Big Program 1: MasterMind

This project is to write a code-breaking game based on the board game MasterMind and adapted to utilize specific concepts focused on in this course. This game was invented by Mordecai Meirowitz in 1970 and resembles an earlier game called Bulls and Cows.

In our implementation the computer (codemaker) generates a code and the user (codebreaker) guesses the code. In the board game there are 4 positions in the code that each can contain one of 6 colored pegs. The codebreaker tries to guess the pattern within 10 guesses. Each guess is made by placing 4 colored pegs in the 4 positions. The codemaker then provides feedback by indicating the number of correct colors in correct positions (black hits) and the number of correct colors in incorrect positions (white hits).

Example from Milestone 3:

```
Enter seed (0 for unrepeatable): 1
Welcome to Master Mind!
I have a 4 symbol code using [1, 2, 3, 4, 5, 6].
Can you guess my code within 10 guesses?
Enter guess (? for help): 1122
 1) [1, 1, 2, 2] B
remaining possibilities: 256
Enter guess (? for help): 1234
 2) [1, 2, 3, 4] BW
 1) [1, 1, 2, 2] B
remaining possibilities: 56
Enter guess (? for help): 2435
 3) [2, 4, 3, 5] WWW
 2) [1, 2, 3, 4] BW
 1) [1, 1, 2, 2] B
remaining possibilities: 4
Enter guess (? for help): ?
 4) [4, 5, 2, 4] BBBB
 3) [2, 4, 3, 5] WWW
 2) [1, 2, 3, 4] BW
 1) [1, 1, 2, 2] B
remaining possibilities: 1
Congratulations! You guessed code with only 4 guesses and 1 hints!
```

In the example:

```
2) [1, 2, 3, 4] BW
```

shows the 2nd guess was 1234 and the feedback was 1 black hit and 1 white hit. A black hit is a symbol in the correct position. A white hit is a symbol used in the code but is in the wrong position. The hits don't show which position is the black hit and which is the white hit. When we reach a solution, we can see the 4 was the black hit and the 2 was the white hit. The computer help capability (?) and the "remaining possibilities" is implemented in milestone 3.

Program Grading

GradingBigPrograms.pdf - Be sure to read this fully as this describes how your assignments will be graded.

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. You will need your partner's first name, last name, cslogin and @wisc.edu email address.
    A. *On You Need Permission error: https://cs200-www.cs.wisc.edu/wp/google-form-permission-error/*
3. The TEAM FORM must be filled out before **6:00 pm on Thursday, March 14th** (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 on 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: Supporting Methods

1. Create a new Java Project in Eclipse. You can name this project whatever you want, but MasterMind is reasonable. Next download MasterMind.java, TestMasterMind.java, and Config.java files (try right-clicking to "Save Link As...") and copy them into the src folder within your new Java Project. You may need to select Refresh from the File menu (or press F5), if you cannot immediately see these files in Eclipse's Package Explorer.
    a. MasterMind.java contains the method definitions for you to complete throughout the project.
    b. TestMasterMind.java contains some testing methods with some tests implemented for you.
    c. Config.java contains some constants to use within the program. A separate Config.java will be used for testing and may have different values for the constants.
2. Suggested Process: complete the methods indexOf, promptInt, generateHiddenCode, isValidCode and promptForGuess
    a. Read Method Requirements
        i. Read the method header for a method (initially indexOf) to learn what the various requirements such as what the method does, the parameters, a return value (if any), and whether information is output.
    b. Prepare Tests
        i. Trace and explain the corresponding provided test case(s) in TestMasterMind (for the indexOf method, testIndexOf).
        ii. Write additional test cases for the method.
            1. Do your tests check the edge or boundary cases?
            2. Do your tests cover a breadth (ideally all) of possible scenarios?
            3. For full credit, write at least 1 test case for each testing method.
        iii. Writing the test cases first is called test driven development.
    c. Design Algorithm
        i. Once you have determined example inputs and outputs (test cases), write an algorithm in words (pseudocode) to compute the appropriate result based on the parameters.
    d. Write Code
        i. Write the code for the method (initially indexOf) by translating your algorithm into Java.
    e. Test and Debug Method
        i. Test your method (for example indexOf) by uncommenting the call to the test method (for example testIndexOf) in TestMasterMind.main and run the TestMasterMind.main method.
        ii. If a test case fails then debug, first making sure the test case code is correct and then the method itself. A helpful technique is to use temporary print statements to show the contents of variables so that you can verify the values are correct.
        iii. If the test cases are thorough then your method is thoroughly tested and if the tests are minimal then your method is minimally tested. Another method that calls your method will also likely fail if your method fails.
    f. Repeat with the next method.
3. Style and Commenting: Comment and style your code for each milestone as described in the Program Grading section of this document.

