

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**FACULTY OF SCIENCE AND TECHNOLOGY**

**COURSEWORK FOR THE**

**BSC (HONS) INFORMATION SYSTEMS; YEAR 1**

**BSC (HONS) INFORMATION TECHNOLOGY; YEAR 1**

**BSC (HONS) COMPUTER SCIENCE; YEAR 1**

**BSC (HONS) INFORMATION TECHNOLOGY (COMPUTER NETWORKING AND**

**SECURITY); YEAR 1**

**BSC (HONS) SOFTWARE ENGINEERING; YEAR 1**

**BIS(HONS) IN MOBILE COMPUTING WITH ENTERPRENEURSHIP; YEAR 1**

**ACADEMIC SESSION 2019; SEMESTER 2,3,4**

**PRG1203: OBJECT ORIENTED PROGRAMMING FUNDAMENTALS**

**DEADLINE: 4 JULY 2019 12PM**

**INSTRUCTIONS TO CANDIDATES**

* This assignment will contribute 20% to your final grade.
* This is a group (maximum 5 students) assignment

|  |
| --- |
| **IMPORTANT**    The University requires students to adhere to submission deadlines for any form of assessment. Penalties are applied in relation to unauthorized late submission of work. |
| * Coursework submitted after the deadline but within 1 week will be accepted for a maximum mark of 40%. * Work handed in following the extension of 1 week after the original deadline will be regarded as a non-submission and marked zero. |

**Academic Honesty Acknowledgement**

“I Nimue Wafiya , Hiba Azhari, Fatima Hesham, Chong Jia Sean Maverick (student name). verify that this paper contains entirely my own work. I have not consulted with any outside person or materials other than what was specified (an interviewee, for example) in the assignment or the syllabus requirements. Further, I have not copied or inadvertently copied ideas, sentences, or paragraphs from another student. I realize the penalties *(refer student handbook undergraduate programme)* for any kind of copying or collaboration on any assignment.”

, , , (Student’s signature / Date)

# Table of Contents

* 1. Team Members ………………………………………………...… 2

1. Marking Scheme …………………………………………………. 2
2. UML Diagram…………………………………………………..... 3
3. Dependencies…………………………………………………….. 4
4. Game Screenshots ………………………………………………...5
5. Source Code…………………………………………………….....7

