Battleship (Variation B)

SOFTWARE DESIGN

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

Suraj Shaw

2023424

Table of Content:

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 3 levels to make the game more challenging:
    1. Easy: Board is 10x10 i.e., total 100 positions.
    2. Medium: Board is 20x20 i.e., total 400 positions.
    3. Hard: Board is 30x30 i.e., total 600 positions.
2. The game will have a "Sherlock mode" modes:
    1. On: User can only see the board 10, 20 and 30 times in easy, medium and hard level respectively.
    2. Off: User will see the board after every shoot.

3. Saving current game option will be given to user to save the game and play it later to make the game more addictive.
4. The game will have an auto-save feature that saves game after every 5 shoots.
5. The game will have a leader board with top 5 scorers.
6. The game will add a radar functionality to make game more interesting to play.

Implemented functions:

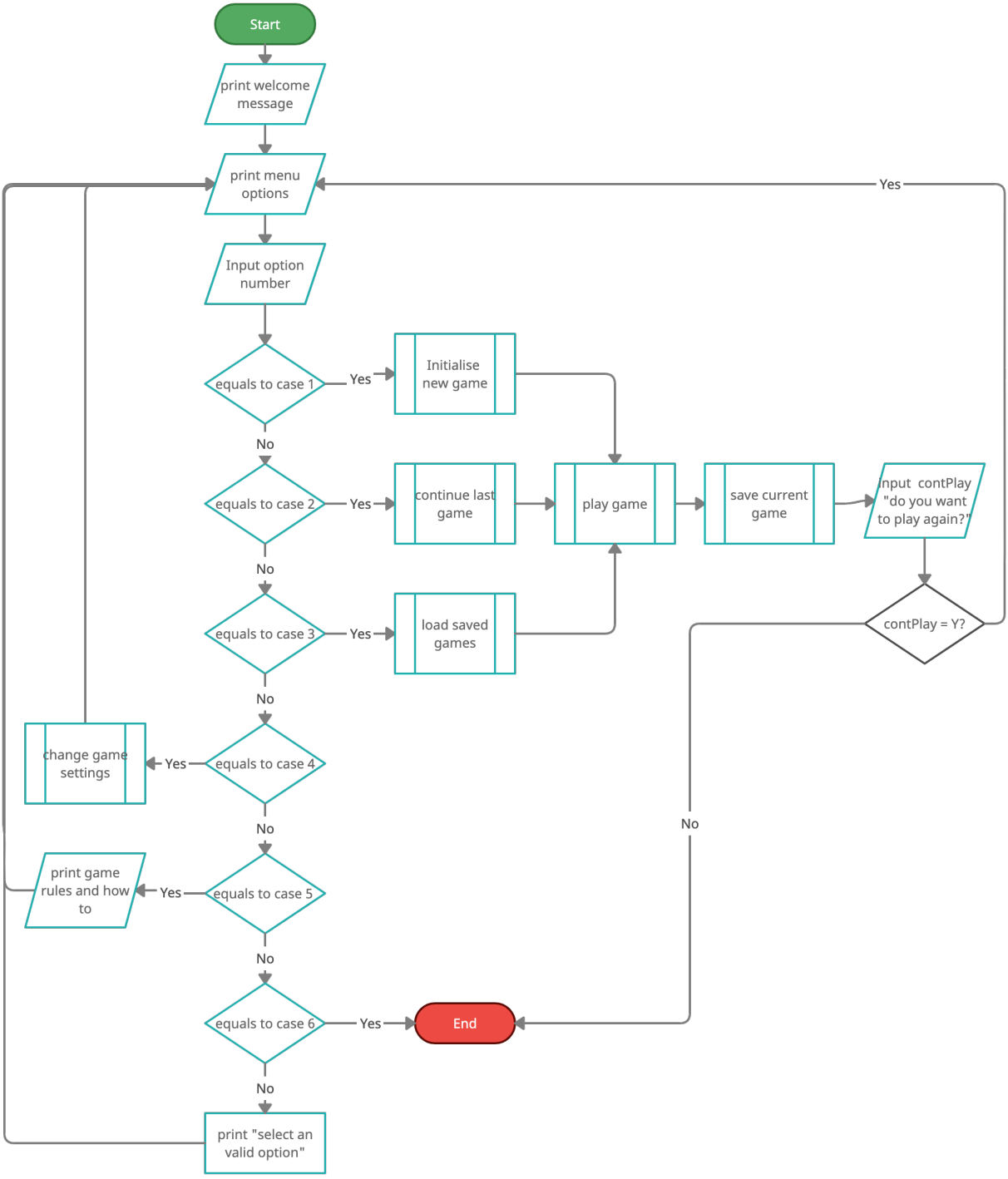
All the core functionalities have been shown in this report in all forms i.e. pseudocodes, flowcharts and the prototype. However, the additional functionalities have not been showed, it's still in alpha phase.

There are also certain options showed in main menu like load game, continue and save game that are showed in pseudocode and flowcharts but those are just dummy options.

