MaGiC APCS pd1
Gilvir Gill, Caleb Smith-Salzberg, Matteo Wong 2017-01-05

Final Project Proposal

MagicMind

Rules

MasterMind is a board game in which one player, the “mastermind,” comes up with a secret code of four colors. The game has a total of six colors to choose from, and the mastermind can choose any combination of those six, with repetitions allowed.
The opponent, or “guesser,” has to guess what the code is in 12 turns. After each guess, the mastermind gives them clues to say how hot/cold they are. Based on the pegs given, the mastermind will put down “pegs” that indicate the accuracy of the player’s guess. No peg means completely wrong. A white peg corresponds to the correct color, but in the wrong position. A red peg corresponds to a correct color in the correct position. However, the player doesn’t know which pegs correspond to which colors in their guess.
More in-depth explanation:
https://magisterrex.files.wordpress.com/2014/07/mastermindrules.pdf

Our Gameplay

User will be prompted in the terminal if they want to be the MasterMind of the guesser. Then they will either provide a code and watch the computer guess it, or be the guesser and try to guess the computer’s code. This will involve prompting from the terminal to input the next guess.

Tentative Classes

Game : This the underlying class to the two play
- Player guesser
- Player mastermind
- int[][] _board
//will be 12 by 4
- int[][] _pegs
//also 12 by 4
- int _turns
- int[] _correctAns
//length 4
+ makeGuess()
+ startGame()
+ getBoard()
+ getPegs()

Mastermind: Will employ scanner or system.in to recieve user input and give prompts assuming that the Player is a mastermind. Will be visible class where board is printed.

+ void printBoard()
+ void main()

Guesser: Will employ scanner or system.in to recieve user input and give prompts assuming that the Player is a guesser. Will be visible class where board is printed.

+ void printBoard()
+ int[] scanGuess()
+ void main()

Player (abstract): This regulates what methods and instance variables users and computers will have to use to manipulate the game board.

```
- int[] nextGuess
- int[] _finalAns
- int _currentTurn
- boolean _masterOrGuesser
- int[][] _gameBoard
- int[][] _pegBoard
// the local boards are important so that the computer algorithm can run

+ int[] setNextGuess()
+ int[] givePegs() //will be used to respond with number of pegs
+ void setMasterOrGuesser()
+ void setFinal()
```

User extends Player

User will define all of the methods in player to take inputs (that will come from the user in the terminal), these will depend on whether the user is the mastermind or guesser

Computer extends Player

```
+ int[] setFinal() //used to update final guess as algorithm progresses
+ void algorithm() //implements guessing algorithm, will use setNextGuess()
```

Computer Algorithm:

- Mastermind has a total of six colors to guess. For the purpose of this explanation the colors will be represented as 1-6, and the word “numbers” will be used in place of “colors” to avoid confusion
- The “slots” for the guess are 0,1,2,3, for an int array of length 4
- This algorithm operates under the premise of finding out what numbers exist and then where they go, and doing so two at a time for simplicity and to make elimination easier

1. the computer would guess [1,1,2,2]
    - By guessing two numbers you are able to see if those two exist in isolation from the others, making it simpler for the computer to do logical operations
    - The computer store how many pegs are given in response, meaning the amount of white and red pegs
2. Unless 1,1,2,2 gets a response of four pegs, the computer guesses [2,2,2,2]
3. Next, compare the number of pegs from the last guess to the amount of pegs in step one. There are a few options:
    a. Number of pegs stays the same and there are 2 or less, which means there are 2’s in slots 2 or 3 and no 1’s at all in the final answer
    b. Number of pegs stays the same and is 3, which means there are three 2’s and one 1
    c. Number of pegs decreases means: the number of pegs left after the second guess is the number of 2’s and the amount by which it decreased is the number of 1’s.
        i. This is not the case IF there are 3 or more 1’s but that is not a particularly difficult case to handle
    d. Number of pegs increases: there are 3 or more 2’s but no 1’s
4. Also check if the colors of the pegs change.
    a. If pegs change from white to red, that means that correct numbers have been moved to the correct place. This means there are 2’s in slot 0 or 1
    b. If pegs change from red to white, that means there were 1’s in slots 0 or 1
5. Once the computer knows how many 2’s or 1’s there are, it stores this, and in the slots where there could be 1’s will place them there. All the other slots remain 2’s so the amount and color of pegs doesn’t get messed up. Once a red peg is confirmed, that value is locked into the finalAns instance variable
6. Repeat step 5 for 2’s and keep all the other slots as 1’s
7. Repeat Steps 1-6 but with the pairs 3 and 4, then 5 and 6. This means that in the remaining slots the computer guesses 3’s and 4’s, then 5’s and 6’s. In the event of 3 slots the computer will put one of the former number and two of the latter