6.1 Game.java……………………………………………………….7

6.2 Board.java……………………………………………………….8

6.3 Difficulty.java………………………………………………… 16

6.4 Player.java……………………………………………………...17

6.5 Highscoreboard.java………………………………………...…19

6.6 Ship.java………………………………………………………..20

6.7 Item.java …………………………………………………….…22

6.8 Potion.java………………………………………………….…..23

6.9 Trap.java…………………………………………………….….24

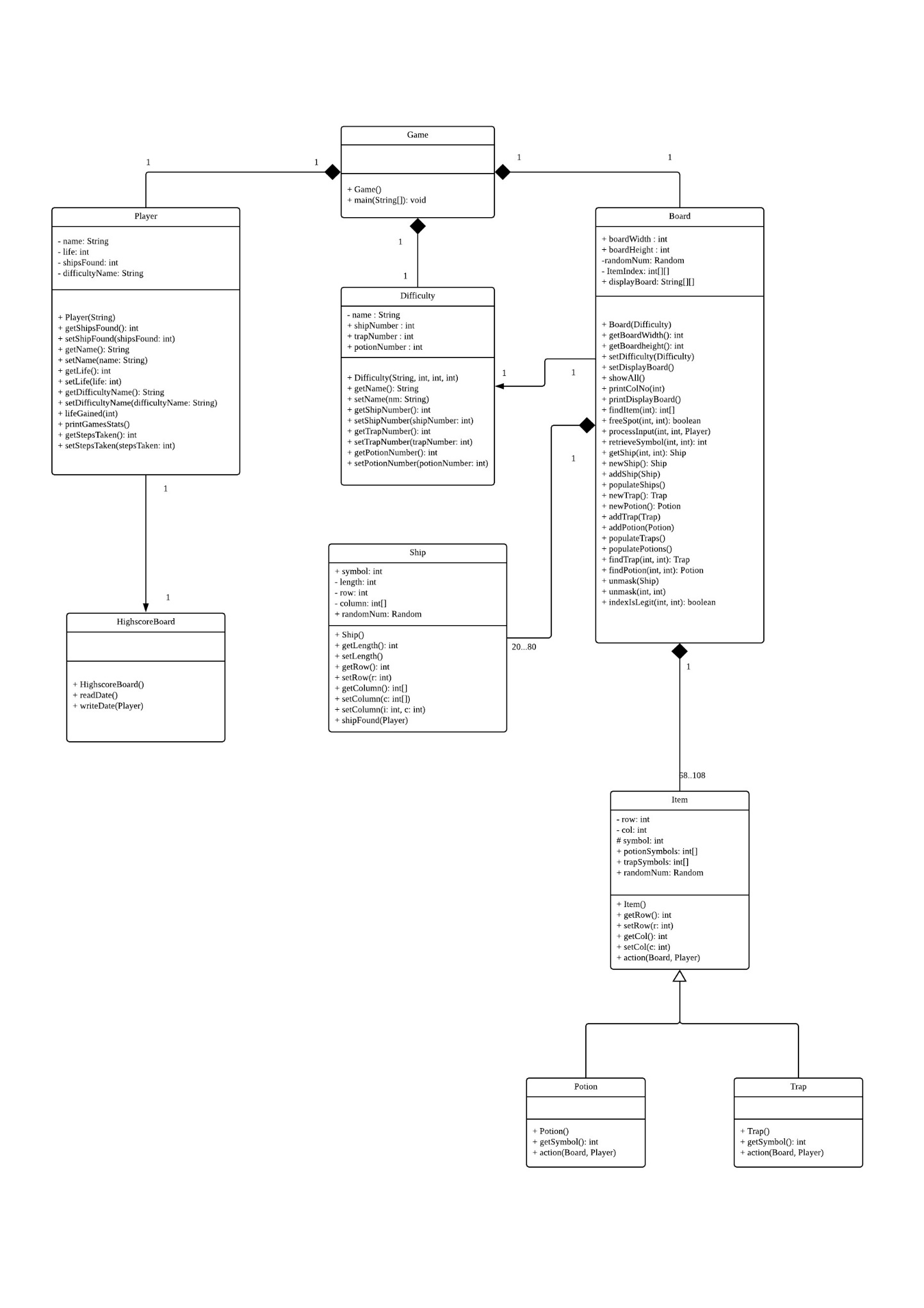
# 1.0 Team Members

|  |  |  |  |
| --- | --- | --- | --- |
| **No** | **Name** | **Student ID** | **Who upload to eLearn (tick)** |
| **1** | Nimue Wafiya | 17009952 |  |
| **2** | Hiba Azhari | 16065005 |  |
| **3** | Fatima Hesham | 17078619 |  |
| **4** | Chong Jia Sean Maverick | 18093369 | **✔** |

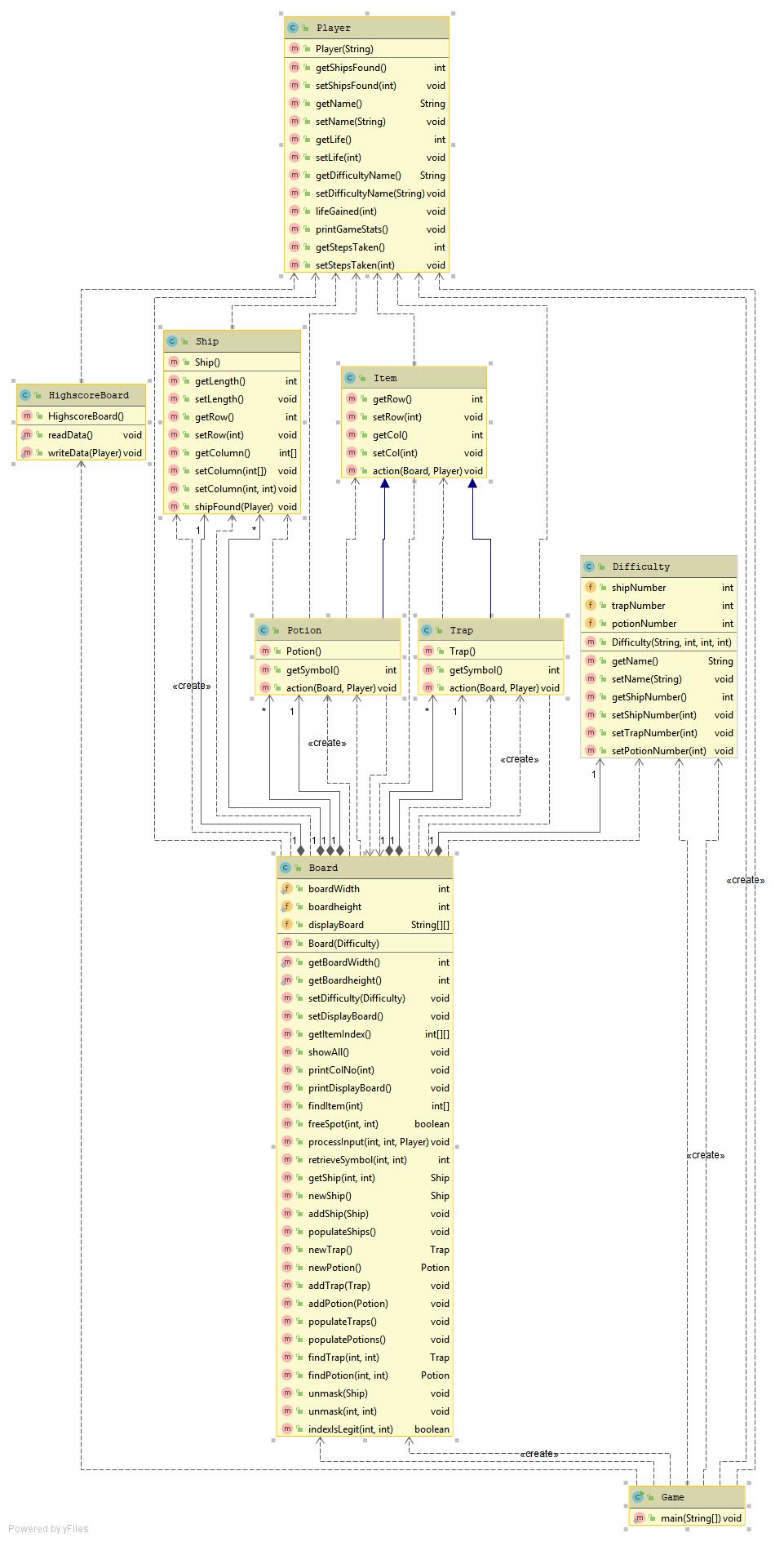
# 2.0 Marking Scheme

|  |  |  |  |
| --- | --- | --- | --- |
| **Criteria** | **Reference Marks** | **Marks** | **Remarks** |
| Design  Use of good object-oriented design to create appropriate classes and their relationship. Good UML class and class relationship diagrams. | **10** |  |  |
| Coding  Fulfil all the functionalities and align to the design. User-friendliness of the system, with appropriate Exception Handling, follow the programming best practices.    Good Javadoc documentation and screenshots to show the functionalities and test results. | **10** |  |  |
| **TOTAL** | **20** |  |  |