Algorithm Design -

1. Start of the program - Main file

Flowchart:



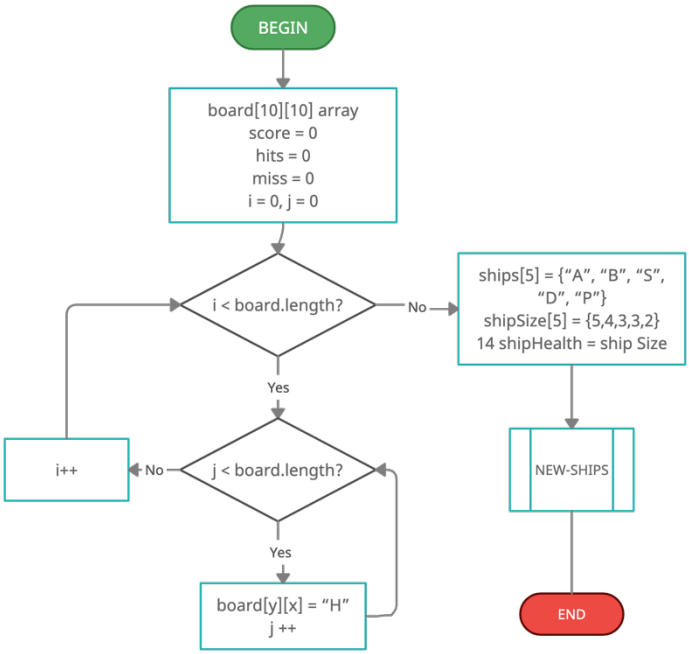
Pseudocode:

```
1  BEGIN
2  Print welcome message
3  option = 0
4  WHILE option is not equal to 6:
5  OUTPUT menu options
6  INPUT option number
7    CASEWHERE option is:
8         1: Call subprocess INITIALISE-NEW-GAME
9         2: Call subprocess CONTINUE-LAST-GAME
10        3: Call subprocess LOAD-SAVED-GAMES
11        4: Call subprocess CHANGE-GAME-SETTINGS
12        5: Print how to play the game and rules of the game.
13        6: Quit the game
14        OTHERWISE: Print "Select a valid game option"
15   ENDCASE
16   Call subprocess PLAY-GAME
17   Call subprocess SAVE-CURRENT-GAME
18   INPUT playAgain
19   IF playAgain equals "N":
20        option = 6
21   ENDIF
22 ENDWHILE
```

2. INITIALISE-NEW-GAME

This subprocess initaliases a new game by creating variables nescessary for the game.

Flowchart:

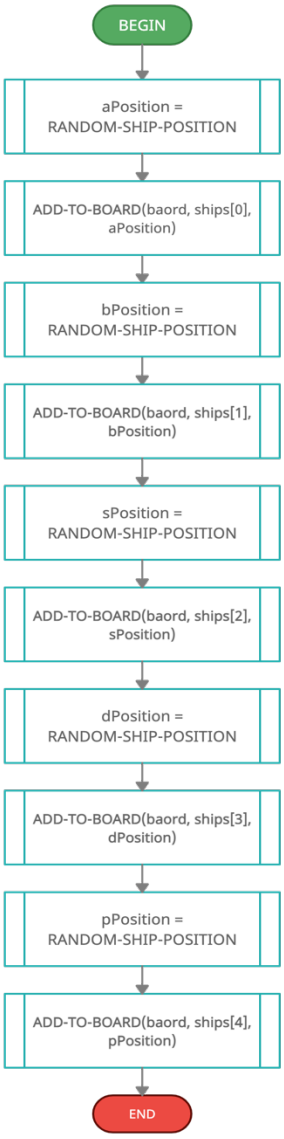


Pseudocode:

```
1  BEGIN
2  let board[10][10] be a new array
3  score = 0
4  hits = 0
5  miss = 0
3  // Initialising board with all
4  // positions as N (N is Nothing)
5  FOR i = 0 to board.length - 1
6 FOR j = 0 to board.length –
7      board[i][j] = "N"
8 ENDFOR
9  ENDFOR
10 // Initialising ships and
11 // their codename
12 ships[5] = {"A", "B", "S", "D", "P"}
13 shipSize[5] = {5,4,3,3,2}
14 shipHealth = ship Size
15 Call subprocess NEW-SHIPS
16 END
```

3. NEW-SHIPS

Flowchart :



Pseudocode:

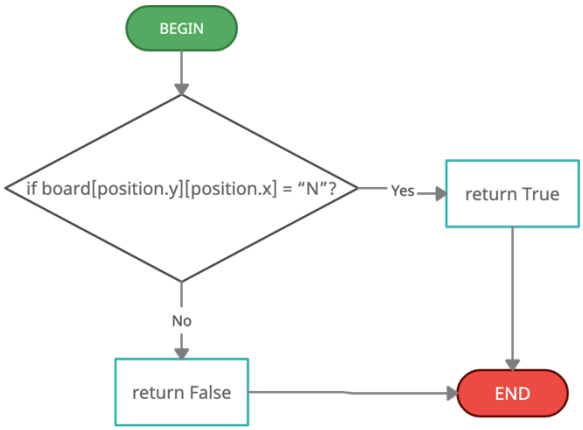
```
1  BEGIN
2  Point aPosition[aSize] = Call subprocess RANDOM-
    SHIP-POSITION(aSize)
3  Call subprocess ADD-TO-BOARD(baord, ships[0],
    aPosition)
4  Point bPosition[bSize] = Call subprocess RANDOM-
    SHIP-POSITION(bSize)
5  Call subprocess ADD-TO-BOARD(board, ships[1],
    bPosition)
6  Point sPosition[sSize] = Call subprocess RANDOM-
    SHIP-POSITION(sSize)
7  Call subprocess ADD-TO-BOARD(board, ships[2],
    sPosition)
8  Point dPosition[dSize] = Call subprocess RANDOM-
    SHIP-POSITION(dSize)
9  Call subprocess ADD-TO-BOARD(board, ships[3],
    dPosition)
10 Point pPosition[pSize] = Call subprocess RANDOM-
    SHIP-POSITION(pSize)
11 Call subprocess ADD-TO-BOARD(board, ships[4],
    pPosition)
12 END
```

4. VALID-POSITION(position)

This subprocess finds if the position is valid(empty) on board or not.

