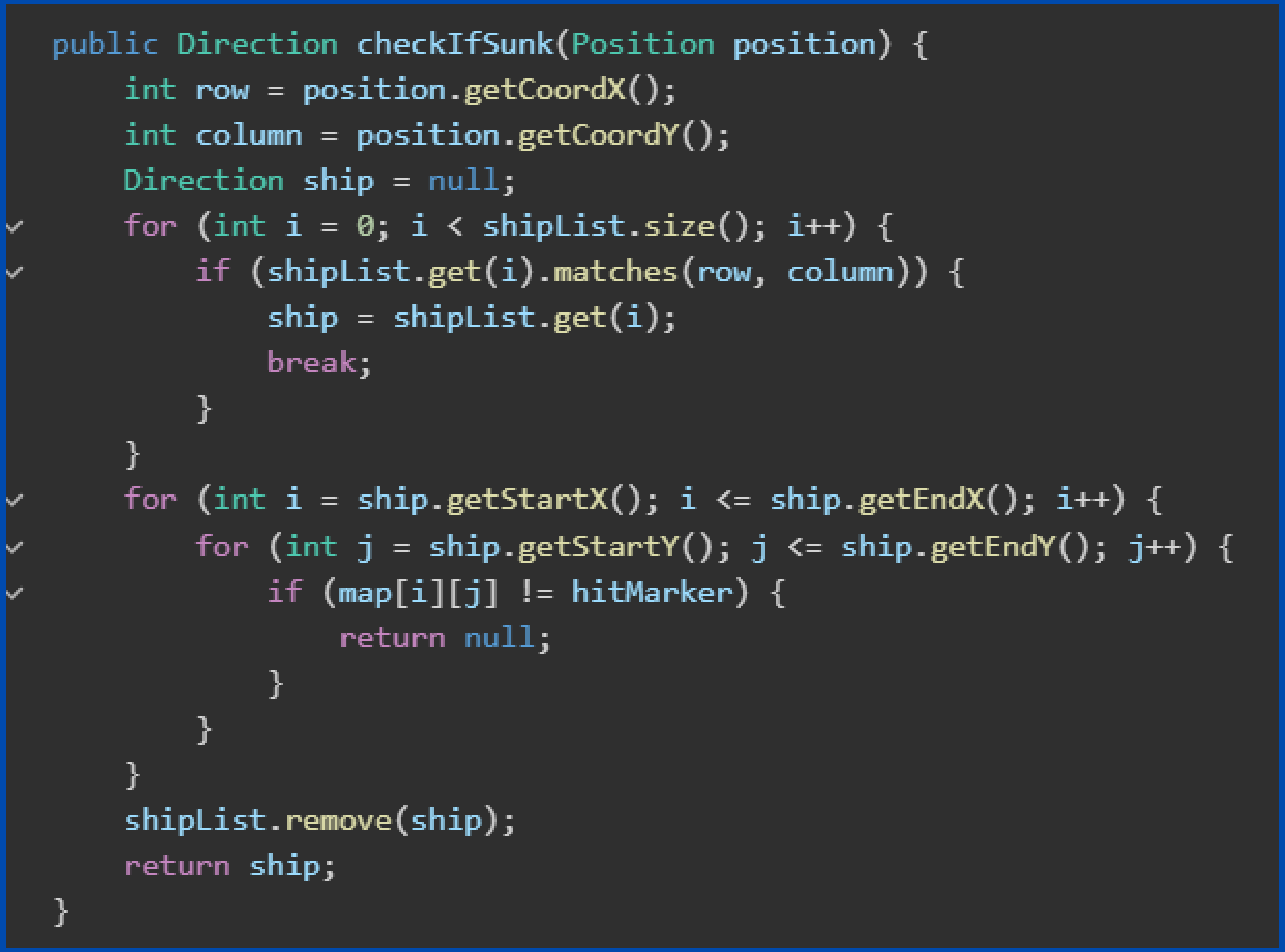
riempimapRandom

Purpose: Fills the map with randomly placed ships of varying sizes.

Notable Features:

- Ensures a variety of ship sizes (1x1, 2x1, 3x1, 4x1) are placed.
- Uses insertNaveRandom to handle random placement.

MAP CLASS



```java
public Direction checkIfSunk(Position position) {
    int row = position.getCoordX();
    int column = position.getCoordY();
    Direction ship = null;
    for (int i = 0; i < shipList.size(); i++) {
        if (shipList.get(i).matches(row, column)) {
            ship = shipList.get(i);
            break;
        }
    }
    for (int i = ship.getStartX(); i <= ship.getEndX(); i++) {
        for (int j = ship.getStartY(); j <= ship.getEndY(); j++) {
            if (map[i][j] != hitMarker) {
                return null;
            }
        }
    }
    shipList.remove(ship);
    return ship;
}
```

checkIfSunk

Purpose: Checks if a ship is fully destroyed (all its positions are hit).

Notable Features:

- Iterates over the ship's positions to verify all segments are hit.
- Removes the ship from the list if it is sunk.