3.0 UML Diagram



4.0 Dependencies



5.0 Screenshots of game in use

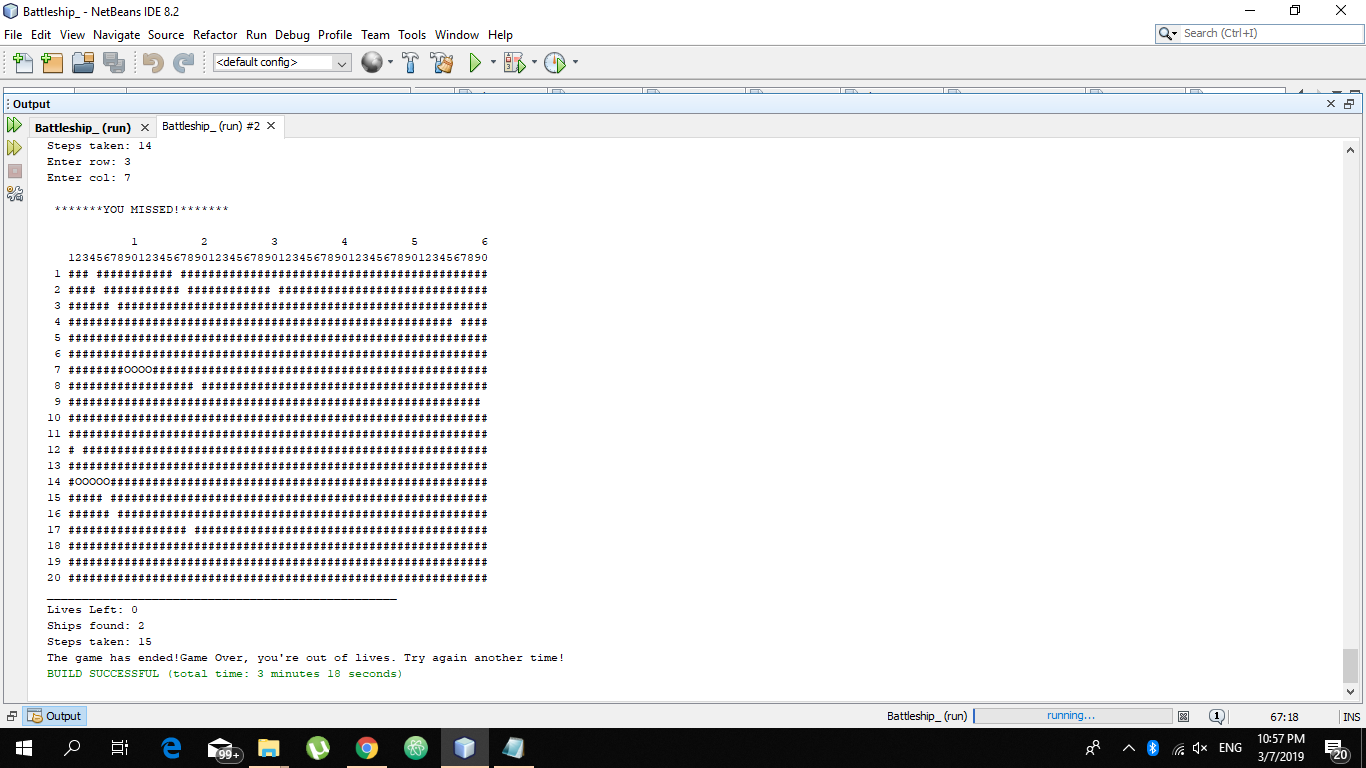


Figure 1: Player lost, comments found at the end based on the if-else statement after the while loop expression (in this case, user life = 0, it exited the while loop based on that value)