Flowchart:

Pseudocode:

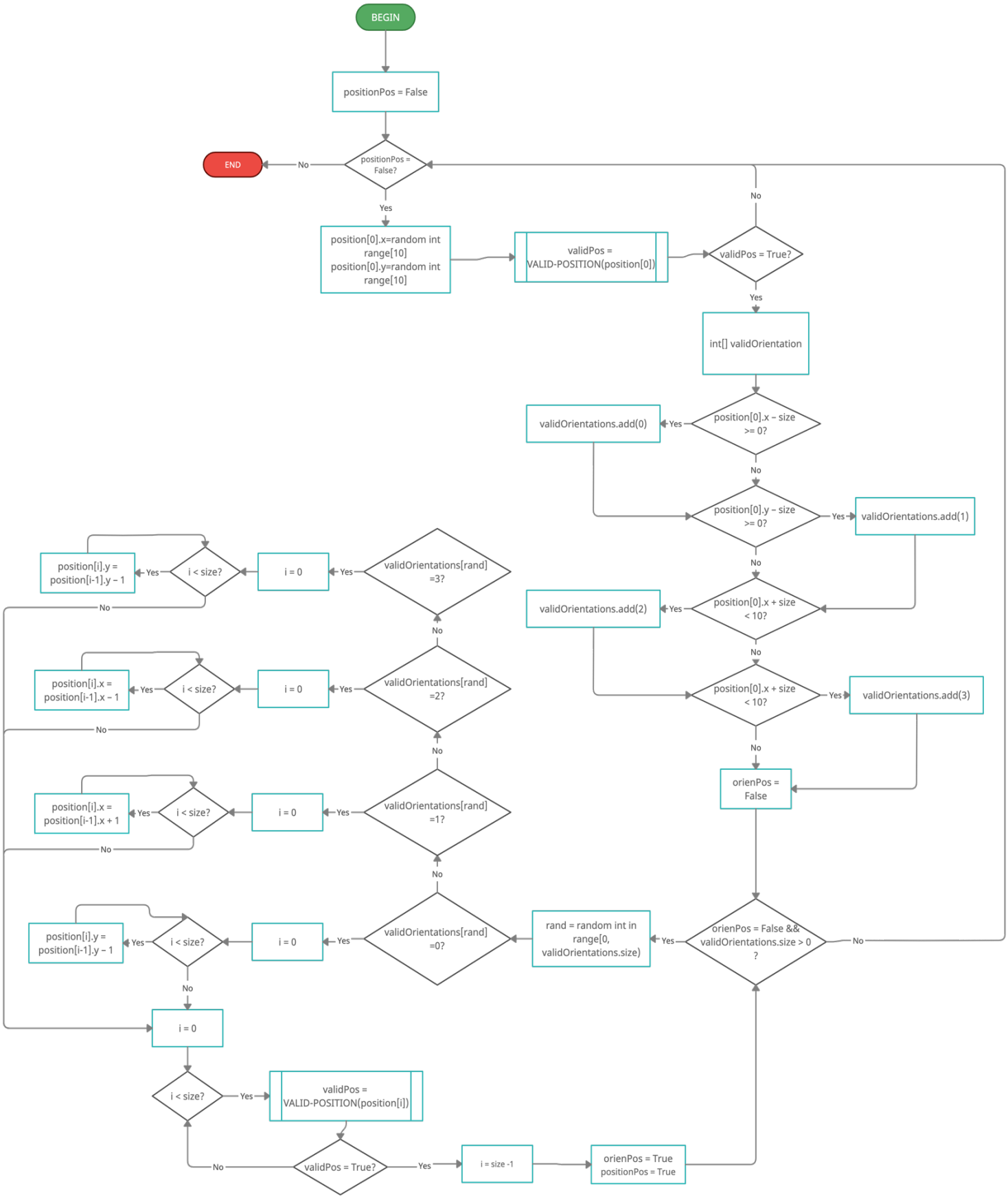



```
1 BEGIN
2 IF board[position.y][position.x] = “N”
3       RETURN True
4 ENDIF
5 RETURN False
6 END
```

5. RANDOM-SHIP-POSITION(size)

This subprocess generates a valid possiton array for ships.

Flowchart:



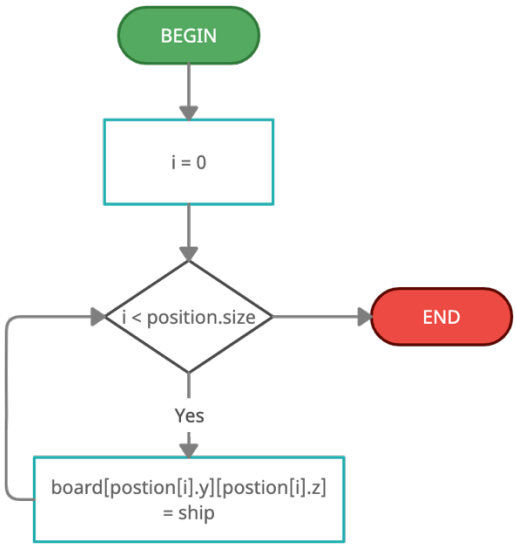
Pseudocode:

```
1  BEGIN
2  let position[size] be a new point array
3  positionPos = False
4  WHILE positionPos = False
5    position[0].x = generate a new random integer in range [0,10)
6    position[0].y = generate a new random integer in range [0,10)
7    IF VALID-POSTION(position[0]) = True
8         let validOrientations be a new int array
9         IF position[0].x – size >= 0
10             validOrientations.add(0)
11        ENDIF
12        IF position[0].y – size >= 0
13             validOrientations.add(1)
14        ENDIF
15        IF position[0].x + size < 10
16             validOrientations.add(2)
17        ENDIF
18        IF position[0].y – size < 10
19             validOrientations.add(3)
20        ENDIF
21        orienPos = False
22        WHILE orienPos = False AND validOrientations.size > 0
23        rand = new random integer in range [0, validOrientations.size)
24             IF validOrientations[rand] = 0
25                  FOR i=0 to size-1
26                       position[i].x = position[i-1].x – 1
27                  ENDFOR
28             ELSEIF validOrientations[rand] = 1
29                  FOR i=0 to size-1
30                       position[i].y = position[i-1].y – 1
31                  ENDFOR
32             ELSEIF validOrientations[rand] = 2
33                  FOR i=0 to size-1
34                       position[i].x = position[i-1].x + 1
35                  ENDFOR
36             ELSEIF validOrientations[rand] = 3
37                  FOR i=0 to size-1
38                       position[i].y = position[i-1].y + 1
39                  ENDFOR
40             ENDIF
41             FOR i=0 to size – 1
42                  IF VALID-POSITION(position[i]) = True
43                       IF i = size -1
44                            positionPos = True
45                            orienPos = True
45                  ENDIF
46             ENDFOR
47        ENDWHILE
48   ENDIF
50 ENDWHILE
51 RETURN position[] array
52 END
```