4. Submission: Submit your MasterMind.java and TestMasterMind.java files to zyBooks for feedback and grading. For testing purposes, we will use our own Config.java that may have different values. This milestone is due **Thursday, March 14th at 6pm, see the Program Grading section for details on bonus points and late penalties.**

Milestone 2: Playing Game

1. In this milestone, implement these methods: countAllHits, determineHits, printBoard and main and add additional test cases to the corresponding testing methods testCountAllHits and testDetermineHits (at least one test case each for full credit).
2. Use the same suggested process described in Milestone 1.
3. Debugging Tip:
   The DEBUG output is to help reveal internal state for your testing and debugging purposes. We suggest adding your own debug statements to generate similar output.
   ```
   if (Config.DEBUG) {
         System.out.println("DEBUG value: " + value);
   }
   ```
   We won't be testing your DEBUG output.
4. Output produced by printBoard and main may be tested by comparing to M2-1.txt, M2-2.txt with DEBUG set to true and M2-3.txt, M2-4.txt with DEBUG set to false. You may compare by hand or with a tool such as DiffChecker.
5. Style and Commenting: Comment and style your code for each milestone as described in the Program Grading section of this document.
6. Submission: Submit your MasterMind.java and TestMasterMind.java files to zyBooks for feedback and grading. This milestone is due **Thursday, March 28th 6pm, see the Program Grading section for details on bonus points and late penalties.**

Milestone 3: Computer Hint

1. In this milestone, implement methods to provide optional help to the player (codebreaker). This algorithm enumerates all possible codes and then eliminates codes based on the feedback (hits for each guess). With a small number of possibilities this works well. However, if the number of code positions and the number of symbols increases a small amount the number of possibilities increases rapidly. With 6 symbols and 4 positions there are 6*6*6*6 == 6^4 == 1,296 possible codes. With 8 symbols and 6 positions there are 8*8*8*8*8*8 == 8^6 == 262,144 possible codes. This is referred to as exponential growth which can be a challenge both with space for the possibilities and time to search through them. This is a challenge that is commonly faced by computer scientists and requires creativity and work to come up with solutions.
2. Design, Implement and Test the following methods following the same process suggested in Milestone 1: nextCode, enumeratePossibilities, updatePossibilities, computerGuess and complete the Milestone 3 parts of the main method in MasterMind.java.
3. Example Output: M3-1.txt, M3-2.txt.

4. TODO (something to be done): Search through your code for TODO items and make sure they are complete and then the TODO should be removed.
5. Style and Commenting: Comment and style your code for each milestone as described in the Program Grading section of this document.
6. Submission: Submit your MasterMind.java and TestMasterMind.java files to zyBooks for feedback and grading. We will supply our own version of Config.java for testing. This milestone is due **Thursday, April 4th at 6pm, see the Program Grading section for details on bonus points and late penalties.**
7. **Congratulations on completing your first Big Program in CS 200 Programming I!**
8. Offline & Human Grading: We will download the highest scoring version of your code turned in to zyBooks by 11:59 pm on the due date for this milestone for comprehensive offline tests and human grading. A detailed report will be emailed to you within 2 weeks that includes the results of the offline tests and the human feedback.
9. Regrade Request: Within one week from when the report is emailed, you will have the opportunity to request the grader to consider some circumstance. Suggesting a very minor change such as one line of code (and deducting a couple of points) may result in many additional tests passing. A regrade is Not an opportunity to add comments, change names, or make other changes.

Extensions

You may have ideas on how to improve this MasterMind program, such as a better computer hint algorithm. If you are interested in pursuing these ideas 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**.

References

Some resources that were helpful when designing this project include:

- The Computer as Master Mind by Donald Knuth

Change/Clarifications Log

- Initial version: 2019-3-5
    - Refresh this page periodically for any changes or clarifications.