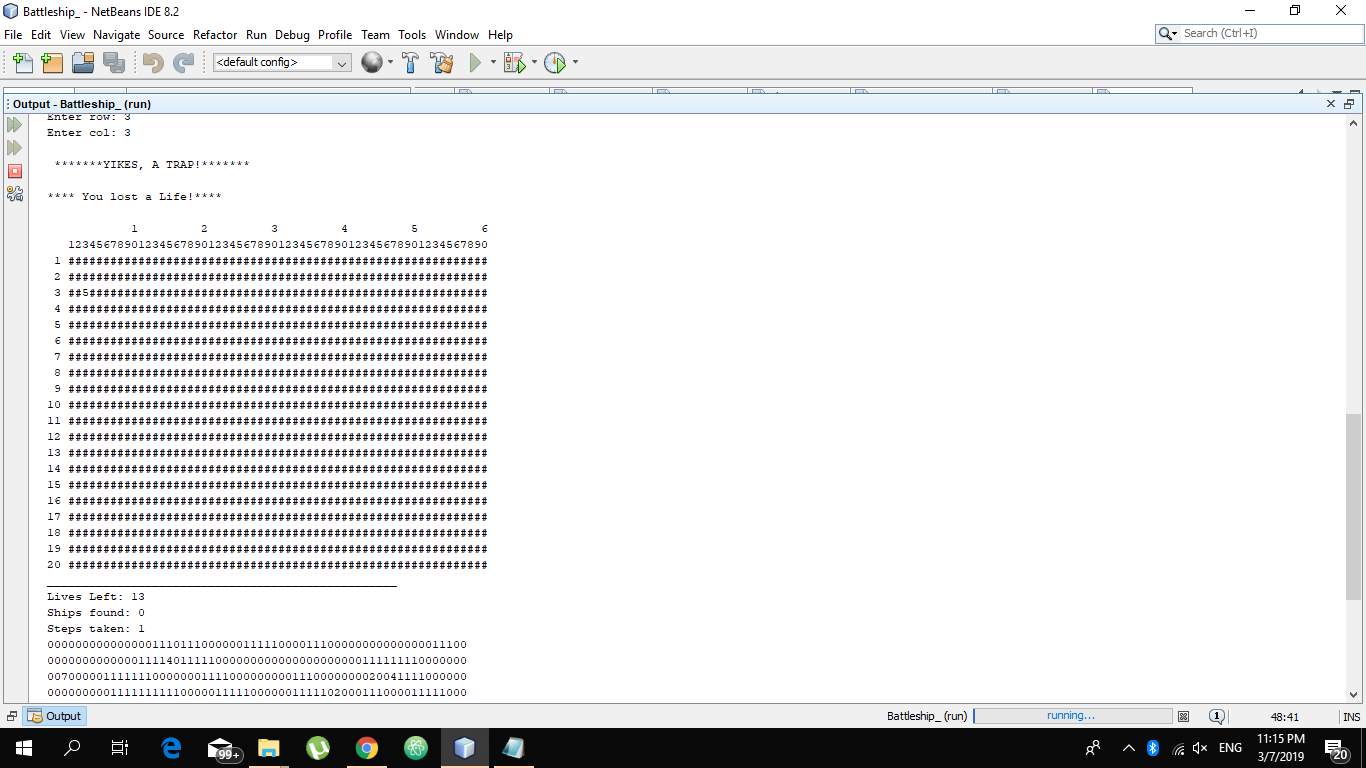


Figure 2: Shows that after the 1st step is taken, 2 lives are taken because the player hit a low danger trap. The same logic applies for the high danger trap. Below the display board is the index board that we used to intentionally hit certain items for debugging.

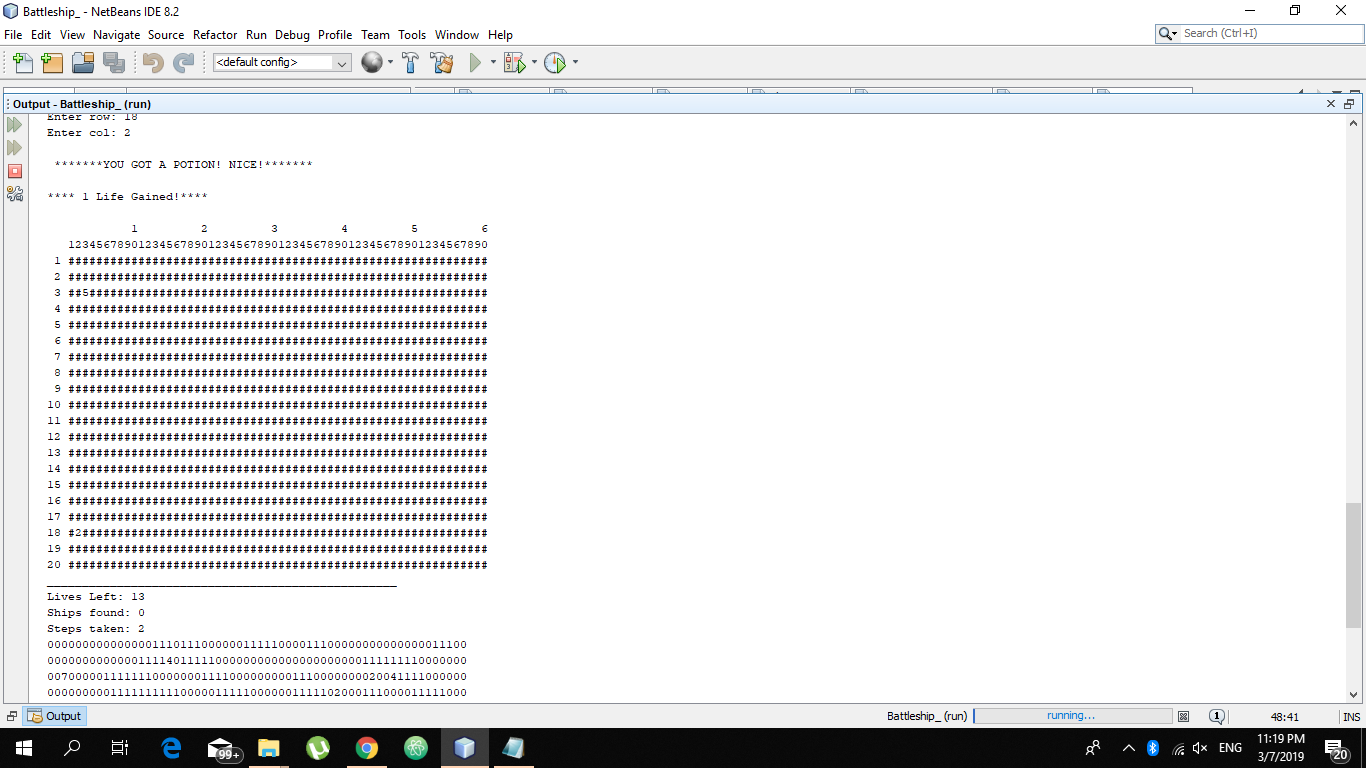


Figure 3: User hits the life potion, it is the 2nd step taken after figure 2, the user is given the life it was supposed to lose for making one move (unmasking one coordinate pair) but it remains the same

1. Source code

6.1 Game.java

**package** battleship\_;

**import** java.util.\*;

/\*\*

\* This class contains the Driver class that will run the main program.

