CSE2003- Data Structures and Algorithms

Slot: B1

MASTERMIND GAME

Group Members:

S. No	Name	Reg. No
1	SAURABH AGRAWAL	19BCE0305
2	TANISHQ SHAH	19BCE2067
3	HARSHIT SHRIVASTAVA	19BCE0289
4	SANJANA SINGH	19BCE2109
5	VANSH AGGARWAL	19BCE0987

PROJECT REPORT

FACULTY

Prof. Gayathri Prakasam

VIT®
Vellore Institute of Technology
(Deemed to be University under section 3 of UGC Act, 1956)

School of Computer Science & Engineering VIT University

WINTER SEMESTER 2019-2020

INDEX:

ABSTRACT

The Mastermind Game is a multi-use arcade electronic game designed during a study week organized by the Swiss Youth in Science Foundation. We the students added some more various data structures in this prototype. The program can be implemented in Python 3 running on a MicroPython compatible microcontroller ESP32 by Espressif while the hardware is connected with a PCB (Printed Circuit Board) or a VeroBoard for prototyping. Many technical challenges were encountered such as limited dynamic memory and hardware failures but the final prototype was complete with all of the previously mentioned features.

AIM

The aim of the project is to design an electronic version of the classic board game Mastermind, using an algorithm that plays the game on its own.

OBJECTIVE

In this game, there are six color pegs to choose from. The object of the game is to get the exact positions of the colors in the computer sequence in as few guesses as possible. After each guess, the computer gives you a score of exact and partial matches.

INTRODUCTION

The goal for our project is to prepare a working prototype for an electronic version of a classic board game called Mastermind. The secondary goal is to implement an algorithm that plays the game on its own, possibly better than how human would. Lastly a third objective is to extend the platform be able to run other games such as Tetris or Snake.

LITERATURE REVIEW

Training in action video games can increase the speed of perceptual processing. However, it is unknown whether video-game training can lead to broad-based changes in higher-level competencies such as cognitive flexibility, a core and neurally distributed component of cognition. To determine whether video gaming can enhance cognitive flexibility and, if so, why these changes occur, the current study compares two versions of a real-time strategy (RTS) game. Using a meta-analytic Bayes factor approach, we found that the gaming condition that emphasized maintenance and rapid switching between multiple information and action sources led to a large increase in cognitive flexibility as measured by a wide array of non-video gaming tasks. Theoretically, the results suggest that the distributed brain networks supporting cognitive flexibility can be tuned by engrossing video game experience that stresses maintenance and rapid manipulation of multiple information sources. Practically, these results suggest avenues for increasing cognitive function. [1]

In this paper we address the issue of compressing and indexing data. We devise a data structure whose space occupancy is a function of the entropy of the underlying data set. We call the data structure opportunistic since its space occupancy is decreased when the input is compressible and this space reduction is achieved at no significant slowdown in the query performance. More precisely, its space occupancy is optimal in an information-content sense because a text T[1, u] is stored using O(Hk(T))+o(1) bits per input symbol in the worst case, where Hk (T) is the kth order empirical entropy of T (the bound holds for any fixed k). Given an arbitrary string P[1,p], the opportunistic data structure allows to search for the occ occurrences of P in T in O(p+occ logɛ u) time (for any fixed ɛ>0). If data are uncompressible we achieve the best space bound currently known; on compressible data our solution improves the succinct suffix array of [12] and the classical suffix tree and suffix array data structures either in space or in query time or both. We also study our opportunistic data structure in a dynamic setting and devise a variant achieving effective search and update time bounds. Finally, we show how to plug our opportunistic data structure into the Glimpse tool. The result is an indexing tool which achieves sublinear space and sublinear query time complexity.[2]

We investigate the use of standard statistical models for quantal choice in a game theoretic setting. Players choose strategies based on relative expected utility and assume other players do so as well. We define a quantal response equilibrium (ORE) as a fixed point of this process and establish existence. For a logit specification of the error structure, we show that as the error goes to zero, QRE approaches a subset of Nash equilibria and also implies a unique selection from the set of Nash equilibria in generic games. We fit the model to a variety of experimental data sets by using maximum likelihood estimation [3]

In the Python world, NumPy arrays are the standard representation for numerical data and enable efficient implementation of numerical computations in a high-level language. As this effort shows, NumPy performance can be improved through three techniques: vectorizing calculations, avoiding copying data in memory, and minimizing operation counts. [4]

