# CS 2336 PROJECT 1 – Obstacles Warrior Game

**Unit Test Due:** 9/12 by 11:59 PM
**Project Due:** 9/20 by 11:59 PM

**KEY ITEMS:** Key items are marked in red. Failure to include or complete key items will incur additional deductions as noted beside the item.

**Submission:**

- **The file containing main must be named Main.java. (-5 points)**
- **The project files must be in a package named ObstaclesWarrior. (-5 points)**
- All project deliverables are to be submitted in eLearning.
- Zip the contents of the src directory into a single zipped file
  - Make sure the zipped file has a .zip extension (not .tar, .rar, .7z, etc.) (-5 points)
- Programs must compile and run with Java SE 22.
- Each student is responsible for developing unit test cases to ensure the program works as expected.
- **Type your name and netID in the comments at the top of all files submitted.** (-5 points)

**Objectives:** Create a Java program using programming fundamentals (file I/O, loops, conditional statements, arrays, functions)

**Problem:** In an effort to win a coding competition, you decided to create an awesome Obstacles Warrior game. The game is played on a 2-dimensional board similar to a Chess board, but the dimensions may be different. The minimum size of the board is 2x2. The board will have one square marked as Start with value equals to 1 and one square marked as Exit with value equals to 2. Start and Exit squares cannot be the same. Some of the board squares contains obstacles in the form of an integer that will define how the warrior position and score will be affected. The obstacle squares can have values from 0 to -10 only. The size of the board, obstacles, Start and Exit squares are all unknow to your code prior to running. This information is stored in a file that your code will read at the beginning of the game. The board.dat file must be read into a 2-D array.

A warrior must start at the Start square and find their way to the Exit square. The warrior can move on the board in any direction including diagonally, one square at a time. A warrior has a running score(integer) maintained from the start of the game until the warrior exits the board. If the warrior lands on an obstacle square with a value of zero, the warrior is sent back to the starting position and the obstacle square will become a normal square (obstacle removed). If the obstacle square has a negative number, that number will be deducted from the warrior's score and the obstacle square will become a normal square (obstacle removed). Each VALID move that the warrior makes without landing on an obstacle will earn the warrior one point. The moves for the warrior are randomly generated by your code in the form of a direction (0-UP, 1-DOWN, 2-LEFT, 3-RIGHT, 4-UPRIGHT, 5-DOWNRIGHT, 6-UPLEFT, 7-DOWNLEFT). If the warrior is at the boundary of the board and your code generates an invalid move, that move will be ignored. Your code will keep generating moves until the warrior exits at the correct square. Once the warrior exits, your program will store the updated board information to a new file resultboard.dat file. The program will also display the total number of valid moves, the total time elapsed, in milliseconds, since the first move until the warrior exited the board, the final score of the warrior and the formatted board information (right aligned columns).

**Output Format:**

```
Enter the board data file path: C:\board.dat   //Repeat prompt until valid
file OR show error and exit.
Type "Start" to start the game or "Exit" to exit the game: exit //Your
program must exit

Enter the board file path: C:\board.dat
Type "Start" to start the game or "Exit" to exit the game: start

//You may display the moves and the running score after each move but it is
not required

The warrior made 270 valid moves in 35 milliseconds. The final score is 175
points.

  #   -5   #    #    #
  #    #   #    0    #
  #    #   1    #    #
  #    #   #    #    #
-10    #   #    2    #

Press any key to exit!
```

**Program Structure:** Your code should be modular and easy to understand. In addition to the main method, the following methods are required to be in your code. These methods will be used by the unit testing to test the accuracy of your code.

```
public static String[][] ReadBoardFromFile(String fileName, Position
startPosition, Position exitPosition)
public static boolean WriteBoardToFile(String fileName,
String[][] boardArray)
public static int GenerateDirection()
public static boolean MoveWarrior(int direction, String[][] boardArray,
Position currentPosition)
public static int CalculateWarriorScore(int currentScore, Position currentPos
ition, String[][] boardArray)
public static String DisplayResults(int currentScore, int numberOfMoves,
int timeElapsed, String[][] boardArray)
```

**Program Flow:**

- Program starts in main() method
- Prompt user for Board.dat file path
- Read board from file
- Generate a direction and start the game
- Move the warrior, calculate score and check conditions
- Repeat until the warrior is at the exit square
- Display the results
- Prompt user to exit game

**Board.dat file format:**

- The data in the file will be separated by one space
- Assume that all data in the file is valid
- Clear, start and exit squares (no obstacles) will be marked with # in the file
- The first line of the file contains the dimensions of the board (rows and columns) e.g. 3 7
- The second line contains the Start square indexes (rowIndex, columnIndex)
- The third line will contain the Exit square indexes (rowIndex, columnIndex)
- The rest of the lines will represent the contents, including obstacles, of a row on the board
- Example of a Board size 5x5 data file:

```
5 5
2 2
4 3
# -5 # # #
# # # 0 #
# # # # #
# -3 # # -4
-10 # # # #
```

**Grading:**

- Coding standards, style and comments (10 Points)
- Unit testing methods x 6, one for each of the methods mentioned above (10 Points)
- ReadBoardFromFile (10 Points)
- WriteBoardToFile (10 Points)
- GenerateDirection (10 Points)
- MoveWarrior (20 Points)
- CalculateWarriorScore (20 Points)
- DisplayResults (10 Points)

**Notes:**

- The “Position” class is provided to you on eLearning
- Part 1 deliverables: Unit Test methods only
- Part 2 deliverables: Functioning program including the Unit Test methods