EE422C Final Project
Making Software a Game

Due Dec 4th, 2015

Projects must be checked by your TA no later than Tuesday December 8th

Purpose and Goals

The purpose of this assignment is to design and implement an OO program with multiple classes to play a game (e.g., Mastermind). During this assignment, you will demonstrate all the software development techniques that you've learned this semester. You'll then present your work to your TA. Since the goal of the assignment is for you to show off what you've learned, the project is deliberately left vague. You fill in the blanks in the way that makes sense. However, we are going to be looking for very specific indicates of your skill and knowledge as a software developer, including

- Appropriate design methods, e.g., MVC architecture, polymorphic programming, good encapsulation, clear interfaces.
- Use of data structures appropriate for the computation required
- Strong algorithm development and software coding skills.
- Robust software design with exception handling
- Graphical user interfaces
- Quality assurance demonstrated with a robust set of unit and/or system tests

During checkout with the TA, you will be asked to demonstrate how you achieved each of the above criteria. Keep in mind that the burden of proof is on you to show the TA what you've done, so plan accordingly.

General Requirements

You are encouraged to develop your final project in teams of two persons. You may also work alone one the project if you wish. Teams of more than two persons are not permitted. If you do work with a partner, you may decompose the work of the project in any manner you see fit.

The project description below focuses on the Mastermind board game. If you would like to develop a different game, you may do so provided you first obtain permission from your TA. The TA will evaluate your proposed game to determine if your game has complexity similar to or greater than that of Mastermind. As an example, the game Tic-Tac-Toe lacks complexity and would not be an acceptable substitution. A game like backgammon would be a reasonable choice. If you plan to develop a game other than Mastermind, then you MUST receive approval from your TA prior to the Thanksgiving break. If you leave for Thanksgiving without your TA granting consent, then you must use the Mastermind game for your project.

If you are working with a partner, and you and your partner are assigned to different TAs, then you must select a TA to supervise your project. Note that JeHo has the most students assigned to him among the TAs and checkout scheduling with JeHo will be more difficult as a result of his workload.

The version of the game you implement will have the following properties.

- The game must include two user interfaces. One interface must use an (at least minimally) attractive and easy-to-use graphical user interface (you will be evaluated partly on how nice your GUI is). The other interface must be a fully functional interface designed solely using System.out and System.in (i.e., the console and the keyboard). Keep in mind that the purpose of requiring two interfaces is for you to show how you can encapsulate various elements of your design and create a modular software system. If you simply write two independent programs, one with a GUI and one with a console interface you will not receive credit for this element of the project.
- When playing the game, the human player can play against the computer. The computer does not need to be a good player. In an asymmetric game (like Mastermind), the computer must be able to play either role (e.g., the computer must be able to play the game where it tries to guess your secret code).
    - Depending on the type of game you select, your TA may determine that the game should permit two humans to play against each other. For example, if you were to select the game chess, the TA may determine that it is not necessary that the computer be able to play the game, only that the computer know all the rules for the game.
- In addition to the software itself (i.e., the Java code), you are required to additionally submit design documentation and software tests for your project. You should discuss the format of the documentation with your TA prior to the deadline for the project. Design documentation may be a mix of Javadoc, class diagrams and use-case-diagrams. Software tests should be a mix of unit tests and system tests.

Scoring

Your project will be scored using the following breakdown.

- Graphical User Interface (ease-of-use and attractiveness): 20%
- Console Interface: 10%
- Software design quality (MVC architecture, good classes and interfaces, use of inheritance as appropriate, well encapsulated code): 20%
- Software testing: 10%
- Correct functioning of the game in all playing modes (e.g., human against computer, computer against human): 40%

In addition to the above, for games other than Mastermind, your score will be adjusted by a game-complexity factor. The game complexity factor will be subjectively rated on a four-point scale by your TA. The appropriate scale is then multiplied by your raw score to achieve your final project score

- Very low complexity (e.g., tic-tac-toe): scale factor 0.5
- Low complexity (limited algorithm complexity and/or no need for data structures – i.e., substantially easier than Mastermind): scale factor 0.9
- Standard complexity (e.g., Mastermind): scale factor 1.0
- High complexity (e.g., Chess): scale factor 1.1

Project Requirements for Mastermind Game

You can find a description of the Mastermind game at Wikipedia
http://en.wikipedia.org/wiki/Mastermind_(board_game)

- The player has 12 guesses to guess the code correctly.
- If the player does not guess the code correctly in 12 or fewer guesses they lose the game.
- The secret code consists of 4 colored pegs in specific position order.
- The valid colors for the pegs are blue, green, orange, purple, red, and yellow.
- The results of a guess are displayed with black and white pegs. The Wikipedia article refers to the results as feedback.
- A black peg indicates one of the pegs in the player's guess is the correct color and in the correct position.
- A white peg indicates one of the pegs in the player's guess is the correct color, but is out of position.
- A peg in the guess will generate feedback of either: 1 black peg, 1 white peg, or no pegs. A single peg in the guess cannot generate more than 1 feedback peg.
- The order of the feedback does not give any information about which pegs in the guess generated which feedback pegs.
- The player's guesses must be error checked to ensure they are the correct length and only contain valid characters.

