Mastermind

By Chris Herre
Course: CSC 7
ID: 42645

Introduction:

In the game of Mastermind, a random 4-digit solution set is generated. The objective of the game is for the player to correctly guess the solution set given 8 attempts to do so. If the player can guess the solution within 8 attempts, they win, if the player fails after 8 attempts, they lose. The possible range for each index value of the solution set is only 0-5. Inputs greater than 5 or less than 0 will result in the game terminating early, giving an error message.

After each attempt, feedback will be given to the player in the form of black and white pegs. In this version, the pegs are not displayed visually, but rather the counter variables for black and white pegs are shown. A black peg means that one of the guesses the player has made is both the correct value (number) and is also in the correct position (index). A white peg means that the value is correct, but the position is wrong. This feedback helps the player know how to change their combination for the next attempt.

When the solution is correctly guessed, or all 8 attempts are over and the player has lost, the solution set will finally be revealed to the player. For debugging purposes, the solution set is revealed early, once at the start of the game.

Summary:

Project size: 193 lines of code
Number of variables: About 18 distinct variables
Number of methods: 9 methods

This project uses concepts such as pointers and dynamic arrays, constant variables, pass by value, pass by reference, arrays as function parameters, and more.

The most difficult part of this game was figuring out how the logic worked for the black and white pegs. It took many tests to get things working the way they should. The rest of the project was very simple and used mainly concepts from CSC-5 such as loops, arrays, ternary statements, conditional logic, and so forth. I spent about 4 days getting everything to work the way it is now.

Flowchart:

Too big to fit on page, attached at PDF file.

Psuedocode:

```
srand(static_cast<unsigned int>(time(0)));
const short ATTEMPTS = 3;//8;
// the solution set will consist of 4 pegs
const short SOLUTION_PEGS = 4;
// we have 6 different possible peg colors, or values
const short PEG_TYPES = 6;
const string WIN = "You Win!!!";
const string LOST = "You Lost!";
// the solution set of 4 values
short *solution = new short[SOLUTION_PEGS];
// the current guess set of 4 values
// guessing the solution set
short *guesses = new short[SOLUTION_PEGS];
short *score = new short[SOLUTION_PEGS];
bool gameOver = false;

// generate a random 4-digit solution set
// set the guesses array to all -1's
// display a rules message
// display the debug solution, AKA the answer
// loop 8 times, or until the game is over
        // check if the elements in the guesses set matches the solution set's elements
                // gameOver = true;
        // display the solution
        // check if gameOver == true
                // cleanUp dynamic arrays
                // cout win message
                // exit the program
        // short black = 0;
        // short white = 0;
        // loop 4 times
                // check each peg to see if its in the right position and has the right value
                        // black++;
                // else if correct number, but wrong position
                        // white++;
        // end of loop
        // cout black counter
        // cout white counter
        // reset the elements of the guesses array to -1
        // cout << "ATTEMPT #" << i + 1 << endl;
        // loop 4 times, guesses loop for 4 indices
                // cout << "Enter a number 0-" << types - 1 << " for index " << j
                        << " of the guess set: ";
                // cin >> guesses[j];
                // check for invalid input
                        // display invalid input error message
                        // clean up dynamic arrays
```

```
                        // exit the program
                    // print the guessed set of 4 number, one at a time
                    // cout << endl;
            // end of loop
// end of loop
// gameOver = true;
// display solution
// check if the elements in the guesses set matches the solution set's elements
            // cout << win << endl;
// else
            // cout << lost << endl;
// cleanUp dynamic arrays
```

