Big Program 1: Battleship

In this assignment, you will create a text-based Battleship (Wikipedia: Battleship (game)) program. It will be a one player game versus the computer. The computer will perform be the simplest possible heuristic, making all of its moves randomly. Sample run of the finished program (interleaved input and output): SampleFinal.txt

As shown in the sample run above, the display to the user during game play includes two boards. The board to the left is the primary board ("My Ships:") and it shows the player's ships and tracks the shots fired by the computer. The board on the right is the tracking board ("My Shots:") and it shows the shots (hits and miss) taken by the human player.

Grading Scheme: GradingBigPrograms.pdf

Row-major order: In the design of this program, we will view the 2d arrays as row-major order. This means that the first dimension represents the y-coordinate of the game board (the rows) and the second dimension represents the x-coordinate (the columns).

Files: Battleship.java, Config.java and TestBattleship.java are provided.

- Battleship.java -- Contains all the methods (as stubs) necessary for all three milestones of this program. Detailed method headers are provided for you in the file. The implementation of each method must meet the detailed descriptions in the method headers.
- TestBattleship.java -- Contains a framework for your test bench.
- Config.java -- Whenever you need these values, use the constants defined in Config.java. The testing may use different values for these constants.

BP1 Team

Students who wish to may work with at most one pair programming partner on Big Program 1.

1. Both students must read the Pair Programming Rules and Guidelines.
2. **Both students must individually** fill out the TEAM FORM requesting the other as their partner.
    A. *To use the above form, you need to be logged into the google account that is associated with your UW Netid. If you see the message "**You need permission**" when following the link above, please 1) direct your browser to google.com 2) click on the profile icon in the upper right corner of the screen and choose sign out, 3) then click "Sign In" and use the link "Sign in with a different account" link at the bottom of the following screen, 4) from there you can "Add an account" and use your netid-based wisc.edu email address to login to google. Doing all of this should allow you to subsequently access the Team Form linked to above. Please report any further problems on piazza for troubleshooting assistance.*
3. The TEAM FORM must be filled out before **6:00 pm on Thursday, March 15th** (when Milestone 1 is due).
4. **Both students must submit the team code to their own zyBooks account for grading.** For milestone 3, from zyBooks, we will pick one partner's copy of the program to human grade and assign both partners the same grade.
5. We suggest finding a partner at about your same programming knowledge and skill level.
6. We recommend working closely together (Pair Programming), not dividing up by milestones or steps. The assignment is designed to be progressive, each step building the previous. If you skip a step, it may be difficult to continue. Work closely together, discussing as you go.

Students are not required to work with another student on Big Program 1.

Milestone 1: Basic Interface and Supporting Methods.

1. Create a new project in Eclipse. (Battleship would be a sensible choice for the project name.)
2. Download Config.java,put in your project src folder and review the contents to become familiar with the constants.
3. Download Battleship.java and TestBattleship.java and save them in your project src folder.
4. **Best practice: *Start by writing the tests.***
    a. Add 2 more tests to the `testCoordAlphaToNum` method.
    b. Using the `testCoordAlphaToNum` method as template, write a method `testCoordNumToAlpha` to test the `coordNumToAlpha` method with at least 2 tests.
    c. OPTIONAL: This is not required for the BP1 assignment, but ideally you would also create test methods for `promptInt`, `promptChar` and `promptString`. These methods require a Scanner object. To do the automated testing, you can create a Scanner object from a String.
5. In this milestone, you will implement the methods: `coordAlphaToNum, coordNumToAlpha, promptChar, promptInt, promptStr,` and `initBoard.` The suggested approach is as follows:
    a. First implement `coordAlphaToNum`.
        i. Use the test bench to check your method.
        ii. Once it passes your test bench, submit to zyBooks to see if you pass zyBooks test 1.
    b. Then, implement `coordNumToAlpha`.
        i. Use the test bench to check your method.
        ii. Once it passes your test bench, submit to zyBooks to see if you pass zyBooks test 2.
    c. Then, implement `promptInt,and promptStr`. If you implemented tests, use your test bench to test.
    d. Finally, implement the `initBoard` method.
6. With the supporting methods written, you can write the basic main method with a play again loop. For Milestone 1, an algorithm for the main method is:
   Begin the play again loop
    - Call `promptInt` for the `"board height"`, where the minimum value is `Config.MIN_HEIGHT` and the maximum value is `Config.MAX_HEIGHT`.
    - Call `promptInt` for the `"board width"`, where the minimum value is `Config.MIN_WIDTH` and the maximum value is `Config.MAX_WIDTH, followed by a newline.`
    - Using the values read in for the height and the width, create the three 2d arrays required (see main method header for more details) and initialize them with `initBoard.`
    - Prompt the user with `"Would you like to play again? (y/n): "`, using the `promptChar` method.