\*/

**public** **class** Game {

**public** **static** **void** main (String[] args) {

HighscoreBoard.*readData*();

Scanner input = **new** Scanner(System.***in***);

Board board;

**int** userRow;

**int** userCol;

Difficulty Beginner = **new** Difficulty("Beginner",80,10,18);

Difficulty Intermediate = **new** Difficulty("Intermediate",50,20,18);

Difficulty Advanced = **new** Difficulty("Advanced",20,30,18);

System.***out***.print("Welcome to the game of battleship!\n"+ " Try to find 5 ships before losing all your lives!\n"+ " Look out for traps & potions!"+ " Enter your name to start: ");

String Pname = input.nextLine();

Player player = **new** Player(Pname);

System.***out***.print("Difficulty Levels:\n" + "1 = Beginner \n" + "2 = Intermediate \n" + "3 = Advanced \n" + "Input 1,2 or 3 to pick your difficulty:");

**int** diff = input.nextInt();

**switch**(diff) {

**case** 1:

board = **new** Board(Beginner);

**break**;

**case** 2:

board = **new** Board(Intermediate);

**break**;

**case** 3:

board = **new** Board(Advanced);

**break**;

**default**:

board = **new** Board(Beginner);

}

board.printDisplayBoard();

player.printGameStats();

**while** ((player.getLife() > 0) && (player.getShipsFound() < 5)){

**try** {

System.***out***.print("Enter row: ");

userRow = input.nextInt();

System.***out***.print("Enter col: ");

userCol = input.nextInt();

**if** (board.indexIsLegit(userRow,userCol)) {

board.processInput(userRow-1,userCol-1,player); //-1 to match indexes of array

board.printDisplayBoard();

player.printGameStats();

}

**else** {

System.***out***.println("\n\*\*\*NOT VALID COORDINATES\*.\n");

board.printDisplayBoard();

}

}**catch** (InputMismatchException e) {

System.***out***.println("That's not even a number");

input.next();

}

}

HighscoreBoard.*writeData*(player);

System.***out***.print("The game has ended!");

**if** (player.getLife() < 1 ){

System.***out***.println("Game Over, you're out of lives. Try again another time!");

}

**else**{

System.***out***.println("Congrats, you found 5 ships in the last minute!");

}

System.***out***.println("You Won");

}

}

6.2 Board.java

**package** battleship\_;

**import** java.util.\*;

**public** **class** Board {

**public** **static** **int** *boardWidth* = 60;

**public** **static** **int** *boardheight* = 20;

**private** Difficulty difficulty;

Random randomNum = **new** Random();

**private** **int** [][] ItemIndex = **new** **int**[20][60]; //generate 2D list containing item indexes (refer to javadoc)

**public** String[][] displayBoard = **new** String[20][60]; //

//CONTENTS

ArrayList<Ship>Ships = **new** ArrayList<Ship>();

ArrayList<Trap>Traps = **new** ArrayList<Trap>();

ArrayList<Potion>Potions = **new** ArrayList<Potion>();

//METHOD: generateItems(), so when you create an object in the driver class, you call Board.generateItems first

**public** Board(Difficulty diff) {

setDifficulty(diff);

populateShips();

populateTraps();

populatePotions();

setDisplayBoard();

}

**public** **static** **int** getBoardWidth() {

**return** *boardWidth*;

}

**public** **static** **int** getBoardheight() {

**return** *boardheight*;

}

**public** **void** setDifficulty(Difficulty difficulty) {

**this**.difficulty = difficulty;

}

/\*\*

\* Initializes the display board with '#'in all array indexes at the beginning of the game.

\*/

**public** **void** setDisplayBoard() { //initializes displayboard list to ####

**for** (**int** i = 0; i < *getBoardheight*(); i++) {

**for** (**int** j = 0; j < *getBoardWidth*(); j++) {

displayBoard[i][j] = "#";

}

}

}

**public** **int**[][] getItemIndex(){

**return** **this**.ItemIndex;

}

/\*\*

\* This is for programmers to print out the board with all indexes revealed.

\*/

**public** **void** showAll() { //prints item index, where every item is

**for** (**int** i = 0; i < *getBoardheight*(); i++) {

**for** (**int** j = 0; j < *getBoardWidth*(); j ++) {

**if** (j == *getBoardWidth*()-1) {

System.***out***.println(ItemIndex[i][j]);

}

**else** {

System.***out***.print(ItemIndex[i][j]);

}

}

}

}

/\*\*

\* This is is used in the method printDisplayBoard().

\* **@param** i A counter for the display board.

\*/

**public** **void** printColNo(**int** i){

**if**(i<9){

System.***out***.printf(" %d ",i + 1);

}

**else**{

System.***out***.printf("%d ",i + 1);

}

}

/\*\*

\* Prints the outer display of the gameboard that is played.

\*/

**public** **void** printDisplayBoard() { //simply prints display board

System.***out***.print(" 1 2 3 4 5 6\n");

System.***out***.print(" 123456789012345678901234567890123456789012345678901234567890\n");

**for** (**int** i = 0; i < *boardheight*; i++) {

printColNo(i);

**for** (**int** j = 0; j < *boardWidth*; j ++) {

**if** (j == *boardWidth*-1) {

System.***out***.println(displayBoard[i][j]);

}

**else** {

System.***out***.print(displayBoard[i][j]);

}

}

}

//System.out.printf("You have %d Lives left\n",player.lives);

}

/\*\*

\* Method locates the items in the board based on its symbol.

\* **@param** symbol determines item to be located by method

\* **@return** returns a 2 dimensional array containing the coordinates of the item found.

\*/

**public** **int**[] findItem(**int** symbol) { //finds first occurence of given item type

**for**(**int** i = 0; i < Board.*boardheight*;i++) {

**for**(**int** j = 0; j < Board.*boardWidth*;j++) { //the ship will always be revealed in the first few rows

**if**(**this**.getItemIndex()[i][j] == symbol) { // if we find an x,y pair that is a ship (1)

**int**[] coordinates = {i,j};

**return** coordinates;

}

}

}

**int**[] coordinates = {0,0};

**return** coordinates;

}

/\*\*

\* This method takes the row and column and checks if that position is occupied by an item or ship.

\* **@param** r contains the row values.

\* **@param** c contains the column values.