The following example illustrates one way in which the console-based UI could work. Note that in the example, the human is playing against the computer. Recall that it must also be possible for the computer to play against the human (i.e., with the roles reversed where the human creates the secret code and provides feedback based on the computer's guesses). In the example below, the computer uses YBRY as the secret code.

```
Welcome to Mastermind.  Here are the rules.

This is a text version of the classic board game Mastermind.
The computer will think of a secret code. The code consists of 4 colored pegs.
The pegs MUST be one of six colors: blue, green, orange, purple, red, or yellow. A
color may appear more than once in the code. You try to guess what colored pegs are
in the code and what order they are in.   After you make a valid guess the result
(feedback) will be displayed.
The result consists of a black peg for each peg you have guessed exactly correct
(color and position) in your guess.  For each peg in the guess that is the correct color,
but is out of position, you get a white peg.  For each peg, which is fully incorrect, you
get no feedback.

Only the first letter of the color is displayed. B for Blue, R for Red, and so forth.
When entering guesses you only need to enter the first character of each color as a
capital letter.

You have 12 guesses to figure out the secret code or you lose the game.  Are you
ready to play? (Y/N):  Y

Generating secret code ....
```

```
You have 12 guesses left.
What is your next guess?
Type in the characters for your guess and press enter.
Enter guess: OOOO

OOOO ->  Result: No pegs

You have 11 guesses left.
What is your next guess?
Type in the characters for your guess and press enter.
Enter guess: oooo  -> INVALID GUESS

What is your next guess?
Type in the characters for your guess and press enter.
Enter guess: kkkk -> INVALID GUESS

What is your next guess?
Type in the characters for your guess and press enter.
Enter guess: RRRRR -> INVALID GUESS

What is your next guess?
Type in the characters for your guess and press enter.
Enter guess:     -> INVALID GUESS

What is your next guess?
Type in the characters for your guess and press enter.
Enter guess: RRRR

RRRR -> Result:  1 black peg

You have 10 guesses left.
What is your next guess?

(Etc. etc., . . . )

What is your next guess?
Type in the characters for your guess and press enter.
Enter guess: YRBY
YRBY -> Result:  4 black pegs – You win !!

Are you ready for another game (Y/N): N
```

Part of the assignment grade will be determined by how easily your program could be altered to allow a different maximum number of guesses, or a different number of pegs in the code and how easily new colors could be added, assuming they start with a different letter than other existing colors. (Up to 10) For example how easily would it be to change the code to have 5 pegs and allow pegs to be the color Maroon?

In addition to the simple case identified above, recall that you must provide a GUI (e.g., see http://www.web-games-online.com/mastermind/index.php

You must also allow a mode of play where the computer attempts to guess a secret code selected by the human. In this form of play, the human should not reveal the code to the computer, and should only provide feedback on the computer's guesses. The computer should, nevertheless, be able to detect when the human is lying. For example, if the computer has made two guesses in which all six possible peg colors are included, and the human provides the "no pegs" feedback to these two guesses, then obviously the human is lying. The computer should be capable of detecting when the feedback provided by the human is inconsistent with all possible secret codes. As soon as the computer has detected that there are no possible secret codes that match the feedback already provided, your program must provide some indication that the human was caught cheating.

Tips and Hints for Mastermind

The key thing algorithmically about Mastermind is the feedback that results from a secret code and a guess. When creating this feedback, it is important to realize black pegs should be generated first and that a given peg from the secret code and the guess cannot be reused once matched.

Here are some examples:

Code: RRRR
Guess: BBBB

This one is easy. There aren't any B's in the code, so the guess of BBBB would generate a result with no pegs.

Code: RRRR
Guess: RBBY

The R in the first position of the guess matches up exactly with the R in the first position of the code. This generates one black peg. The key is once a peg has generated a peg it cannot be used to generate any more. So the R in the first position of the code and the first position of the guess cannot generate any more pegs. The guess of RBBY generates a result with 1 black peg and 0 white pegs.

Here is another example.

Code: RBYR
Guess: BBRG

This code and guess would give a result of 1 black peg and 1 white peg. Black pegs take precedence so the B in the second position of the code and guess result in a black peg. One useful way of solving this problem is to determine the black pegs and scratch out or replace the pegs that were used. So given the Code of RBYR and the Guess of BBRG there could be a temporary result of

Code: R-YR

```
Guess: B-RG
```

To get the number of white pegs in the result the remaining characters can be compared for matches, independent of position. If a match is found then those pegs should be scratched out as well. The B in the first position of the guess doesn't match any characters in the code. The "-" in the second position of the guess should be skipped. The r in the third position of the guess matches the R in the first position of the code. This is a peg in the guess that is the right color, but in the wrong position so it generates 1 white peg. Once used the R in the first position of the code and the R in the third position of the guess can be reused so they should be scratched out.

```
Code:  --YR
Guess: B--G
```

The G in the fourth position of the guess doesn't match anything so the result would be 1 black peg and 1 white peg. To summarize

```
Code:  RBYR
Guess: BBRG
```

generates a result of 1 black peg and 1 white peg.

In correctly guessing the code, it is helpful for you to be able to review the history of the guesses and replies. This represents the state of the Board. Therefore, you should implement a way to have the program output the history in order from the beginning whenever you want to see it. So, when the user types in “history” for the guess, the program would then display that history.

```
What is your next guess?
Type in the characters for your guess and press enter.
Enter guess: history
```

. . . (the history of all guesses and replies for this game are displayed in tabular format)

Warning - You may not acquire, from any source (e.g., another student or an internet site), a partial or complete solution to this problem. Except for your partner, you may not show another student your solution to this assignment.

Adapted from an assignment written by Herb Krasner, which in turn was adapted from an assignment written by Mike Scott.