Mastermind

version 2.3

Stephanie Peacock

CIS-7

Fall 2024

Description

A code breaking game. A random four digit code is created, and the AI must determine what that code is.

GitHub

https://github.com/StephaniePeacock/CIS7

AI Algorithm

Phase 1 – Overview

During phase 1, the AI must locate all four digits of the code. Once all four digits have been found, we proceed to Phase 2. This phase uses static integer **pos** to keep track of the next digit to check, and static integer **indx** to keep track of index of the digits string. Both are reset to 0 before Phase 2 begins.

Phase 1 – Guessing Strategy

Each possible digit is checked, in numerical order. Beginning with 0000. If a digit is found, it is added to the next guess, and any remaining spots are filled with the next number. By process of elimination, if digits is fewer than 4 characters long when pos has reached 9, those positions are automatically filled with 9s, and we proceed to Phase 2.

Phase 1 – Locating Newly Found Digits:

A non-static int variable **num** is used to keep track of the newly found digit hits. Num sums the amount of right digit right place (**rr**) and right digit wrong place (**rw**) then subtracts the **total** number of digits already found. If num is more than 0, it has found a digit (or more) in the code, and it is added to the **digits** string. Num is then added to total, to be used in the next round.

Phase 2 - Overview:

Phase 2 is aimed at locating the correct placement of the digits in the code. The general process is checking each digit in the code in each possible location. The static variables **pos** and **indx** are reused, with pos tracking the position of the guess to be checked, and indx keeping track of the digit we are trying to find placement for.

Phase 2 – Condition to Assign a Placement:

Placement is assigned if and only if rr is 1 and rw is 0. If a placement is found, the digit at that indx is copied into placed[pos-1]. We subtract one from pos because it was incremented between filling and getting the results back. Pos is then reset to 0, and indx is incremented, so we can check the next index in the digits string.

Phase 2 – Ensuring Placement Condition:

To ensure we can achieve the conditions for placement, a digit is found that does not exist within the code. All other spaces except the pos being checked are filled with this wrong digit, assuring that when the location is found we will have rr ==1 and rw == 0.

Phase 2 – Preventing Guessing Known Placements:

To avoid checking placements that have already been found, a loop is used, that advances pos to the next position to check.

Phase 2 – Handling Double Digits:

When two digits are the same in the code, the check is only run once, until all locations for that digit have been found. Each time a digit is found, the indx is incremented, moving to the next index, but not resetting the position to check next.

Phase 2 – When Two Placements Have Been Found:

To potentially save some guess attempts, once the first two placements have been found the last two digits are placed directly into the unknown spots. In preparation for the placement being incorrect, their placement within the digits string is swapped, assuring the correct placement if needed.

I chose to place them with the last digit in the digits string first, as there is a possibility of duplicating a guess if they are placed smallest to largest (due to how the Phase 1 guesses are formed). An example code that with would occur is the code 1176. The algorithm always fills the digits string from smallest to largest. During the course of finding all the digits, we would check 1167 (to locate the 7). If the smaller digit was placed into the first unplaced position, this guess would be repeated, wasting a guess. I chose to flip the order of insertion to avoid instances like 1176, where the unflipped attempt would have already been made. There is a 50% chance that the order selected is the correct one. If it is not, then we retry with the other order.

Maximum Solve Attempts:

The maximum solve attempts is 18 attempts, although a specific pattern must be met for this to occur:

- There can be no duplicate numbers.
- The second digit must be greater than 7 and must be the largest number in the code.
- The last digit must be the smallest number in the code.
- The first digit must be the second largest number in the code.
- Examples that fit this pattern: 2810, 7865, 6852.

The reason this is 18 instead of 17 is because of the decision to flip the values of the last two unfound digits above.

