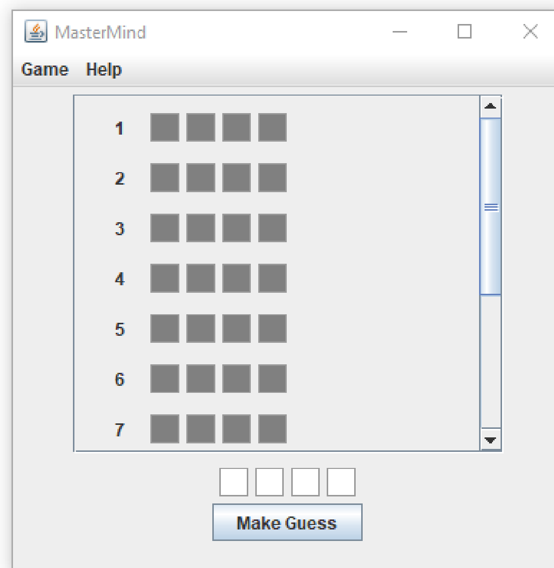
Object-Oriented Design
COMP 2000
Spring 2024

Lab 10.3

Mastermind Part IV

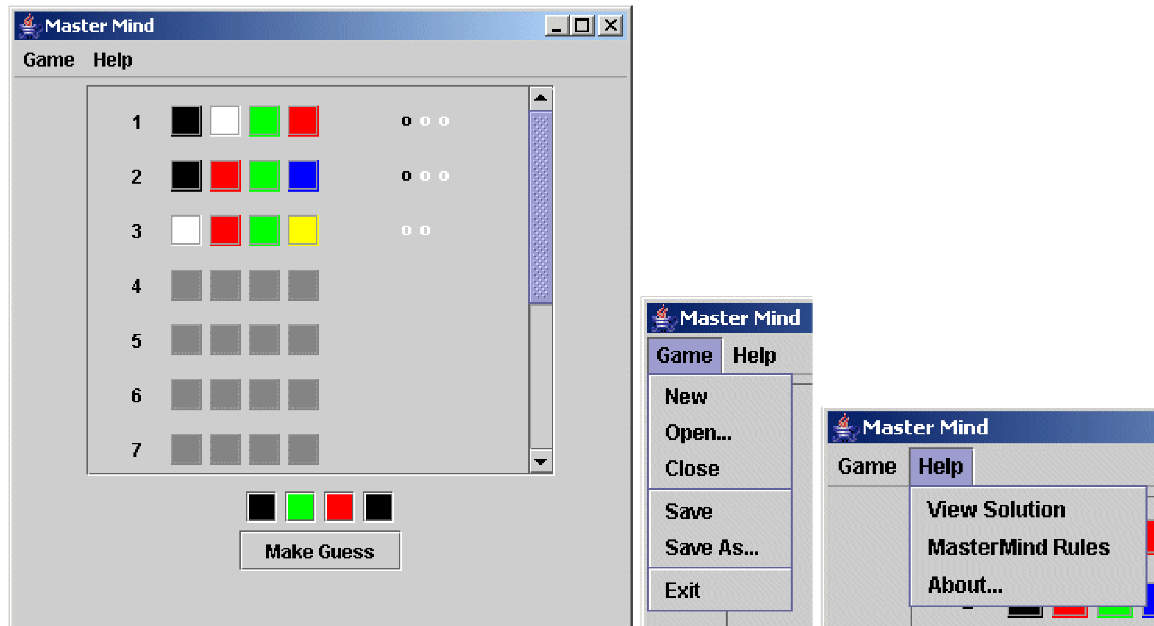
Part IV is entirely focused on implementing a new version of `MasterMind`, which represents the View and Controller parts of the MVC design. I will provide a working MMGame class for you to use on Friday after 10.2 is due. The GUI will provide game management and also display a history of guesses and feedback. All user interaction will be through the mouse and all guesses and responses will use colors instead of digits. When you run the completed program, the GUI should display a window that looks like this:



As you can see from this figure, the GUI has four basic sections. They are, from top to bottom,

1. **Frame Title: This is the title bar area across the top of the window. It is easily configured using JFrame methods.**
2. **Menu Bar: The menu bar contains two menus, Game and Help. They each have several menu options (pictured below).**
3. **Game History: This is a numbered list of previous guesses and feedback items. It is configured using nested panels with the outermost panel placed into a scrolling pane. The figure above shows its initial configuration in which each guess is shown as a gray box with no feedback.**
4. **Guess Input: This is the row of four boxes with the Make Guess button beneath. The four boxes are actually buttons that have no text. Each time one is clicked on, its background color changes. This is how the user specifies the sequence of colors for guessing. Each one is initialized to white.**

The first graphic below shows a game in progress, just before the user presses the Make Guess button to submit the fourth guess. The other two graphics show each expanded menu with all the menu options.



