CSC 110 Assignment 5:
Search and Rescue: A Battleship Game!

Motivation:

The computer gaming industry once created primitive computer based toys for geeks and kids. Today it has morphed into a major form of entertainment. Canada is home to a thriving video game industry. There are over 100 Canadian-based companies devoted to interactive game development, publishing or distribution, of these 30 are in British Columbia! Vancouver is internationally recognized as a hub of the game industry.

As the size and complexity of games has increased, industry is looking for a higher-level of education in their employees. They are turning to academia to help prepare a new generation of game developers. Graduates from such programs are in high demand!

The game we are creating is a variation on the popular game: Battleship. We prefer, however, to simply locate the forlorn souls that are aboard lost ferries, cruise ships, sailboats and dingys.

Learning Outcomes:

When you have completed this assignment, you should understand:

- How to identify instance variables, instance methods, and constructor methods in a class definition.
- How to summarize the action of instance methods for *external* users.
- How to instantiate and use an object given its class definition.
- How to work with two dimensional arrays and search within that space efficiently.

Pair Programming :

This assignment can be completed as part of a pair programming team. If you choose to do so:

- read the description of pair programming provided in the Resources section of the course web page
- find a suitable pair programming partner.
- (both members of the pair programming team must) send an email to the course email (csc110@csc.uvic.ca) informing the instructional staff of the pairing. (Even if you were a partnered with the same person in Assignment, you **must** send an email for Assignment 5!)
- together with your partner, create a schedule for assignment completion

Your Task:

For this assignment you will be writing the program that implements a simple version of a game that locates lost ships. This game is played with two players, each player knows the location of the ships in one area of the water and is searching for the ships in another. An area of water is a square grid called the board. A ship fills one or more grid locations on the board. The game proceeds in turns with a player asking the other if there is a ship in a particular grid location on the board. Once all locations that contain a single ship have been found, the searching player is informed of the find. Since this is a game, the challenge of the game is to be the first to find all the ships. (And there are no

bonuses for finding a lost BC ferry containing hundreds of souls before finding a dingy containing only one!)

In our version of the game, the size of the square board is variable and chosen by a user. A single square on the board is referred to as a grid location. There are six ships on each board: two dingys, each that uses only a single grid location, a sailboat that uses two grid locations, a private yacht that uses three grid locations, a BC ferry that uses four grid locations and a cruise ship that uses five grid locations. Each ship is perfectly oriented in either a north/south or in an east/west direction, except for those dingys which can spin around!

Part 1: Examine a Class and Instantiate and Use Objects: The Player Class

The following code defines a very simple class called **Player**. Observe that the class contains the data associated with one player, and the core functionality (methods) a player needs. More specifically, it contains three member *data items* (or *attributes*) and a collection of member *methods* (or *behaviours*) for changing the values of that data.

```
/* ADD COMMENTS TO THIS CLASS, DO NOT CHANGE ANY CODE!
*/

import java.util.*;

public class Player {

  private int gamesWon;
  private String name;
  private int searchSpace;

  public Player(String player, int maxSize) {
    gamesWon    = 0;
    name = player;
    searchSpace = maxSize;
  }

  public void wonAGame() {
    gamesWon++;
  }

  public int move(String prompt) {
    int moveIndex;
    Scanner input = new Scanner(System.in);
    do {
      System.out.print("Please enter a " + prompt);
      System.out.print(" between 0 and " + (searchSpace
- 1) + ": ");
      moveIndex = input.nextInt();
    } while (moveIndex >= searchSpace);
    return moveIndex;
  }

  public String getName() {
    return name;
  }
```

```
  public int getGamesWon() {
    return gamesWon;
  }
}
```

Use this code but ***do not*** alter it. You are required to enhance the documentation by including
- For each method add a precise description of its purpose, input and output.
- Explain what each member variable is used for.