\* **@return** returns true or false.

\*/

**public** **boolean** freeSpot(**int** r, **int** c){

**if**(**this**.getItemIndex()[r][c] == 0) { // if we find an x,y pair that is a ship (1)

**return** **true**;

}

**return** **false**;

}

/\*\*

\*This method unmasks the board depending on the input of the player based on the three cases.

\* **@param** r contains the row value.

\* **@param** c contains the column value.

\* **@param** player Player who inputted the row and column,

\*/

**public** **void** processInput(**int** r, **int** c, Player player){ //takes user input and calls necessary methods to process it. Uses switchcase

**int** symbolAtPosition = retrieveSymbol(r,c);

**if** (symbolAtPosition != 7) { //if not yet found, as 7 represents found items

player.lifeGained(-1);

player.setStepsTaken(1);

}

**switch**(symbolAtPosition){

**case** 0:

//nothing at position:

displayBoard[r][c] = " ";

ItemIndex[r][c] = 7;

System.***out***.println("\n \*\*\*\*\*\*\*YOU MISSED!\*\*\*\*\*\*\*\n");

**break**;

**case** 1:

//Ship Found:

//Find ship object in ships array

Ship ship = getShip(r,c);

//Reveal it all - loop through ship coordinates and change corresponding displayboard positions to "O"

unmask(ship);

//Update player accordingly

ship.shipFound(player);

System.***out***.println("\n \*\*\*\*\*\*\*YOU FOUND A SHIP! GOOD JOB!\*\*\*\*\*\*\*\n");

**break**;

**case** 7:

System.***out***.println("\n \*\*\*\*\*\*\*YOU'VE ALREADY MADE THAT MOVE!\*\*\*\*\*\*\*\n");

**default**:

**if**(symbolAtPosition < 5){

findPotion(r,c).action(**this**,player);

}

**else** **if** (symbolAtPosition < 7){

findTrap(r,c).action(**this**,player);

}

unmask(r,c);

}

}

/\*\*

\* This methods retrieves the symbol occupied by position of the row and column.

\* **@param** r contains the row value.

\* **@param** c contains the column value.

\*/

**public** **int** retrieveSymbol(**int** r, **int** c){ //takes user input and translates it to a corresponding board symbol in ItemIndex

**return** ItemIndex[r][c];

}

/\*\*

\* Finds the ship in the array that occupies the row and column inputted.

\* **@param** r contains the row value.

\* **@param** c contains the column value.

\* **@return** returns the ship found in the array.

\*/

**public** Ship getShip(**int** r, **int** c){

**for** (Ship s : Ships) {

**if**(r == s.getRow()){

**for** (**int** i = 0 ; i < s.getLength(); i++) {

**if** ( c == ((s.getColumn())[i])) {

**return** s;

}

}

}

}

**return** **null**;

}

/\*\*

\* This method produces a ship object that does overlap any other ship object on the board.

\* **@return** Returns a valid ship.

\*/

**public** Ship newShip(){

Ship ship = **new** Ship();

//create a ship that does not overlap anything

**while**(**true**){

**int** r = randomNum.nextInt(Board.*boardheight*);

**int** c = randomNum.nextInt(Board.*boardWidth* - ship.getLength());

**int** i = 0;

**while**(i < ship.getLength()){

**if**(freeSpot(r,c + i)){

ship.setColumn(i,c + i);;

i++;

}

**else**{

**break**;

}

}

**if**(i == ship.getLength()){

ship.setRow(r);

**return** ship;

}

}

}

/\*\*

\* This method adds ship to the board.

\* **@param** ship Contains the ship object.

\*/

**public** **void** addShip(Ship ship){

**for** (**int** i = 0; i < ship.getLength(); i++) {

ItemIndex[ship.getRow()][ship.getColumn()[i]] = 1;

}

Ships.add(ship);

}

/\*\*

\* Adds all ship to the board based on the difficulty.

\*/

**public** **void** populateShips(){

**for** (**int** i = 0; i < difficulty.getShipNumber(); i++) {

addShip(newShip());

}

}

/\*\*

\* This method produces a trap object that does overlap any other trap object on the board.

\*/

**public** Trap newTrap(){

Trap trap = **new** Trap();

**while**(1==1){ //random condition temporarily

**int** r = randomNum.nextInt(Board.*boardheight*);

**int** c = randomNum.nextInt(Board.*boardWidth*);

**if**(freeSpot(r,c)){

trap.setRow(r);

trap.setCol(c);

**return** trap;

}

}

}

/\*\*

\* This method produces a potion object that does overlap any other potion object on the board.

\*/

**public** Potion newPotion(){

Potion potion = **new** Potion();

**while**(**true**){ //random condition temporarily

**int** r = randomNum.nextInt(Board.*boardheight*);

**int** c = randomNum.nextInt(Board.*boardWidth*);

**if**(freeSpot(r,c)){

potion.setRow(r);

potion.setCol(c);

**return** potion;

}

}

}

/\*\*

\* This method adds trap to the board.

\* **@param** trap Contains the trap object.

\*/

**public** **void** addTrap(Trap trap){

ItemIndex[trap.getRow()][trap.getCol()] = trap.getSymbol();

Traps.add(trap);

}

/\*\*

\* This method adds potion to the board.

\* **@param** potion Contains the potion object.

\*/

**public** **void** addPotion(Potion potion){

ItemIndex[potion.getRow()][potion.getCol()] = potion.getSymbol();

Potions.add(potion);

}

/\*\*

\* Adds all traps to the board based on the difficulty.

\*/

**public** **void** populateTraps(){

**for**(**int** i = 0;i<difficulty.trapNumber;i++){

addTrap(newTrap());

}

}

/\*\*

\* Adds all potions to the board based on the difficulty.