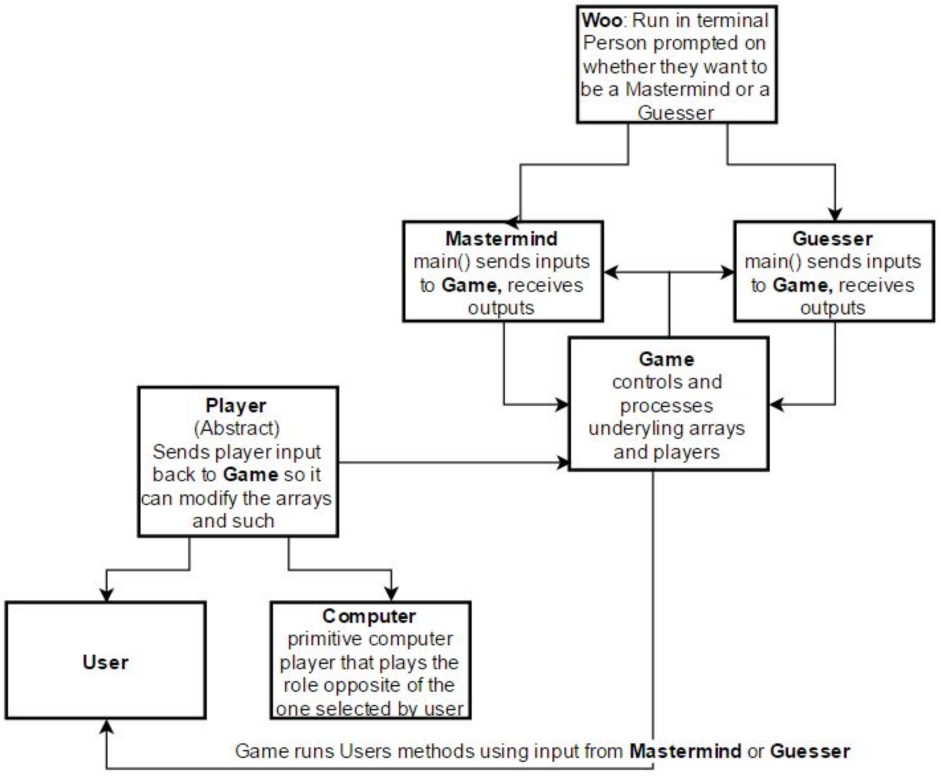
Example of Computer Algorithm

Correct: [4,6,1,1]

****Italics* refers to numbers or pegs you know with certainty are correct

guess	pegs	reasoning
1 1 2 2	WW	Standard first guess
2 2 2 2	none	Test changes in pegs
2 2 1 1	RR	The number of pegs decreased which means they both corresponded to 1's. No red pegs means they have to be in slots 2 or 3
3 4 *1 1*	W*RR*	Trying with 3 and 4
4 4 *1 1*	R *R R*	You switched the 3 to a 4 to see what changes. It changed color but stayed the same so you know there is a 4 and it can't be in slot 1
4 4 *1 1*	*RRR*	The 4 must be in slot 0.
4 5 *1 1*	*RRR*	Try 5
6 5 *1 1*	R*RRR*	Try 6. Victory

Flowchart of Classes


Woo: Run in terminal Person prompted on whether they want to be a Mastermind or a Guesser
Mastermind main() sends inputs to Game, receives outputs
Guesser main() sends inputs to Game, receives outputs
Game controls and processes underlying arrays and players
Player (Abstract) Sends player input back to Game so it can modify the arrays and such
User
Computer primitive computer player that plays the role opposite of the one selected by user
Game runs Users methods using input from Mastermind or Guesser