6. ADD-TO-BOARD(board, ship, positon[])

This subprocess adds ships to the board.

Flowchart:



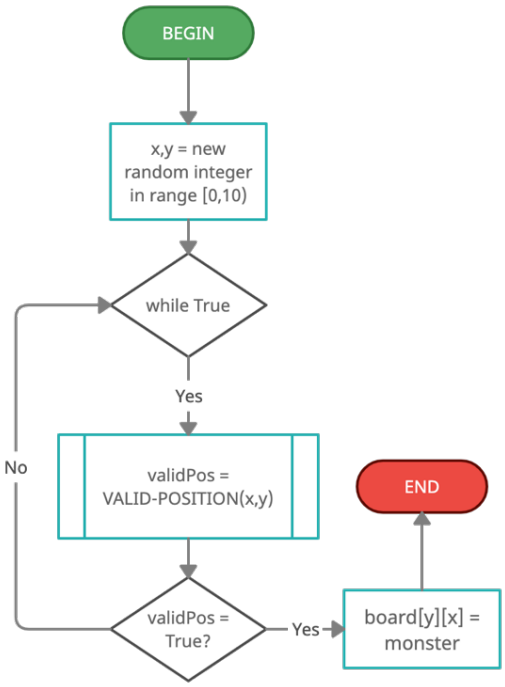
Pseudocode:

```
1 BEGIN
2 FOR i=0 to postion.size - 1
3     board[postion[i].y][postion[i].z]
= ship
4 ENDFOR
5 END
```

7. NEW-MONSTER(monster)

This subprocess adds monsters with random position on the board.

Flowchart:



Pseudocode:

```
1  BEGIN
2  x = generate a new random integer in range
    [0,10)
3  y = generate a new random integer in range
    [0,10)
4  WHILE True
5    IF VALID-POSITION(y,x) = True
6         board[y][x] = monster
7         Break
7    ENDIF
8  ENDWHILE
9  END
```

8. PLAY-GAME

This subprocess handles all the game logic.

Flowchart:

Pseudocode:

```
1  BEGIN
2  contPlay = True
3  WHILE contPlay = True
4    WRITE In-game options
5    READ option
6    IF option = “Shoot”
7         ContShoot = True
```

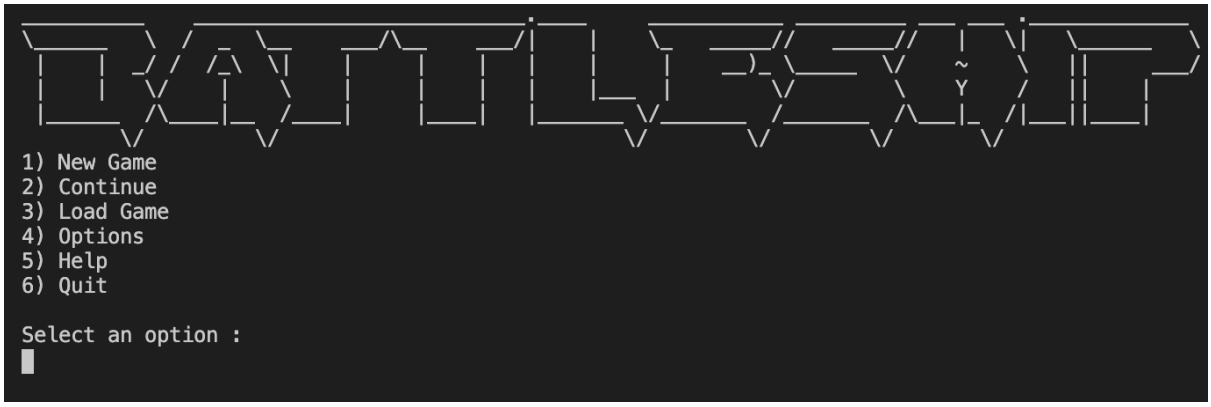
```
8         WHILE contShoot = True
9              WRITE board
10             READ x and y shooting coordinates
11             // Checks for correct user input
12             IF x>=0 AND y>=0 AND x<=10 AND y<=10
13                  IF x=0 OR y=0
14                       contShoot = False
15                  ELSE
16                       x-=1 and y-=1
17                       score--
18                       IF board[y][x] != “N”
19                            IF board[y][x] = “K”
20                                 WRITE “You have annoyed kraken!”
21                                 score = 0
22                                 Call Subprocess NEW-MONSTER(“K”)
23                            ELSEIF board[y][x] = “C”
24                                 WRITE “You have annoyed Cetus!”
25                                 FOR i = 0 to board.length - 1
26                                      FOR j = 0 to board.length–1
27                                      board[i][j] = “N”
28                                      ENDFOR
29                                 ENDFOR
30                                      Call subprocess NEW-SHIPS
31                                      Call Subprocess NEW-MONSTER(“C”)
32                                      Call Subprocess NEW-MONSTER(“K”)
33                                 ELSE // Ship is hit
34                                 WRITE “My ship was hit”
35                                 hits++
36                                 score++
37                                 board[y][x] = “H”
38                                 totalHealth = 0
39                                 FOR i=0 to 4   // Checks the health of all ships
40                                      IF shipHealth[i] = 0
41                                           WRITE “My ships[i] is destroyed”
42                                      ELSE
43                                           totalHealth += shipHealth[i]
44                                      ENDIF
45                                 ENDFOR
46                                 IF totalHealth = 0
47                                      WRITE  finished game message
48                                      contShoot = False
49                                      contPlay = False
50                                 ELSE
51                                           WRITE board and stats(score, hits, miss,
     hit-to-miss ratio, sunked and unsunked ships)
52                                 ENDIF
53                            ENDIF
54                       ELSE
55                            WRITE “It was a miss”
56                            score–
57                            miss++
58                       ENDIF
59                  ENDIF
60             ELSE
61                  WRITE “Enter valid shooting coordinates in range[0,10]”
62             ENDIF
63        ENDWHILE
64   ELSEIF option = “Quit”
65        contPlay = False
66   ELSE
67        WRITE “Enter valid option”
```

```
68   ENDIF
69 ENDWHILE
70 END
```