An approach for capturing and modeling individual entertainment ("fun") preferences is applied to users of the innovative Playware playground, an interactive physical playground inspired by computer games, in this study. The goal is to construct, using representative statistics computed from children's physiological signals, an estimator of the degree to which games provided by the playground engage the players. For this purpose, children's heart rate (HR) signals, and their expressed preferences of how much "fun" particular game variants are, are obtained from experiments using games implemented on the Playware playground. A comprehensive statistical analysis shows that children's reported entertainment preferences correlate well with specific features of the HR signal. Neuro-evolution techniques combined with feature set selection methods permit the construction of user models that predict reported entertainment preferences given HR features. These models are expressed as artificial neural networks and are demonstrated and evaluated on two Playware games and two control tasks requiring physical activity. The best network is able to correctly match expressed preferences in 64% of cases on previously unseen data (p-value 6 • 10-5). The generality of the methodology, its limitations, its usability as a real-time feedback mechanism for entertainment augmentation and as a validation tool are discussed. [5]

IMPLEMENTATION

First, we will declare an array and the program will randomly allocate a sequence of colors in that array. Now we store the number of chances a user will get in a variable and run a loop that many times. In each iteration user will enter a sequence of color, trying to match with the original sequence created by the program. Each time the user enters a sequence, program matches it with the original sequence and gives the output according to the rules. If any color is right in the sequence but the position is different then the computer gives the output as “white”. And, when the color and position, both are in the correct place, program gives the output as “black”.

Lastly the program will declare the user winner if he gets four “black” in the output or loser if he is not able to. After that user gets an option which is implemented through switch case that if he wishes to continue to another game or wants to end it.

The players should be able to enter four colors as their guess. When they enter their guess, then your program should display their guess and next to the guess it should display the score.

After the player completes playing the game once (after either they win or they had twelve guesses), your program should ask the user if they would like to continue if they do then your program should generate a new code.

MODULE DESCRIPTION

WORKING:

For each of the pegs in the guess that is the correct color and the correct position, the computer will give you one small black peg to the right of that move. For each of the pegs in your guess that is a correct color in an incorrect position, the computer will give you one small white peg to the right of that move. Together, there will be no more than four small black and white pegs for each move. If none of the pegs in your guess is of a correct color, you will see no small pegs to the right of that move. If you score four small black pegs on a guess, you have guessed the secret sequence.

RULES:

1) The sequence can contain pegs of colors: red, yellow, green, blue, purple, orange.

2) A color can be used any number of times in the sequence.

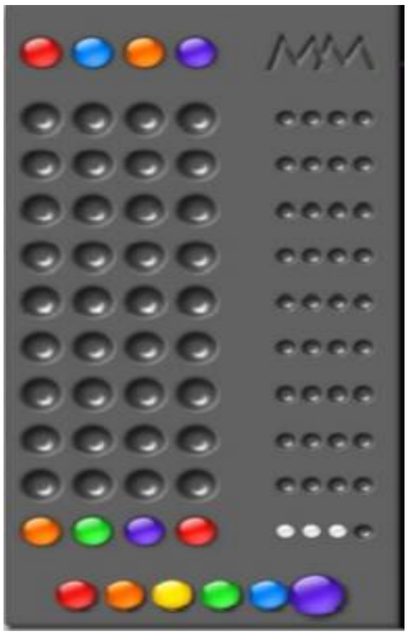
3) All four pegs of the secret sequence will contain a color. (no blanks/empties are allowed.)

4) Each guess must consist of 4 peg colors. (no blanks.)

5) The player has ten guesses to find the secret sequence.