\*/

**public** **void** populatePotions(){

**for**(**int** i = 0;i<difficulty.potionNumber;i++){

addPotion(newPotion());

}

}

/////////////////////////////////

/\*\*

\* Finds trap in the array of traps

\* **@param** r contains the row value.

\* **@param** c contains the column value.

\*/

**public** Trap findTrap(**int** r,**int** c){ //in arraylist

**for**(Trap t : Traps){

**if**(r == t.getRow() && c == t.getCol()){

**return** t;

}

}

**return** **null**;

}

/\*\*

\* Finds potion in the array of potion.

\* **@param** r contains the row value.

\* **@param** c contains the column value.

\*/

**public** Potion findPotion(**int** r,**int** c){ //in arraylist

**for**(Potion p : Potions){

**if**(r == p.getRow() && c == p.getCol()){

**return** p;

}

}

**return** **null**;

}

/\*\*

\* Unmasks a ship in the board.

\* **@param** ship contains the ship to be unmasked.

\*/

**public** **void** unmask(Ship ship){ //takes ship, loops through its coord, sets corresp ItemIndex to 2 and displayBoard to "O"

**int** r = ship.getRow();

**int**[] c = ship.getColumn();

**for** (**int** i = 0; i < ship.getLength(); i++) { //successful

displayBoard[r][c[i]] = "O"; //unmask accordingly

ItemIndex[r][c[i]] = 7;

}

}

/\*\*

\* Unmasks a item in the board.

\* **@param** r contains the row value.

\* **@param** c contains the column value

\*/

**public** **void** unmask(**int** r, **int** c){ //takes item, gets coord, sets corresp ItemIndex

displayBoard[r][c] = Integer.*toString*(ItemIndex[r][c]);

ItemIndex[r][c] = 7;

}

/\*\*

\* This method checks if the users input is within the the range of 0-20 for rows and 0-60 for columns.

\* **@param** r contains the row value.

\* **@param** c contains the column value

\*/

**public** **boolean** indexIsLegit(**int** r, **int** c) {

**if** ((r > 0 && r <= 20) && (c > 0 && c <= 60)){

**return** **true**;

}

**else** {

**return** **false**;

}

}

}

6.3 Difficulty.java

**package** battleship\_;

**public** **class** Difficulty {

**private** String name;

**public** **int** shipNumber;

**public** **int** trapNumber;

**public** **int** potionNumber;

/\*\*

\* Creates a pre-set difficulty that we define.

\* **@param** nm contains the name of the difficulty.

\* **@param** shipNum contains the number of ships associated with the difficulty level.

\* **@param** trapNum contains the number of traps associated with the difficulty level.

\* **@param** potionNum contains the number of potion associated with the difficulty level.

\*/

**public** Difficulty (String nm, **int** shipNum, **int** trapNum, **int** potionNum){

setShipNumber(shipNum);

setTrapNumber(trapNum);

setPotionNumber(potionNum);

}

**public** String getName() {

**return** name;

}

**public** **void** setName(String nm) {

**this**.name = nm;

}

**public** **int** getShipNumber() {

**return** shipNumber;

}

**public** **void** setShipNumber(**int** shipNumber) {

**this**.shipNumber = shipNumber;

}

**public** **void** setTrapNumber(**int** trapNumber) {

**this**.trapNumber = trapNumber;

}

**public** **void** setPotionNumber(**int** potionNumber) {

**this**.potionNumber = potionNumber;

}

}

6.4 Player.java

**package** battleship\_;

**public** **class** Player {

String name;

**private** **int** life;

**private** **int** shipsFound;

**private** **int** stepsTaken;

**private** String difficultyName; //to keep inside the score board

**public** Player(String nm) {

setName(nm);

setLife(15);

setStepsTaken(0);

shipsFound = 0;

}

**public** **int** getShipsFound() {

**return** shipsFound;

}

**public** **void** setShipsFound(**int** shipsFound) {

**this**.shipsFound = shipsFound;

}

**public** String getName() {

**return** name;

}

**public** **void** setName(String name) {

**this**.name = name;

}

**public** **int** getLife() {

**return** life;

}

**public** **void** setLife(**int** life) {

**this**.life = life;

}

**public** String getDifficultyName() {

**return** difficultyName;

}

**public** **void** setDifficultyName(String difficultyName) {

**this**.difficultyName = difficultyName;

}

**public** **void** lifeGained(**int** lives) {

**this**.life = **this**.life + lives;

}

**public** **void** printGameStats(){

**for**(**int** i=0;i<50;i++){

System.***out***.print("\_");

}

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

System.***out***.printf("Lives Left: %d \n",**this**.life);

System.***out***.printf("Ships found: %d \n", **this**.shipsFound);

System.***out***.printf("Steps taken: %d \n", **this**.stepsTaken);

}

**public** **int** getStepsTaken() {

**return** stepsTaken;

}

**public** **void** setStepsTaken(**int** stepsTaken) {

**this**.stepsTaken = **this**.stepsTaken + stepsTaken;

}

}

6.5 Highscoreboard.java

**package** battleship\_;

**import** java.io.File;

**import** java.io.FileNotFoundException;

**import** java.lang.SecurityException;

**import** java.util.Formatter;

**import** java.util.FormatterClosedException;

**import** java.util.NoSuchElementException;

**import** java.util.Scanner;

/\*\*

\* This class contains the scoreboard.