Hit=0

Select a random position on the map. => The shot position is recorded in lastShot.=> Checks for a hit.

If hit:

Hit is incremented to 1.
Check if the ship is sunk using.

If sunk:

Update **Report** with setsunk(true).
Remove surrounding positions.
Reset hitted and direction.

If not sunk:

Store the first hit position.
Initialize possible directions.
Return the Report with the shot position and hit status.

Hit=1

Shoot in one of the four directions around primohit.
Record the shot position in lastShot.
Check for a hit using.
If hit:
    Increment hitted to 2.
    Set possibilitize to null.
    Check if the ship is sunk using plMap.sunk().
    Handle sunk ship as in the previous case.
Return the Report.

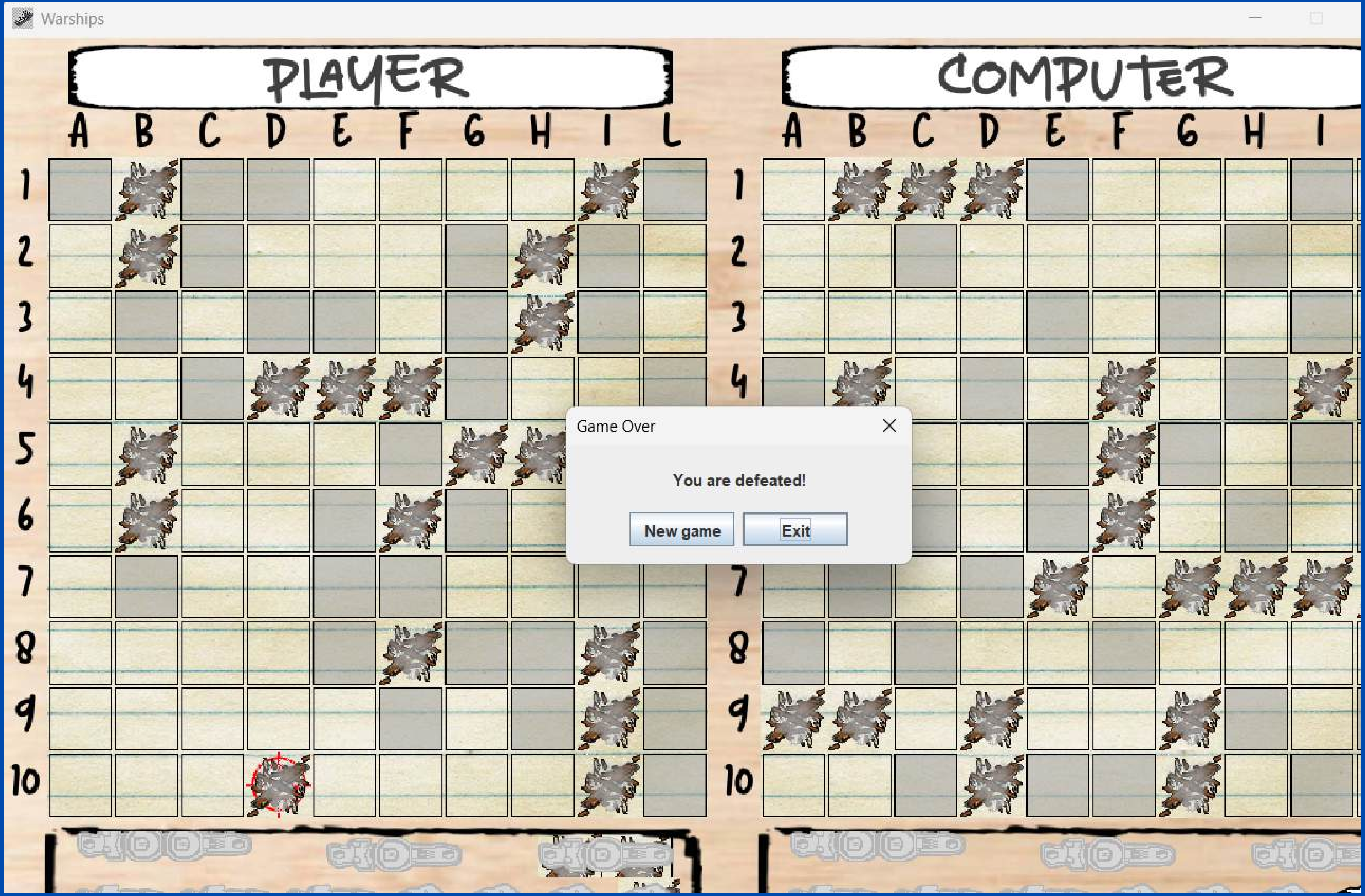
Hit>=2

Continue shooting in the current direction.
If the current direction is blocked, invertidirection() reverses the direction.
Record the shot position in lastShot.
Check for a hit using plMap.hit1().
If hit:

```
    Increment hitted.
    Check if the ship is sunk using plMap.sunk().
    Handle sunk ship as in previous cases.
```

Return the Report.

Demo



You are defeated!

THE END