Part II: Creating Objects of Type Player: A simple Ship Searching Game

The following Java program (Searcher.java) defines two *objects* of type Player and demonstrates some initial simple functionality associated with the search and rescue game. Please note that this is just enough code to demonstrate some of the behaviours of Player objects, it does not implement all the functionality of the game. (This as a start, but not an end---use Control-C at the command prompt to stop the infinite loop! :)

```
public class Searcher {

  public static final int BOARD_DIMENSION = 10;

  public static void initBoard(char[][] game) {
    for (int r=0; r<game.length; r++)
      for (int c=0; c<game[r].length; c++)
        game[r][c] = '+';
   }


  public static void printBoard(String playerName,
char[][] game) {
    for (int r=0; r<game.length; r++) {
      for (int c=0; c<game[r].length; c++)
        System.out.print( game[r][c] + "  ");
      System.out.println();
    }
    System.out.println("That's the board for " +
playerName +'\n' + '\n');
  }

  public static boolean takeATurn(Player p, char[][] b)
{
    System.out.println("Ok " + p.getName() + " here we
go!");
    int theRow = p.move("row");
    int theColumn = p.move("column");
    if (theRow < 0 || theColumn < 0) {
        System.out.println("Exiting Game Before
Completion . . . ");
        Return true;
    }
    b[theRow][theColumn] = p.getName().charAt(0);
    printBoard(p.getName(), b);

    return false; // fix this!
  }
  public static void main(String[] args) {
    char [][] player1_board = new
char[BOARD_DIMENSION][BOARD_DIMENSION];
    char [][] player2_board = new
char[BOARD_DIMENSION][BOARD_DIMENSION];

    Player p1 = new Player("Pat", BOARD_DIMENSION);
    Player p2 = new Player("Atilla", BOARD_DIMENSION);

    initBoard(player1_board);
    initBoard(player2_board);

    boolean done = false;
    do {
      //Revisit this to make it work! Use Control-C
to stop the loop...
```

```
      //Right now it is the game that NEVER ends!
      takeATurn(p1, player2_board);
      takeATurn(p2, player1_board);
    } while (!done);

   }
}
```

Add the following functionality to the Searcher program--in some cases this means modifying existing code in Searcher!:

1. To each player's initial configuration add a dingy (1 grid location long).
    - Note: the placement of the dingy (and all ships) should be done interactively, allowing the player to select the coordinates (row, column) of its position before the game begins.
    - Make sure only correct coordinates are accepted by your program.

- Remember that you will not change the code in the Player class, so do this within Searcher by adding a new method.

2. The location of the dingy should not be revealed to the other player. That is, when the board is displayed, it is not shown. This will require that you modify the existing method that prints the board.
3. Allow the game to be played until the dingy is found, and declare the winner, or until one of the players inputs a negative row or column coordinate (effectively a request to exit the game early).
    - Note that, after each time a player takes a turn, the game *could* be over. Right now the code in the main method of Searcher is NOT leveraging the fact that the "takeATurn" method returns a boolean. This boolean can be used to determine if the game is over, but you will need to add a new method to test to see we have a winner!

Part III: Creating A Full Featured Game

There will be a number of enhancements to make in order to complete the game--this list can serve as a guide. In order to get full marks you will need to provide all the required functionality by incrementally modifying the Searcher program above. That is, you must keep the 2D arrays and Players declared locally within main (you must pass parameters!), and use good judgment about what NEW methods you want to introduce into your code.

1. Allow players to introduce all the ships to the water, not just a dingy.
    - The placement should be done interactively, allowing the player to select the starting (x,y) position of each and its orientation, horizontal or vertical.
2. Add the ability to report if part or all of a ship has been found
    - In the case of a dingy, since it is only one grid location on the board, finding "part" means finding "all".
    - In all other cases, it must be determined if the complete ship has been found--do this efficiently! You should not have to search the entire board...
3. Determine when a player has won the game!
    - There are trade-offs in your approach here--for example, either you will have to track progress as the game is being played, OR maybe you will be doing a search on the board to see if there are any remaining elements to be found. Please document your decision carefully!
4. Increment the number of games the winning player has accumulated so far, print the scores for the two players, and allow for a rematch if the players would like to play again.

What to Hand In:

- Put your FIRST NAME, LAST NAME, STUDENT ID in each file of your assignment.
- Submit the following:
    - **Source code** with **documentation** from PARTS 1, 2 and 3 (as 3 separate files).

Although all 3 parts are due before your registered lab the week of November 29 – December 3, it is recommended that Parts 1 and 2 are submitted before your lab the week of November 22-26.

-
- **Grading:**

  Part 1: 1%

  Part 2: 1%

  Part 3: 2%

  More to Discover:

  Games are excellent applications to motivate the study of the programming and the computer science topics of programming languages or operating systems. There is a lot of web based documentation on planning and specifying a game programming initiative.

  The wonderful world of Object Oriented Programming (OOP) awaits you as well! If you check out Wikipedia, you will find out that OOP "is a programming paradigm that uses "objects" and their interactions to design applications and computer programs. It is based on several techniques, including inheritance, modularity, polymorphism, and encapsulation. It was not commonly used in mainstream software application development until the early 1990s". It's funny though, because if you read on, you'll find out it was REALLY around in the 60s!

  Intrigued? Want to know more? How about giving CSC115 a try? :)