\*/

**public** **class** HighscoreBoard{

**public** HighscoreBoard(){

}

//find the highest score from the text file (using MAX) and keep it inside topTen, remove that score from the text file

//find highest score again, with previous highest score removed, add to topTen, repeat until all has been read

//by the end, add all high scores back to the text file

**public** **static** **void** readData() {

String topTen[][] = **new** String[10][3];

Scanner input;

**try** {

input = **new** Scanner(**new** File("Scores.txt"));

**while**(input.hasNext()) {

System.***out***.printf("%-12s%-10d%-12s",input.hasNext(), input.hasNext(),input.hasNext());

}

**if** (input != **null**) {

input.close();

}}

**catch**(FileNotFoundException fe) {

System.***out***.println("Error opening file.");

}

**catch**(NoSuchElementException ex) {

System.***out***.println("File impropery formed.");

}

}

//write player results on a new line in the text file, checking for duplicate high scores and replacing them if they are duplicates

**public** **static** **void** writeData(Player p){

Formatter output;

**try** {

output = **new** Formatter("Score.txt");

output.format("%s%s%s\n",p.getName(), Integer.*toString*(p.getLife()), p.getDifficultyName());

**if** (output != **null**)

{

output.close();

}

}**catch** (SecurityException se){

System.***out***.println("You don't have write access.");

System.*exit*(1);

} **catch** (FileNotFoundException fe) {

System.***out***.println("Error opening/creating file.");

System.*exit*(1);

}

}

}

6.6 Ship.java

package battleship\_;

import java.util.Arrays;

import java.util.Random;

/\*\*

\* This class is contains the ship class.

\*/

public class Ship{

static int symbol = 1;

private int length;

private int row;

private int column[];

Random randomNum = new Random();

public Ship (){

setLength();

column = new int[this.length];

}

public int getLength() {

return this.length;

}

/\*\*

\* The method to randomize the ships length starting from 3-5.

\*/

public void setLength() {

this.length = 3 + randomNum.nextInt(3);

}

public int getRow() {

return this.row;

}

public void setRow(int r) {

this.row = r;

}

public int[] getColumn() {

return this.column;

}

public void setColumn(int[] c) {

this.column = c;

//column size it can appear on

}

public void setColumn(int i,int c){

this.column[i] = c;

}

/\*\*

\* The method to increase ships found by 1..

\*/

public void shipFound(Player player) {

player.setShipsFound(player.getShipsFound() + 1);

}

}

6.7 Item.java

package battleship\_;

import java.util.ArrayList;

import java.util.Random;

/\*\*

\* Is a superclass that will be inherited by potion,ship and trap.

\* @param row contains the row position of the item.

\* @param col contains the column position of the item.

\* @param symbol contains the symbol that denotes the item in the gameboard.

\* @param potionSymbols contains array of symbols that denotes the potions in the gameboard.

\* @param trapSymbols array of array of symbols that denotes the traps in the gameboard.

\*/

public class Item {

private int row;

private int col;

protected int symbol;

static int[] potionSymbols = {2, 3, 4};

static int[] trapSymbols = {5, 6};

Random randomNum = new Random();

public int getRow() {

return row;

}

public void setRow(int r) {

this.row = r;

}

public int getCol() {

return col;

}

public void setCol(int c) {

this.col = c;

}

public void action(Board board, Player player){

}

}

6.8 Potion.java

/\*

\* To change this license header, choose License Headers in Project Properties.

\* To change this template file, choose Tools | Templates

\* and open the template in the editor.

\*/

package battleship\_;

public class Potion extends Item{

public Potion(){

super();

this.symbol = potionSymbols[randomNum.nextInt(potionSymbols.length)];

}

public int getSymbol(){

return this.symbol;

}

/\*\*

\* This method executes the potion based on its symbol.

\* @param board Contains the board that is modified by the potion.

\* @param player Contains the player that is modifie.

\*/

public void action(Board board, Player player){

System.out.println("\n \*\*\*\*\*\*\*YOU GOT A POTION! NICE!\*\*\*\*\*\*\*");

switch(this.symbol){

case 2:

player.lifeGained(1);

System.out.println("\n\*\*\*\* 1 Life Gained!\*\*\*\*\n");

break;

case 3:

int[] Scoordinates = board.findItem(1);

Ship ship = board.getShip(Scoordinates[0],Scoordinates[1]);

board.unmask(ship);

ship.shipFound(player);

System.out.println("\n\*\*\*\* 1 Ship Revealed!\*\*\*\*\n");

break;

case 4:

int trapSymbol = trapSymbols[randomNum.nextInt(trapSymbols.length)];

int[] Tcoordinates = board.findItem(trapSymbol);

board.unmask(Tcoordinates[0],Tcoordinates[1]);

System.out.println("\n\*\*\*\* 1 Trap Revealed!\*\*\*\*\n");

break;

default:

}

}

}

6.9 Trap.java

/\*

\* To change this license header, choose License Headers in Project Properties.

\* To change this template file, choose Tools | Templates

\* and open the template in the editor.

\*/

**package** battleship\_;

/\*\*

\* This class has inherits the methods and Item class

\*/

**public** **class** Trap **extends** Item{

/\*\*

\* The switch case is to identify which type of trap it is for each type of method ; low or high danger

\* **@param** board References the board class.

\* **@param** player References the player class.

\*/

**public** Trap(){

**super**();

**this**.symbol = *trapSymbols*[randomNum.nextInt(*trapSymbols*.length)];

}

**public** **int** getSymbol(){

**return** **this**.symbol;

}

/\*\*

\* The switch case is to identify which type of trap it is for each type of method ; low or high danger

\* **@param** board References the board class.

\* **@param** player References the player class.

\*/

**public** **void** action(Board board, Player player){

System.***out***.println("\n \*\*\*\*\*\*\*YIKES, A TRAP!\*\*\*\*\*\*\*");

**switch**(**this**.symbol){

**case** 5:

player.lifeGained(-1);

System.***out***.println("\n\*\*\*\* You lost a Life!\*\*\*\*\n");

**break**;

**case** 6:

player.lifeGained(-2);

System.***out***.println("\n\*\*\*\* You lost 2 lives!\*\*\*\*\n");

**break**;

**default**:

}

}

}