Game Play

To play the game shown here,

- Run the program or start a new game by selecting "New" from the "Game" menu.
- Click on the four boxes just above the "Make Guess" button to form your guess. They are initialized to white, which is one of the colors. Each time you click, the next color in the round-robin list of colors will fill the background. There is no way to make an invalid guess because only the allowed number of boxes and colors are shown!
- Click until all four boxes contain your guess then click the "make guess" button.
- Your guess will be evaluated, then it and the response will be displayed in the history section. The input boxes also revert to white.
- The response resembles the board version of the game. Each black "o" represents a "complete" and each white "o" represents a "partial".
- You enter guesses until either you guess the solution or run out of guesses. In either case a message box will appear with the appropriate message and the "make guess" button is disabled.

To Do

1. **Create a new project directory for MasterMind that is different than the one for Lab 10.2. Within the src directory create a mastermind package. Then download and unzip the two files `MasterMind.java` the `MMGame.class` into this folder.**
2. **Study the source code in `MasterMind.java` and make sure that you understand ALL of it.**
3. **Note: I have added a new method to `MMGame` called `getSolution()` that can be used to implement the View Solution option in the Help menu.**
4. **Follow the development plan below *very carefully!***

Development Plan

1. **Run the program to see what the existing code does.**
   The downloaded file should compile and run as is and will display a window as pictured on page 1.
2. **Create additional menus and menu items.**
   a. **Define all the menu-related objects as local variables in the `menuSetup()` method.**
   b. **The menu bar, Game menu, and Exit menu item are provided and fully operational.**
   c. **Add the additional menu and menu items as illustrated in the graphic on page 2.**
   d. **The horizontal separators within a menu are created by the `addSeparator()` method.**
3. **Define 'stub' action listeners for each of the menu items.**
   a. **An action listener is defined for "Exit". You can follow this as a template pattern for the menu items you just created. A 'stub' listener just prints an informative message.**
   b. **Create a private inner class `MenuItemStubListener` that implements `ActionListener`, to handle events for remaining menu items.**
   c. **Inside its `actionPerformed()` method, call `event.getActionCommand()`, where `e` is the parameter, to retrieve a String containing the menu item text for the selected menu item. Then print this text in an appropriate message to `System.out`. Example message: Menu item Save is not implemented.**
   d. **Back in `menuSetup()`, create a `MenuItemStubListener` object and add it to each of the menu items. The same object can be added to all the menu items.**

STOP! Test all the above to make sure it works before proceeding. Each each menu item should be connected to the action listener and display an appropriate message when selected.

4. **GUI objects for obtaining a user guess**
   a. **Declare two instance variables:**
      i. `private Jbutton[] inputButtons;` **// array of buttons that will make up the guess**
      ii. `private Jbutton makeGuess;` **// button for submitting a guess**
      iii. **Remember: instance variables are declared outside any method at the top of the class**

b. **Define two action listener classes:**
   i. `MakeGuessListener` **// for the guess submission button**
   ii. `InputButtonListener` **// for the array of guess buttons**
   iii. **Write 'stub'** `actionPerformed()` **methods for both of these**

STOP! Do not proceed until everything compiles correctly!

c. **The next five steps will implement the** `inputSetup()` **method (a stub version is provided)**
   i. **Note that I have declared and created a** `JPanel` **object called** `inputButtonPanel` **to hold all the buttons.**
   ii. **Instantiate the** `inputButtons` **array to have** `NUM_PEGS` `JButton` **elements.** `NUM_PEGS` **is provided and is currently set to 4.**
   iii. **Compose a loop to create the** `NUM_PEGS` `JButton` **objects. In each iteration of this loop, a** `JButton` **object will be created and assigned to an element of the** `inputButtons` **array. The button will have no label. Add an** `InputButtonListener` **object to the button (all buttons can share the same listener object, so create it above the loop). Set its background to** `colors[DEFAULT_COLORS_INDEX]`**. The array** `colors` **is provided.**
   iv. **Create and assign the "Make Guess"** `JButton`**. Add a** `MakeGuessListener` **to it.**
   v. **Add all the buttons to** `inputButtonPanel`**.**

STOP! Test all the above to make sure it works so far before proceeding! The input buttons should be white, and the two stub action listeners should respond with whatever messages you have programmed into them.