User Interface Design (‘Prototype’) -

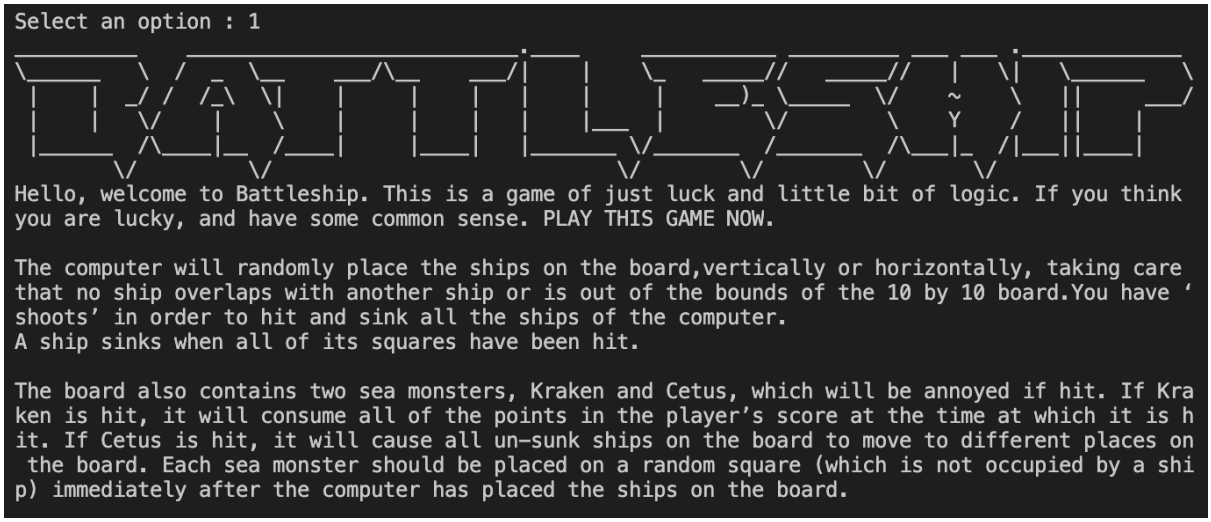
This is a CLI (Command Line Interface) based game. So everything is done on the terminal

Home Page :

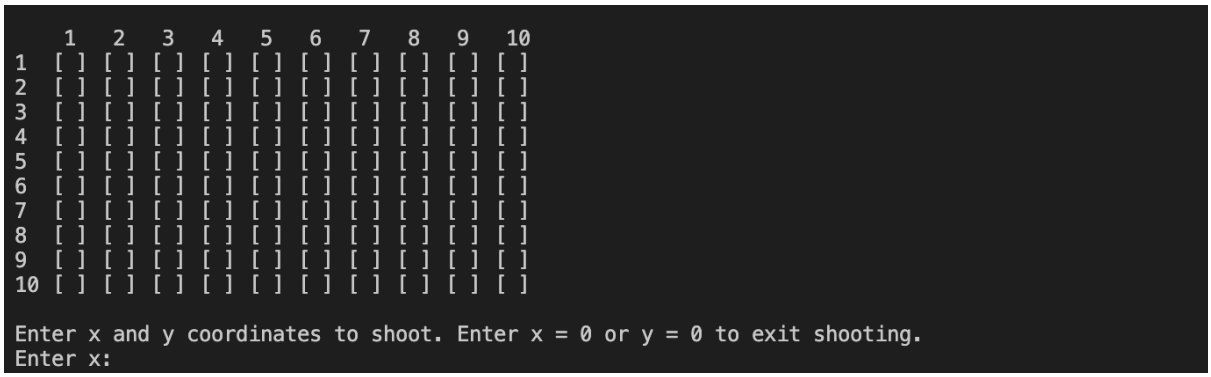


Home page appears when the game starts. It contains a heading made by using figlet. The menu has some beta functions which are not usable like option 2 and 3.
By entering the number 1 it will create a new game. And will let the user play.