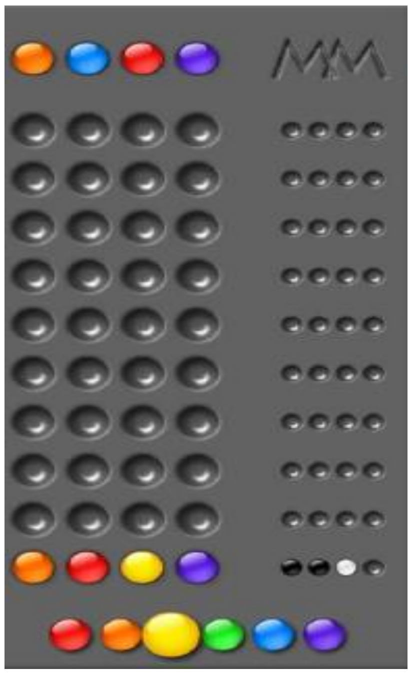
DIFFERENT CASES IN THE GAME

CASE: I



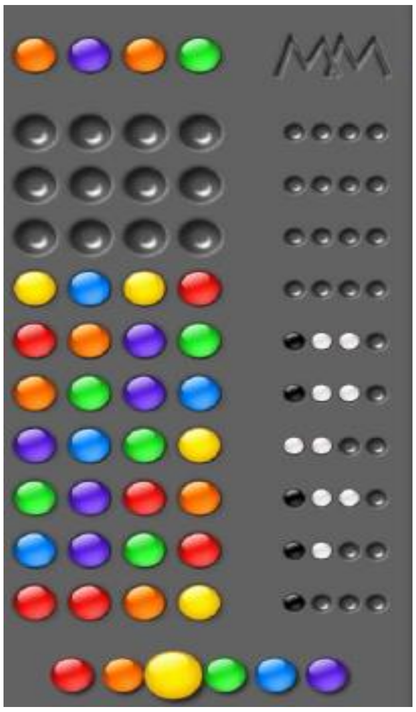
In this case three of the colors (red, orange and purple) in our guess is correct but the position is not correct, as we can see from the correct sequence, hence three white pegs appeared.

CASE: II



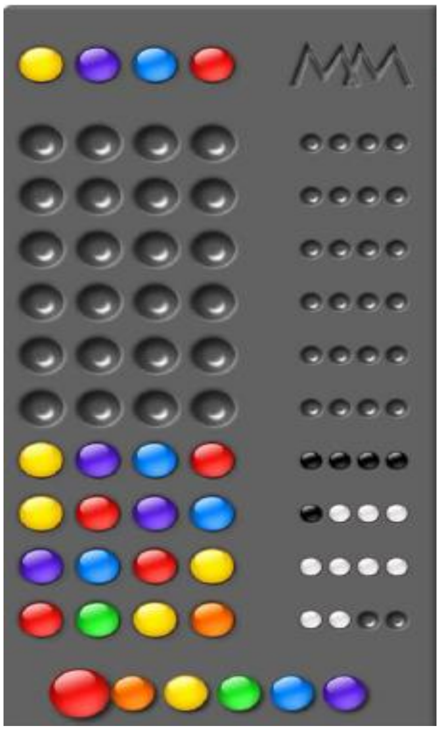
In this case two of the colors (orange and purple) in our guess is correct and their position is also correct, as we can see from the correct sequence, hence two black pegs appeared. Other than this one more color (red) in our guess is correct but its position is not correct, hence one white peg appeared.

CASE: III



In this case none of the colors (yellow, blue and red) of our guess matches with the actual sequence as we can see and hence no pegs appeared.