   Exit the loop if the value of `promptChar` is not a `'y'`
7. Submit your **Battleship.java** and **TestBattleship.java** files to zyBooks for feedback and grading. We recommend you complete commenting and styling your code for each milestone as described in the Program Grading section of this document. Your highest scoring results prior to 6:00pm or (11:59pm with a 10% deduction) **Thursday, March 15th** will be recorded as your grade for this weekly milestone. Note: We may use different values in the Config.java for testing.

Milestone 1 Sample run: Milestone1.txt (interleaved input and output) [Milestone1In.txt, Milestone1Out.txt]

Milestone 2: Placing Ships

1. **Best practice: *Start by writing the tests.***
    a. Using the `testCoordAlphaToNum` method as template, write test methods with at least 2 tests in your test bench to test the `checkWater` and `placeShip` methods.
    b. OPTIONAL: This is not required for the BP1 assignment, but ideally you would also create test methods for `addShip`, `printBoard`, and `placeRandomShip`. Some of these methods require a Scanner object and a Random object. To do the automated testing, you can create a Scanner object from a String and a Random object from a specific seed.
2. In this milestone, you will implement the methods: `checkWater, placeShip, placeRandomShip, printBoard,` and `addShip.` The suggested approach is as follows:
    a. First implement `checkWater`.
        i. Use the test bench to check your method.
        ii. Once it passes your test bench, submit to zyBooks to see if you pass zyBooks test 1.
    b. Then, implement `placeShip`, use the test bench to check your method, and submit to zyBooks to see if you pass zyBooks test 2.
    c. Then, implement `placeRandomShip`, and submit to zyBooks to see if you pass zyBooks test 3.
    d. Finally, implement `printBoard,` and `addShip`. If you implemented tests, use your test bench to test.
3. With the supporting methods written, you can update the main method to place ships on the game boards. For Milestone 2, an algorithm for the main method is:
   Begin the play again loop
    - Call `promptInt` for the `"board height"`, where the minimum value is `Config.MIN_HEIGHT` and the maximum value is `Config.MAX_HEIGHT`.
    - Call `promptInt` for the `"board width"`, where the minimum value is `Config.MIN_WIDTH` and the maximum value is `Config.MAX_WIDTH, followed by a newline.`
    - Using the values read in for the height and the width, create the three 2d arrays required (see main method header for more details) and initialize them with `initBoard.`
    - Call `promptInt` for the `"number of ships"`, where the minimum value is `Config.MIN_SHIPS` and the maximum value is `Config.MAX_SHIPS`.
    - For each ship (start the index of the ships from 1), call the `addShip` method.
    - If any `addShip` method call fails, prompt the user with `"Error adding ships. Restart game? (y/n): "`, using the `promptChar` method.
    - Otherwise, if all the `addShip` method calls succeed, prompt the user with `"Would you like to play again? (y/n): "`, using the `promptChar` method.

   Exit the loop if the value of `promptChar` is not a `'y'`
4. Submit your **Battleship.java** and **TestBattleship.java** files to zyBooks for feedback and grading. We recommend you complete commenting and styling your code for each milestone as described in the Program Grading section of this document. Your highest scoring results prior to 6:00pm or (11:59pm with a 10% deduction) **Thursday, March 22nd** will be recorded as your grade for this weekly milestone.
   Note: We may use different values in the Config.java for testing.

Milestone 2 Sample run: Milestone2.txt (interleaved input and output) [Milestone2In.txt, Milestone2Out.txt]