New Game:



When new game option is selected. It gives a brief introduction of the game to help player understand the game so that they are able to play it without any difficulty.



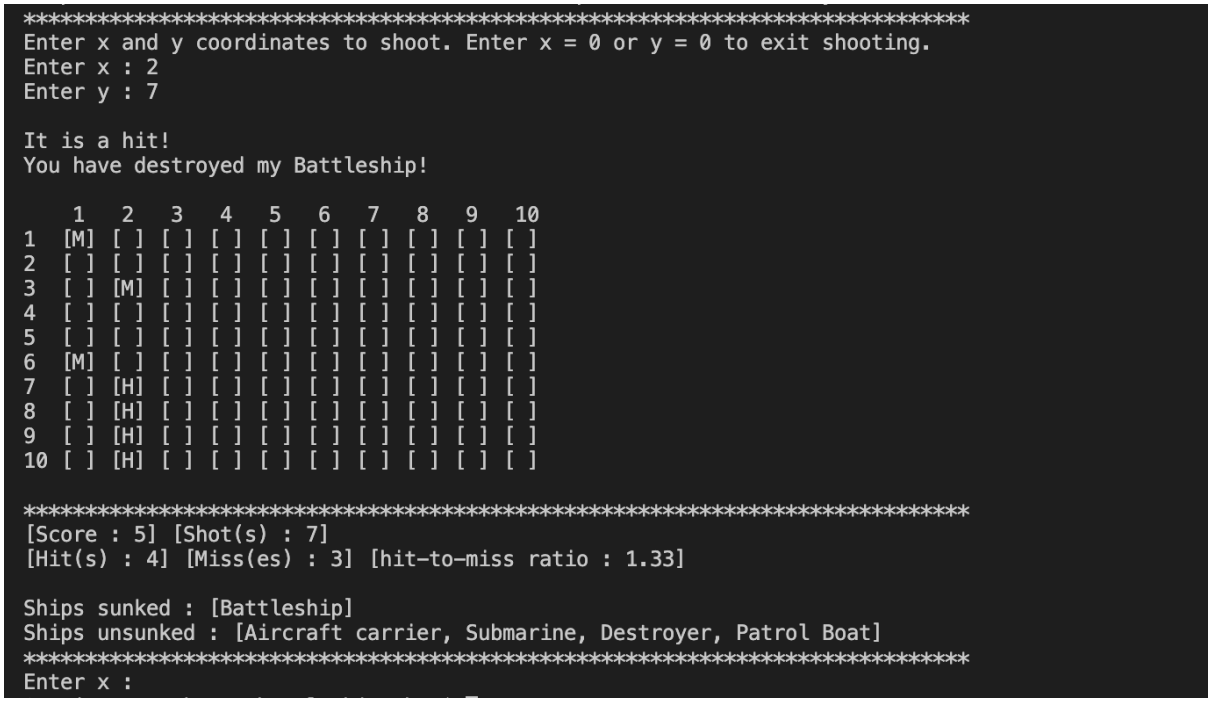
A empty board is shown in the beginning. This will be constantly updated after every shoot. The board is marked with coordinates numbers where the horizontal axis is x-axis and vertical axis is y axis.

Shooting:

```
Enter x and y coordinates to shoot. Enter x = 0 or y = 0 to exit shooting.
Enter x:  1
Enter y:  1

You missed!

    1   2   3   4   5   6   7   8   9   10
1  [M] [ ] [ ] [ ] [ ] [ ] [ ] [ ] [ ] [ ]
2  [ ] [ ] [ ] [ ] [ ] [ ] [ ] [ ] [ ] [ ]
3  [ ] [ ] [ ] [ ] [ ] [ ] [ ] [ ] [ ] [ ]
4  [ ] [ ] [ ] [ ] [ ] [ ] [ ] [ ] [ ] [ ]
5  [ ] [ ] [ ] [ ] [ ] [ ] [ ] [ ] [ ] [ ]
6  [ ] [ ] [ ] [ ] [ ] [ ] [ ] [ ] [ ] [ ]
7  [ ] [ ] [ ] [ ] [ ] [ ] [ ] [ ] [ ] [ ]
8  [ ] [ ] [ ] [ ] [ ] [ ] [ ] [ ] [ ] [ ]
9  [ ] [ ] [ ] [ ] [ ] [ ] [ ] [ ] [ ] [ ]
10 [ ] [ ] [ ] [ ] [ ] [ ] [ ] [ ] [ ] [ ]

********************************************************************************
[Score : -1] [Shot(s) : 1]
[Hit(s) : 0] [Miss(es) : 1] [hit-to-miss ratio : 0.00]

Ships sunked : []
Ships unsunked : [Aircraft carrier, Battleship, Submarine, Destroyer, Patrol Boat]
********************************************************************************
Enter x and y coordinates to shoot. Enter x = 0 or y = 0 to exit shooting.
Enter x :
```