d. **This is a good time to fill in the** `actionPerformed()` **for the** `InputButtonListener` **class. Here's what it needs to do:**
   i. **Get a reference to the button that was clicked. Use the event's** `getSource()` **to do this. Note: you will need to use type-casting to assign the return value to a** `JButton` **variable.**
   ii. **Get the button's background color using** `getBackground()`**.**
   iii. **Determine which index in the colors array contains this background color. I strongly recommend you write a private method that has one parameter to receive the background color, and returns the array index for that color. Its algorithm will require a small loop to find the match. This method will also be needed below.**
   iv. **Change the button's background color, using** `setBackground()`**, to the next color in the array. Don't forget to wrap around when you get to array index** `NUM_COLORS`**! The % operator is good for this.**

STOP! Test all the above to make sure it works so far before proceeding! Each of the input buttons should independently cycle through the array of colors. It should only cycle through the first NUM_COLORS elements of the array.

e. **Finally, write the action listener for the "Make Guess" button. Take it in phases:**
   i. **Start by transforming the state of the input buttons into a suitable string for the MMGame's `guess()` method. Do that by composing a loop to traverse the `inputButtons` array. In each loop iteration:**
      - **get the button's background color and determine its array index in the colors array. Hey, didn't you just do the same thing above? If you defined it as a private method as I recommended, you can just call that method here.**
      - **Convert this array index to a string and concatenate it to the end of your guess string.**

   ii. **After completing the loop, you should have a string containing the guess in a format that the MMGame class understands. Now you can interact with MMGame and produce some test output. The provided code includes an MMGame object called game. Invoke its methods and print the return values to `System.out`, in particular `getSolution()`, `guess()`, `numComplete()`, `numPartial()`. Test everything thoroughly until you are convinced it works correctly.**

   iii. **Add the logic and interaction for finishing a game. Call `game.isGameOver()` and if it returns true, then show a congratulatory `JOptionPane.showMessageDialog()` to the user! If `game.isGameOver()` returns `false` and `game.getNumGuesses()` returns `MAX_GUESSES` then show a conciliatory `JOptionPane.showMessageDialog()` to the user. In either of these situations, disable the "Make Guess" button using its `setEnabled()` method with argument `false`.**

   iv. **Reset the input button colors to all white after each move is processed.**

STOP! Test all the above to make sure it works so far before proceeding! This is a major milestone! Pat yourself on the back and get ready for more.

5. **Make sure that you understand all the provided code that is in the `historySetup()` method.**
6. **Moving a Guess into the History**
   a. **Program the "Make Guess" action listener to record each guess and the game's response into the history by manipulating the colors of the JButtons and JLabels in the appropriate row of the history and response arrays. These are set up in `historySetup()`, so read it now if you haven't yet done so. You'll know what row to use by retrieving the number of guesses from the provided `MMGame` object called game.**
   b. **For each button in the selected row of history, set the background color to the background color of the corresponding guess input button. For labels in the selected row of response, set the foreground color to `Color.black` for each of `game.numComplete()` and set it to `Color.white` for each of `game.numPartial()`. In other words, if the number complete is 1 and number partial is 2, set one of them to black, two to white, and leave the fourth one alone.**

7. **Reset the History when the Game Ends**
    a. **Section 4.e.iii, above, described what to do when a game is over (due to correct guess or out of guesses). In particular, the program is to display a message dialog. For this section, you are to extend your program to visually reset the history and response elements after the user responds from the message dialog. For all the history buttons, set their background color to `Color.gray`. For all the response labels, set their foreground color to be the same as their background color.**

8. **Extra Credit: if you are up for something less challenging, implement a few menu item action listeners:**
    a. **New: should reset the history to its initial visual state (see step 7) and create a new MMGame object**
    b. **Close: For now, disable the "Make Guess" button and reset this history and input buttons to initial visual state.**
    c. **View Solution: Call `game.getSolution()` to get the solution and then display it.**
    d. **MasterMind Rules: Display a description of how to play the game.**
    e. **About…: Use `JoptionPane.showMessageDialog()` to display a brief message with your name and a 2021 copyright. Also state that Mastermind is a registered trademark of Pressman Toy Corporation.**
9. **Extra-Extra Credit**
    a. **Add additional menu items that allow the user to choose the number of pegs and number of colors (max 10 colors). You will need to make sure that if this breaks any of your other code that you fix any issues.**
    b. **(Only attempt this if you have everything else working) Implement action listeners for the Save, Save As…, and Open menu items that allow a user to save a game in progress to a file and the open it back up to resume play.**

To Turn In

Email `MasterMind.java` to prof. stucki as an attachment.