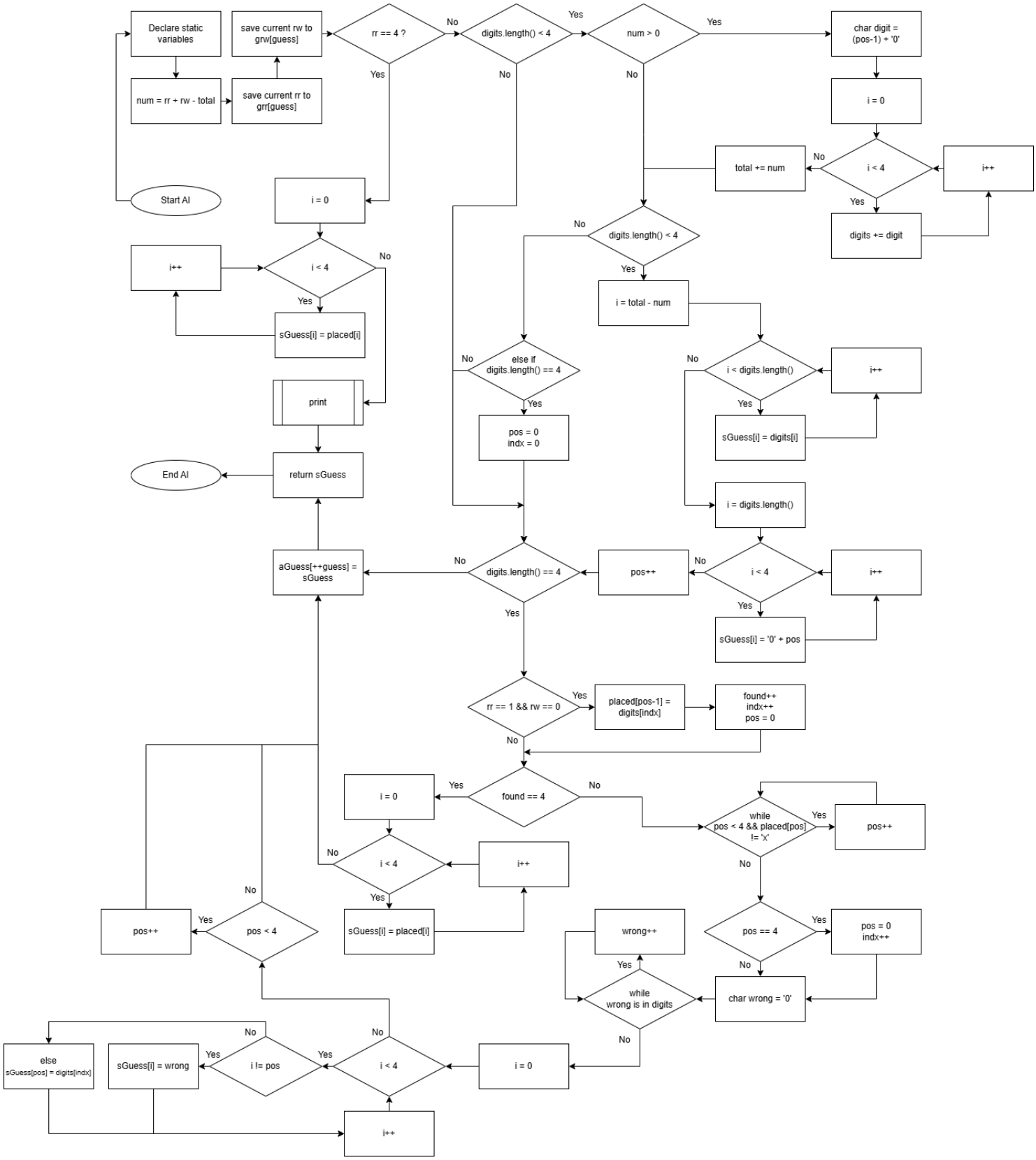
Version Progress:

Version 2.1 – This version checked in the dumbest way possible. It began with 0000 and cycled through each digit until 9999. Then it checked every index of digits in each position.

Version 2.2 – This version skipped placed positions, but still checked the fourth digit, instead of assigning it.

Version 2.3 – This is the final version, skipping checking 9999, as well as only finding the first 2 placement before adding the last two in together, swapping them and running again if the first attempt at random placement failed.

AI Flowchart



Pseudocode

```
// Print Helper Function Code
// Save the historical values of guesses and results
        // How many guesses to save
        // Save the guesses
        // Save right guess in right spot
        // Save right guess in wrong spot
        // Increment the guess number
        // To store the identified digits
        // How many placements found
        // Current guess string
        // Digits then indexes to test
        // To hold the placement
        // So we know how many digits we've found but not placed
        // Number of found digits to add
// Store the results from the last guess
// Test the helper function
        //We found all 4 placements
        //Copy placed into sGuess
        //Call Helper print function
        //We have our code, we can skip the rest of this & return sGuess

//Phase 1 - Identify the digits in the code
        //Digits has less than 4 characters
        //See if we need to add a digit to our guess – num > 0
        //Find the digit we just got results for
        //If they are, add that digit to digits that # times
        //Increment total digits found - existing total
        //Check if we need to prep the next guess (digits < 4)
                //Prep the next guess
                //Add the known digits to guess
                //Add the next tester digit after the found digits
                //Increment pos so we test the next 0-9 digit
        //We have all 4 digits now - reset the counters
                //Set pos to 0 – it's now the position to test in sGuess
                //Set indx to 0 - it's the index to use from digits
```

```
// Phase 2 - Determine the positions of each digit
        //See if we have a match for current (rr == 1 & rw = 0)
                //Add the current digit to the placed location
                //Increment found
                //Increment indx - Move to next index of digits
                //Pos = 0 - Restart our search at the beginning
        //We found all 4 let's skip the rest
                //Copy placed into sGuess
                //Return sGuess
        //While pos < 4 & placed position already found
                //Increment pos to move to the next missing position
        //All positions tested – pos == 4
                //Set pos back to 0
                //Increment indx so we check the next digit
        //Start with '0'
        //While the wrong digit exists in the digits string
                //Increment wrong to check the next possible wrong digit
        //Now check each digit's position
                //If position isn't being checked, plug in wrong digit
                //Else copy the digit into the position to check
        //If pos is < 4
                //Increment pos so we move to the next position
//Save the guess to the previous guesses array and increment the guess count
//Return the guess
```