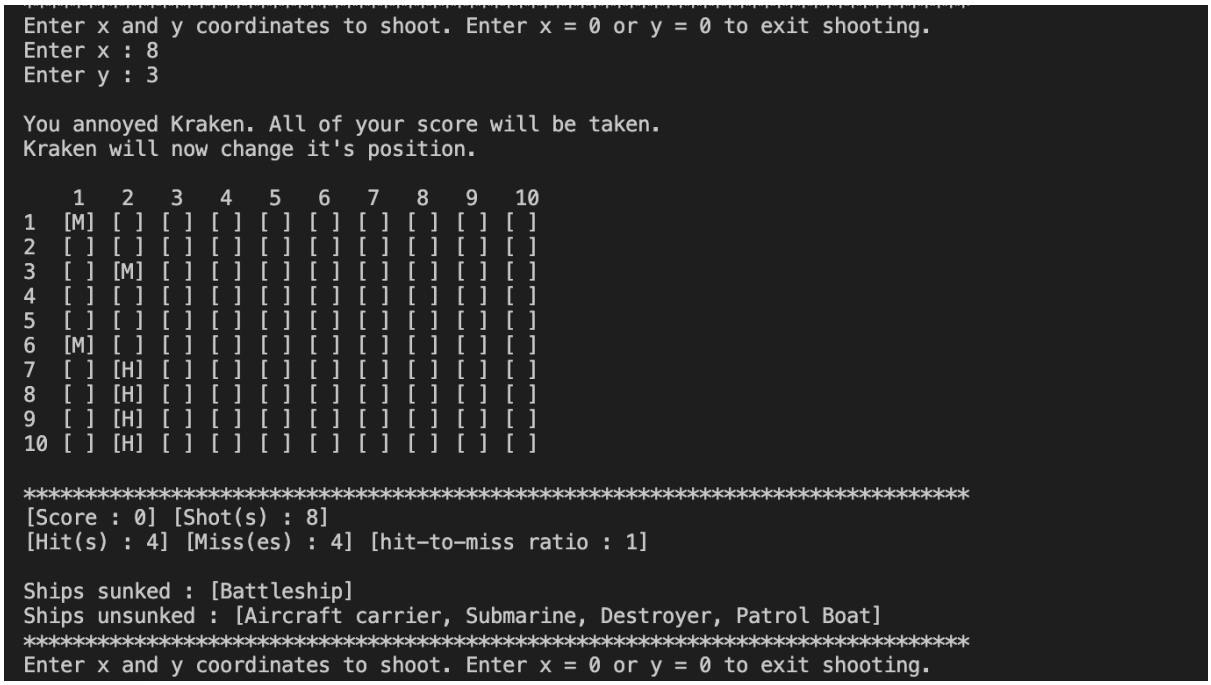
```
********************************************************************************
Enter x and y coordinates to shoot. Enter x = 0 or y = 0 to exit shooting.
Enter x : 2
Enter y : 8

It is a hit!

    1   2   3   4   5   6   7   8   9   10
1  [M] [ ] [ ] [ ] [ ] [ ] [ ] [ ] [ ] [ ]
2  [ ] [ ] [ ] [ ] [ ] [ ] [ ] [ ] [ ] [ ]
3  [ ] [M] [ ] [ ] [ ] [ ] [ ] [ ] [ ] [ ]
4  [ ] [ ] [ ] [ ] [ ] [ ] [ ] [ ] [ ] [ ]
5  [ ] [ ] [ ] [ ] [ ] [ ] [ ] [ ] [ ] [ ]
6  [M] [ ] [ ] [ ] [ ] [ ] [ ] [ ] [ ] [ ]
7  [ ] [ ] [ ] [ ] [ ] [ ] [ ] [ ] [ ] [ ]
8  [ ] [H] [ ] [ ] [ ] [ ] [ ] [ ] [ ] [ ]
9  [ ] [H] [ ] [ ] [ ] [ ] [ ] [ ] [ ] [ ]
10 [ ] [H] [ ] [ ] [ ] [ ] [ ] [ ] [ ] [ ]

********************************************************************************
[Score : -3] [Shot(s) : 6]
[Hit(s) : 3] [Miss(es) : 3] [hit-to-miss ratio : 1.00]

Ships sunked : []
Ships unsunked : [Aircraft carrier, Battleship, Submarine, Destroyer, Patrol Boat]
********************************************************************************
Enter x and y coordinates to shoot. Enter x = 0 or y = 0 to exit shooting.
Enter x :
```

The user can shoot the any position on the board by entering the x and y coordinates. It will be either a hit or miss or monster-hit. After every shoot the user is updated with board which will show “M” or “H” depending if the shoot was a miss or hit respectively.
And also, game stats such as score, shots, hits, misses, hit-to-miss ratio, ships sunk and ships unsunk are updated and shown after every shoot.

