**Name of the game: MASTERMIND**

**1. Program Concept**

<Describe the overall concept of your game>

It’s master mind. It’s where the player guesses a code and is given hints along the way. Or, the computer tries to guess the player’s code.

<List the major functionalities that your program has, and briefly describe each of them>

|  |  |
| --- | --- |
| Functionality Name | Description |
| Add Guess Pegs | User can guess pegs each turn (if playing as guesser) |
| Remove Pegs | User can remove pegs in case they made a mistake (before submission) |
| Add Feedback Pegs | User can input feedback pegs (if playing as coder) |
| Input Code | User can add the code for the computer to guess (if playing as coder) |
| Adjust Difficulty | User can change difficulty of code generation (if playing as guesser) |
| Console Mode | User can play game on console, instead of a GUI |
| Check Feedback | Computer can check the feedback pegs of the user to see if they are right, the user will be notified if its wrong, and can change or not change it |
| Check Pegs | Computer can check if the pegs the user guessed are correct, and output the write feedback pegs |
| Reset Game | User can reset the game. |
| Change difficulty | User can change the difficulty of the generated code they have to guess. |

**2. Code structure**

**2.1 Class diagram**



**2.2 Briefly describe the overall structure of your code**

<Describe your program structure with consideration on: MVC architecture, polymorphism, encapsulation of classes, and use of interfaces>

I used polymorphism for the pegs in order to get their color image without knowing what pegs they are. I used an abstract class for the Peg, and then went down into another class with FeedbackPegs. I had a class for GUI stuff, and also a class for alert windows. I had a class for the mastermind game, as well as a class for all the data used in the master mind game.

**2.3 What are the data structures you used? Why did you used that specific data structure?**

|  |  |
| --- | --- |
| Data Structure | Explanation |
| ArrayLists | I used arraylists to store the peg guesses or feedback pegs. The arraylists don’t store the individual pegs, but the entire peg set (Peg[] or FeedbackPeg[]). I used arraylists because it’s very easy to add or take away sets of pegs (which is crucial for the function of the game). |
| Arrays[] (Peg[] and FeedbackPeg[]) | I used simple arrays to store the peg sets. I didn’t want to use arraylists or any type of complex data structure because I would end up having to do an arraylist of an arraylist for storing peg sets, which would be too confusing to use. Arrays are easy to manipulate, minus the fact that add and taking away elements needs some extra code. I do not need a complex data structure for peg sets because taking or adding pegs to the peg set rarely happens (and if it does it won’t take long to write code to do that for). |

**2.4 What is the major algorithms of your program?**

|  |  |
| --- | --- |
| Algorithm | Explanation |
| Code Generation (hard) | While generating the code, it will be completely random.  Each peg will be given a random color.  Then the peg set will be returned as the code. |
| Code Generation (easy levels) | While generating the code, it will only use a certain amount of colors  Each peg will be given a random color, from a random set of 2 or 3 colors only.  Then the peg set will be returned as the code. |
| AI Code Guesser (easy) | While the computer guesses the code, it will be completely random.  The AI does not account for the feedback pegs while guessing.  Each peg will be given a random color.  Then the peg set will be returned as the guess. |
| Feed back peg generation | This algorithm generates the correct feed back pegs (black or white) for a given guess.  It first determines all of the white pegs. This is defined as a peg that is the right color or color AND position.  It then determines all the black pegs, this is defined as all of the pegs in the write color AND position.  It then adjusts the number of white pegs by subtracting the black pegs number from it.  It then returns the feed back pegs. |

**3. Test**

**3.1 Unit tests**

<List the unit test cases for your program, and briefly describe the purpose of each of them.>

|  |  |
| --- | --- |
| Test Case Name | Purpose |
| Feedback Generation Test | Purpose: To determine if the generated feedbacks are indeed correct for a single solution.  Solution Code: RBGY  Input1: BBBB  Output: Black  Input2: BRBB  Output: White  Solution Code: RRGG  Input1: RRGG  Output: Black Black Black Black  Input2: GGRR  Output: White White White White  Input3: BBBB  Output: None |
| Feedback Check Test | Purpose: To determine if the user’s hint is correct  Solution Code: RRGG  Guess Input: RRBB  Input1: Black Black  Output: Correct feedback  Input2: None  Output: Incorrect feedback  Input3: White White  Output: Incorrect feedback  Guess input: YYYY  Input1: None  Output: Correct Feedback  Input2: Black  Output: Incorrect feedback  Guess Input: GGRR  Input1: White White White White  Output: Correct feedback  Input2: None  Output: Incorrect feedback |
| Undo | Purpose: See if undo works  Input: Input all the pegs, then before submission, undo them all.  Output: Program should allow for pegs to be put in, and the turn should be the same. |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |

**3.2 System tests**

<List the system test cases for your program, and briefly describe the purpose of each of them.>

|  |  |
| --- | --- |
| Test Case Name | Purpose |
| <functionality under test> | <describe the purpose, input, and expected output> |
| Check User Win Ability with 12 Turns | Purpose: See if the game allows a win in 12 turns  Input: Input the wrong input until the last turn, and the input the right input guess.  Output: Should allow win |
| Check User Lost Ability with 12 Turns | Purpose: See if the game lets the user lose after 12 turns  Input: Input the wrong input for every turn:  Output: Should allow lose |
| Check User Win Ability | Purpose: See if the game lets the user win when the user guesses correctly  Input: Input the right input for the first turn:  Output: Should allow win |
| Check if game can be scaled to allow more pegs | Purpose: See if the game can increase the amount of pegs a user/ai must guess  Input: Change the data field from 4 to 5  Output: Should see the game play with 5 pegs now |
| Check AI win ability within 12 turns | Purpose: See if the AI can win the game  Input: Make the AI input the right guess at any turn  Output: AI wins  Input: Make the AI input the right guess on the last turn  Output: AI Wins |
| Check AI Lost ability | Purpose: See if the AI can lost the game  Input: Make the AI input the wrong guess for all turns  Output: AI Loses |
| Check User Interface | Purpose: See if the user interface is intuitive  Input: Give a person who is not in 422c the game to play. The person will be given no instructions from me.  Output: The user should be able to figure out how to play the game, and use the interface intuitively. |
|  |  |