Milestone 3: Playing the Game

1. **Best practice: *Start by writing the tests.***
    a. Using the `testCoordAlphaToNum` method as template, write test methods with at least 2 tests in your test bench to test the `takeShot` and checkLost methods.
    b. OPTIONAL: This is not required for the BP1 assignment, but ideally you would also create test methods for `shootPlayer`, and `shootComputer`.
2. In this milestone, you will implement the methods: `takeShot, checkLost, shootPlayer,` and `shootComputer.` The suggested approach is as follows:
    a. First, implement `takeShot`, test, then submit to zyBooks to see if you pass zyBooks test 1.
    b. Then, implement `checkLost`, test, then submit to zyBooks to see if you pass zyBooks test 2.
    c. Finally, implement `shootPlayer,` and `shootComputer`.
3. With the supporting methods written, you can update the main method to play the game. For Milestone 3, an algorithm for the main method is:
   Begin the play again loop
    - Call `promptInt` for the `"board height"`, where the minimum value is `Config.MIN_HEIGHT` and the maximum value is `Config.MAX_HEIGHT`.
    - Call `promptInt` for the `"board width"`, where the minimum value is `Config.MIN_WIDTH` and the maximum value is `Config.MAX_WIDTH, followed by a newline.`
    - Using the values read in for the height and the width, create the three 2d arrays required (see main method header for more details) and initialize them with `initBoard.`
    - Call the `promptInt` for the `"number of ships"`, where the minimum value is `Config.MIN_SHIPS` and the maximum value is `Config.MAX_SHIPS`.
    - For each ship (start the index of the ships from 1), call the `addShip` method.
    - If any `addShip` method call fails, prompt the user with `"Error adding ships. Restart game? (y/n): "`, using the `promptChar` method.
    - Otherwise, if all the `addShip` method calls succeed, run the game loop until there is a winner:
        - Display the game boards and start with the user by calling `shootPlayer`
        - Use `checkLost` to verify if the player has won. If so, print out `"Congratulations, you sunk all the computer's ships!"`, terminated by a new line, followed by the game boards.
        - If the user has not won, call `shootComputer` and check if the computer has won. If so, print out `"Oh no! The computer sunk all your ships!"`, terminated by a new line, followed by the game boards.
    - After the game, prompt the user with `"Would you like to play again? (y/n): "`, using the `promptChar` method.

   Exit the loop if the value of `promptChar` is not a `'y'` from either `promptChar` call.
4. Submit your **Battleship.java** and **TestBattleship.java** files to zyBooks for feedback and grading. We recommend you complete commenting and styling your code for each milestone as described in the Program Grading section of this document. Your highest scoring results prior to 6:00pm or (11:59pm with a 10% deduction) **Thursday, April 5th** will be recorded as your grade for this weekly milestone and used for the human grading.
   Note: We may use different values in the Config.java for testing.

Milestone 3 Sample run: SampleFinal.txt (interleaved input and output) [SampleFinalIn.txt, SampleFinalOut.txt]

Appendix

Debugging Tips

- Write your tests first!
- Compile, run tests and run the program often! Think about doing this with each code change and each completed method.
- Do your tests cover a breadth (ideally all) of possible scenarios?
- Do your tests check the boundary or edge cases?
- Do your tests cover valid, but unexpected, inputs?
- Make use of print statements! Think about printing out the computer primary board for debugging purposes.

Random Shot Algorithm

In the `shootComputer` method, the heuristic used by the computer is simply targeting a random cell. Moreover, the random procedure doesn't even limit its choices to cells that have yet to be targeted (which would be a natural way to speed up the running time of this algorithm). During your tests, unless you are extremely (un)lucky, you will never notice any delay in the computer picking a cell to target. Some natural questions to consider are: Could there be a long delay? And, how long will it take to randomly pick an untargeted cells when most have already been targeted?

Since it is a random process, it is possible to experience a long delay (more than a second or two), but such an event has an extremely low probability (essentially 0). The reason is essentially due to the answer to the second question of "how long will it take to randomly pick an untargeted cells when most have already been targeted?" Let's consider the worst case: there is one remaining untargeted cell and it will be the last one we choose randomly. How long do we expect this to take? This problem is called the Coupon Collector's problem and it will take, on average, around n ln(n) random draws to have drawn each cell at least once. That means, in a 10 by 10 grid with 100 cells, we would expect to have drawn every cell at least once after about 460 draws which happens very quickly on a computer. A 100 by 100 grid with 10000 cells is still only about 92200 draws which still happens very quickly on an average computer today that can process 50000 million instructions per second or more.

Extensions

The are many ways your might consider improving this Battleship program. If you are interested in pursuing these extensions or others that you might think of, please submit your fully completed Milestone 3 **before** attempting any extensions. The instructional staff would be happy to review your extensions with you or answer questions regarding extensions, but **do not submit your extensions**.

Some possible extensions would be:

- Better Interface: The interface is very simple and it could be improved. In particular, it might be nice to provide direct feedback on hits and misses; or to have the game boards side-by-side.
- Smarter AI: The computer player uses the simplest possible heuristic. Its game play could be improved significantly by:
    - Reacting to hits: After a hit, the computer could target the adjacent cells.
    - Improved searching strategy: It has been shown that, when searching for targets on a plane, the best strategy is a spiral search pattern.

- Multiplayer: Add in an option to play with a friend! The easiest would be to base it on the honour system where each player turns away from the screen when the other player is placing a ship or taking a shot.

Change Log

- Initial version: 2018-02-27
- Corrected second link to Team Form: 2018-03-02
- Added missing details to Smarter AI in Extensions in the Appendix: 2018-03-13