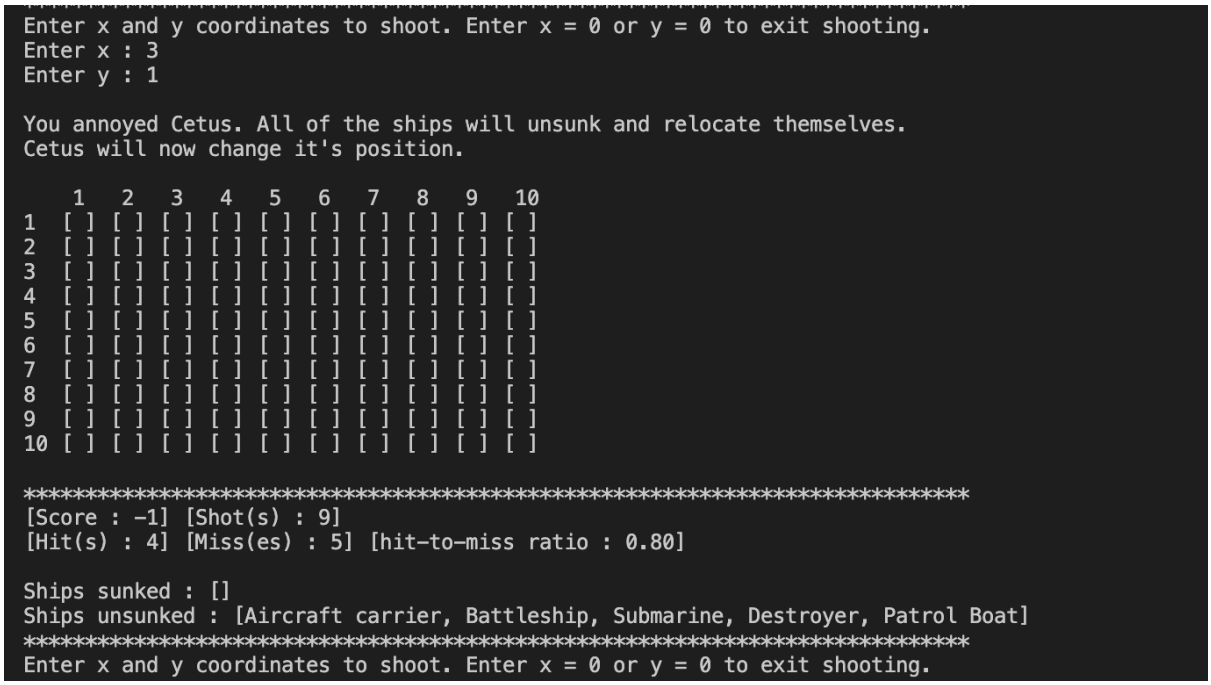


When user destroys a ship, it goes to sunk ships section from unsunked ships.

Monsters

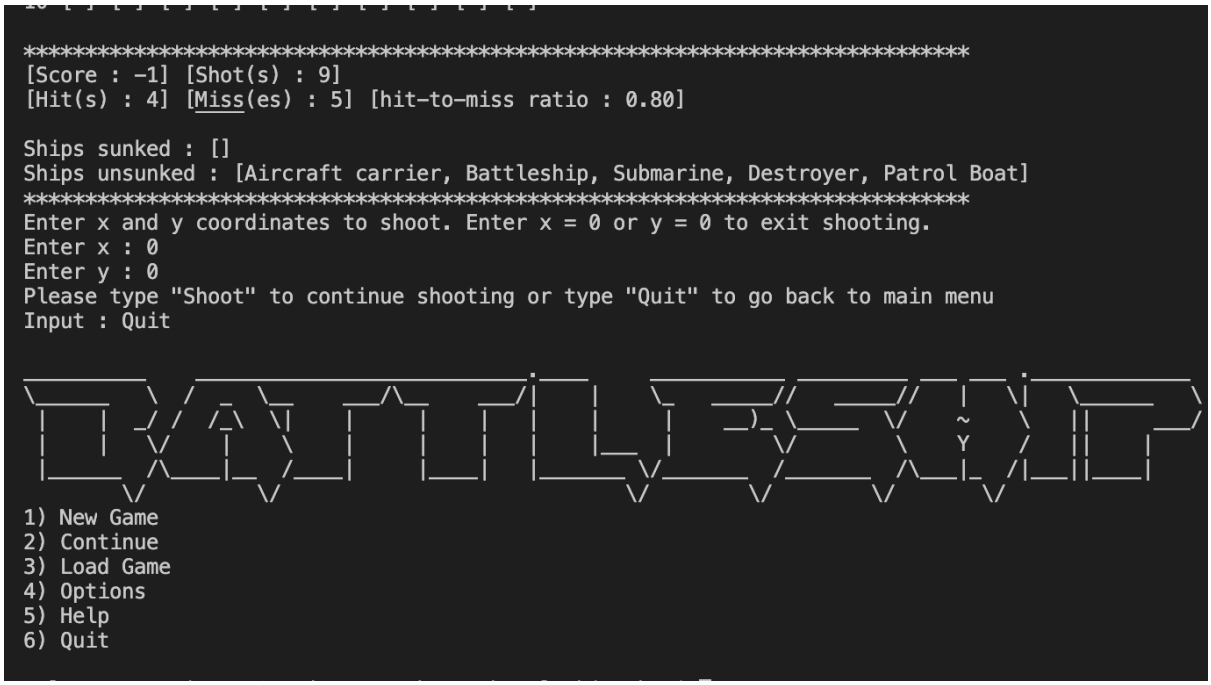


When the Kraken monster is hit, it will take all the points of the user and will move to new position randomly.

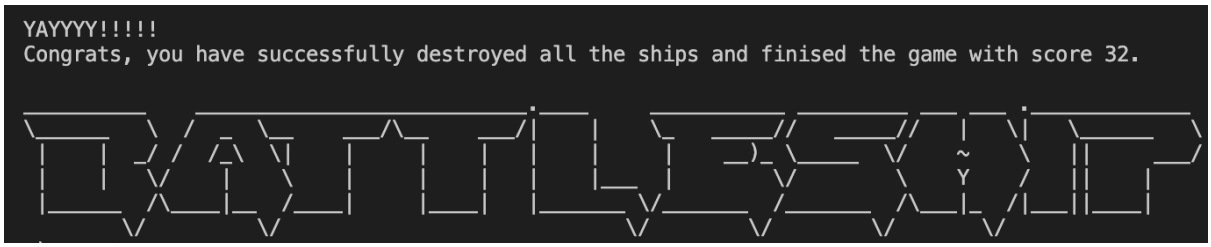


When the Cetus monster is hit, it will unsunk all the ships and relocate them randomly and then the monster will move to a new position randomly.

Exiting



User will have to enter 0,0 to exit shooting mode and then have to type “Quit” to go back to the main menu. After that user can simply exit the game by entering 6 or can play again by .



When the user has destroyed all the ships, he will be showed a message “Congrats, you have successfully finished the game with score [score]” and will go back to the menu where the user can either play again or exit the game.