Source Code:

```
/*
 * File:   main.cpp
 * Author: Chris Herre
 *
 * Created on April 5, 2017, 10:27 PM
 */

#include <cstdlib>
#include <iostream>
#include <string>
#include <ctime>

using namespace std;

void initGuesses(short*, short);
void initSolution(short*, short, short);
void displaySolution(short*, short, bool);
void doFeedback(short*, short*, short&, short&, short);
void cleanUp(short*, short*, short*);
void doAttempt(short*, short*, short*, short, short, short);
void doPreattempt(short*, short*, short*, bool&, short, string);
void start(short*, short*, short*, string, string, bool&, short, short, short);

/*
 * Known bugs
 *
 */

int main(int argc, char** argv) {
    srand(static_cast<unsigned int>(time(0)));
    const short ATTEMPTS = 3;//8;
    // the solution set will consist of 4 pegs
    const short SOLUTION_PEGS = 4;
    // we have 6 different possible peg colors, or values
```

```
   const short PEG_TYPES = 6;
   const string WIN = "You Win!!!";
   const string LOST = "You Lost!";
   // the solution set of 4 values
   short *solution = new short[SOLUTION_PEGS];
   // the current guess set of 4 values
   // guessing the solution set
   short *guesses = new short[SOLUTION_PEGS];
   short *score = new short[SOLUTION_PEGS];
   bool gameOver = false;
   initSolution(solution, SOLUTION_PEGS, PEG_TYPES);
   // set the guesses display to all *s
   initGuesses(guesses, SOLUTION_PEGS);
   start(solution, guesses, score, WIN, LOST, gameOver, ATTEMPTS,
         PEG_TYPES, SOLUTION_PEGS);
   return 0;
}

void start(short* solution, short* guesses, short* score, string win,
      string lost, bool& gameOver, short attempts, short types, short pegs)
{
   cout << "A random solution set has been chosen. You will\nhave "
         << attempts << " attempts to guess all of the elements\n"
         "of the solution set to win the game. This version\nuses numbers "
         "0-" << types - 1 << " instead of colored pegs.\n" << endl;
   // display the answer to save Dr. Lehr time
   cout << "Debug ";
   displaySolution(solution, pegs, true);
   // attempts
   for (short i = 0; !gameOver && i < attempts; i++)
   {
      doPreattempt(solution, guesses, score, gameOver, pegs, win);
      cout << "ATTEMPT #" << i + 1 << endl;
      // current guesses
      for (short j = 0; j < pegs; j++)
      {
         doAttempt(solution, guesses, score, j, pegs, types);
      }
   }
   gameOver = true;
   displaySolution(solution, pegs, gameOver);
   if (guesses[0] == solution[0] && guesses[1] == solution[1]
         && guesses[2] == solution[2] && guesses[3] == solution[3])
   {
      cout << win << endl;
   }
   else
   {
      cout << lost << endl;
```

```
    }
    cleanUp(solution, guesses, score);
}

void cleanUp(short* solution, short* guesses, short* score)
{
    delete[] solution;
    delete[] guesses;
    delete[] score;
}

void initGuesses(short* guesses, short pegs)
{
    for (short k = 0; k < pegs; k++)
    {
        guesses[k] = -1;
    }
}

// randomly generates a 4 value solution
void initSolution(short* solution, short pegs, short pegTypes)
{
    for (short i = 0; i < pegs; i++)
    {
        solution[i] = static_cast<short>(rand() % pegTypes);
    }
}

// prints the 4 value solution
void displaySolution(short* solution, short pegs, bool end)
{
    cout << "Solution:\n";
    for (short i = 0; i < pegs; i++)
    {
        cout << (end ? static_cast<char>(solution[i] + 48) : '*') << "  ";
    }
    cout << endl;
}

// these are the black and white dots, for now just counter variables
void doFeedback(short* solution, short* guesses, short& black, short& white,
        short pegs)
{
     for (short l = 0; l < pegs; l++)
    {
        // correct number and is in the correct position
        if ((l == 0 && guesses[l] == solution[0])
                || (l == 1 && guesses[l] == solution[1])
                || (l == 2 && guesses[l] == solution[2])
```

```
            || (l == 3 && guesses[l] == solution[3]))
        {
            black++;
        }
        // correct number, but wrong position
        else if (solution[l] == guesses[0] || solution[l] == guesses[1]
            || solution[l] == guesses[2] || solution[l] == guesses[3])
        {
            white++;
        }
    }
    cout << "black:\t" << black << endl;
    cout << "white:\t" << white << endl;
}

// process the logic of a single attempt
void doAttempt(short* solution, short* guesses, short* score, short j,
        short pegs, short types)
{
    cout << "Enter a number 0-" << types - 1 << " for index " << j
        << " of the guess set: ";
    cin >> guesses[j];
    if (guesses[j] > types - 1 || guesses[j] < 0)
    {
        cout << "Critical error! Input cannot be greater than "
            << types - 1 << ", or less than 0!" << endl;
        cleanUp(solution, guesses, score);
        exit(0);
    }
    for (short k = 0; k < pegs; k++)
    {
        cout << (guesses[k] < 0
            ? '*' : static_cast<char>(guesses[k] + 48)) << " ";
    }
    cout << endl;
}

void doPreattempt(short* solution, short* guesses, short* score,
        bool& gameOver, short pegs, string win)
{
    // if the guesses set and the solution set
    // have equal elements, game over
    if (guesses[0] == solution[0] && guesses[1] == solution[1]
        && guesses[2] == solution[2] && guesses[3] == solution[3])
    {
        gameOver = true;
    }
    displaySolution(solution, pegs, gameOver);
    // handle game over and exit
```

```
    if (gameOver)
    {
        cleanUp(solution, guesses, score);
        cout << win << endl;
        exit(0);
    }
    short black = 0;
    short white = 0;
    doFeedback(solution, guesses, black, white, pegs);
    // reset the guesses display to all *s
    initGuesses(guesses, pegs);
}
```