CASE: IV

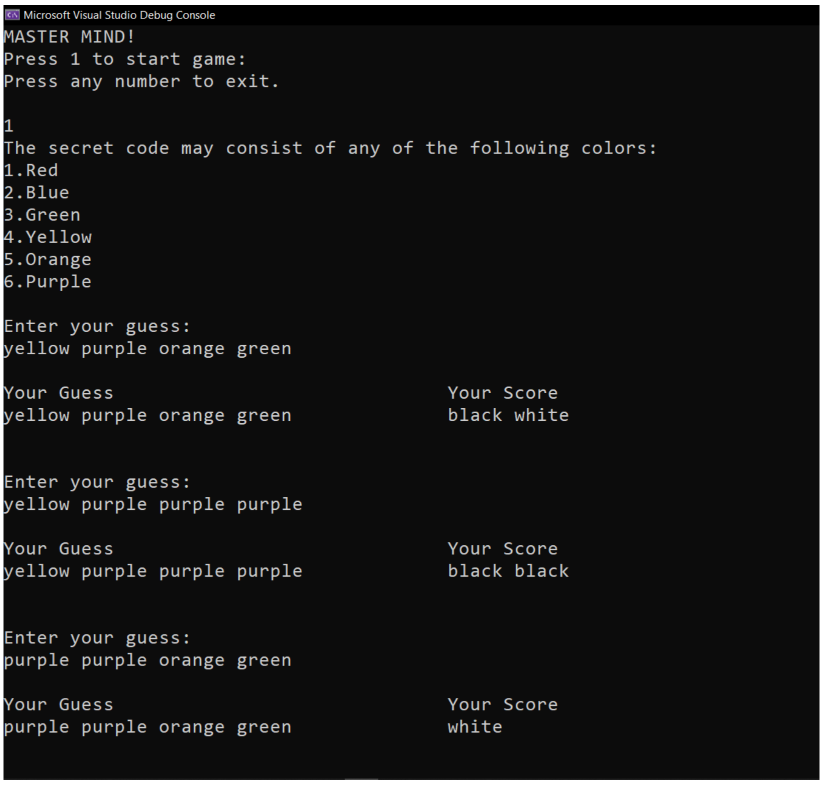


In this case all the colors (yellow, purple, blue, red) are matching with the actual sequence and there position is also correct and hence four black pegs appeared.

RESULTS AND DISCUSSION

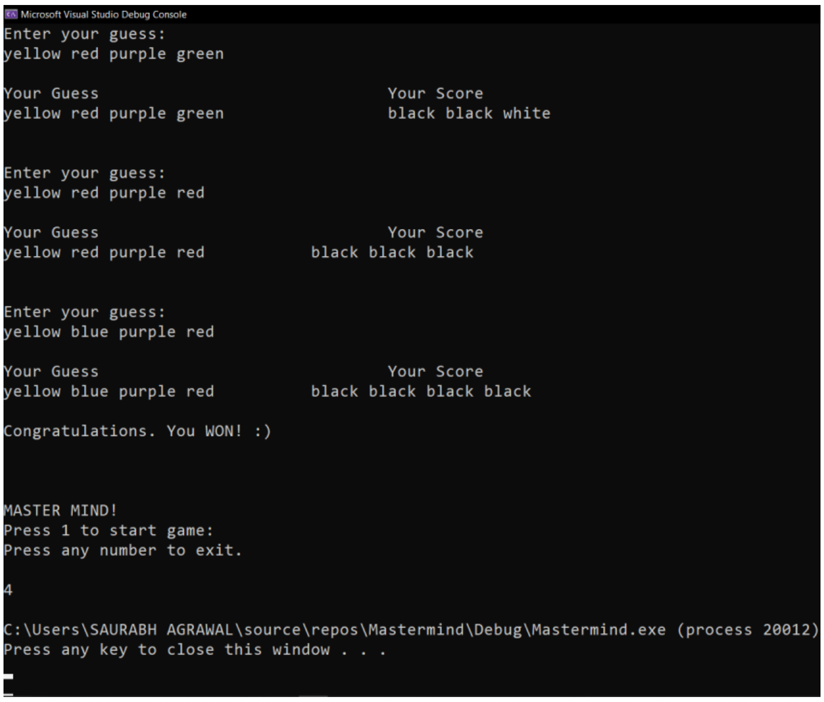
```
#ifdef _MSC_VER
#define _CRT_SECURE_NO_WARNINGS
#endif
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
#include<time.h>


void makeCode(char secretCode[4][10])
{
    int i, randColor;
    for (i = 0; i < 4; i++)
    {
        randColor = 1 + rand() % 6; //creates a number
        switch (randColor) //converts number created to a string
        {
        case 1: strcpy(secretCode[i], "red"); break;
        case 2: strcpy(secretCode[i], "yellow"); break;
        case 3: strcpy(secretCode[i], "green"); break;
        case 4: strcpy(secretCode[i], "blue"); break;
        case 5: strcpy(secretCode[i], "orange"); break;
        case 6: strcpy(secretCode[i], "purple"); break;
        }
    }

}
void guess(char guessCode[4][10])
{
    int i;
    printf("\nEnter your guess:\n");
    for (i = 0; i < 4; i++)
        scanf("%s", guessCode[i]);
}
void codeCheck(char secretCode[4][10], char guessCode[4][10], int *blackPeg, int *whitePeg)
{
    int i, j, checkSecret[4] = { 1,1,1,1 }, checkGuess[4] = { 1,1,1,1 };
    *blackPeg = *whitePeg = 0;
    for (i = 0; i < 4; i++) //if secret’s and guess's position and color are same, blackpeg increases
                                  and marks "check"
    {

        if (strcmp(guessCode[i], secretCode[i]) == 0)
        {
            ++*blackPeg;
            checkSecret[i] = checkGuess[i] = 0;
        }
    }
    for (i = 0; i < 4; i++)
    {
        for (j = 0; j < 4; j++)
        {
            if (strcmp(secretCode[i], guessCode[j]) == 0 && checkGuess[i]
                && checkSecret[j] && i != j) // determines crushes and eliminates extra whitePegs
            {
                ++*whitePeg;
                checkSecret[j] = checkGuess[i] = 0;
            }
        }
    }
}
void displayGuess(char guessCode[4][10], int blackPeg, int whitePeg)
{
    int i;
    printf("\nYour Guess\t\t\t\tYour Score\n");
    for (i = 0; i < 4; i++)
        printf("%s ", guessCode[i]);
    printf("\t\t");
    for (i = 0; i < blackPeg; i++)
        printf("black ");
    for (i = 0; i < whitePeg; i++)
        printf("white ");
    printf("\n\n");
}
int main()
{
    srand(time(NULL));
    int i, option = 1, blackPeg, whitePeg, wrongGuess;
    char secretCode[4][10], guessCode[4][10];
    while (1)
    {
        printf("MASTER MIND! \nPress 1 to start game: \nPress any number to exit.\n\n");
        scanf("%d", &option);
        if (option == 1)
        {
            makeCode(secretCode);
            for (wrongGuess = 1; wrongGuess <= 10; wrongGuess++) //gives 10 rights to guess
            {
                if (wrongGuess == 1)
                    printf("The secret code may consist of any of the following
                            colors:\n1.Red\n2.Blue\n3.Green\n4.Yellow\n5.Orange\n6.Purple\n");
                guess(guessCode);
                codeCheck(secretCode, guessCode, &blackPeg, &whitePeg);
                displayGuess(guessCode, blackPeg, whitePeg);
                if (blackPeg == 4) //if player guess correct all, than the game finishes
                {
                    printf("Congratulations. You WON! :)\n\n\n\n"); break;
                }
            }
            if (wrongGuess == 11) //if player cannot guess correct colors in 10 rounds, he will lose
                printf("\nYou LOST! :(\nSecret Code: %s %s %s %s\n\n\n\n\n",
                           secretCode[0], secretCode[1], secretCode[2], secretCode[3]);
        }
        else
            exit(1);
    }
}
```

Case 1 (Winning Case):



Microsoft Visual Studio Debug Console

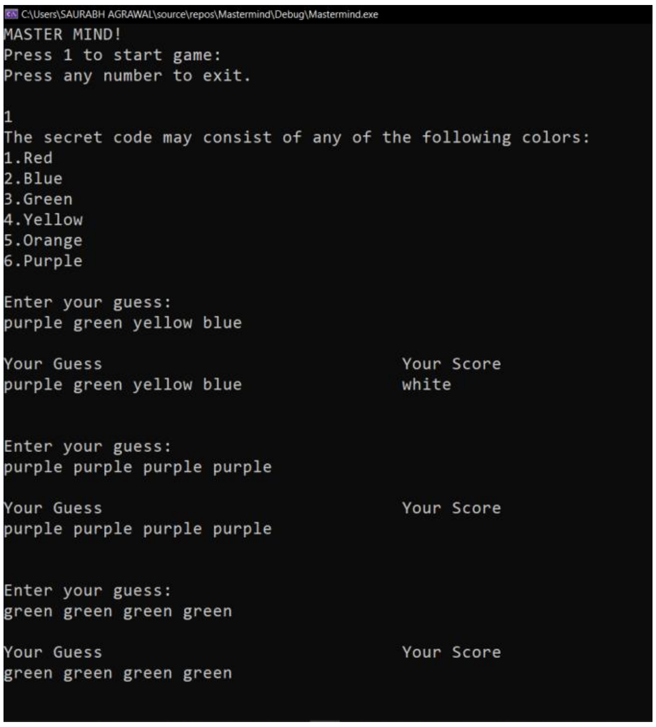
```
MASTER MIND!
Press 1 to start game:
Press any number to exit.

1
The secret code may consist of any of the following colors:
1.Red
2.Blue
3.Green
4.Yellow
5.Orange
6.Purple

Enter your guess:
yellow purple orange green

Your Guess                              Your Score
yellow purple orange green              black white


Enter your guess:
yellow purple purple purple

Your Guess                              Your Score
yellow purple purple purple             black black


Enter your guess:
purple purple orange green

Your Guess                              Your Score
purple purple orange green              white
```



Microsoft Visual Studio Debug Console

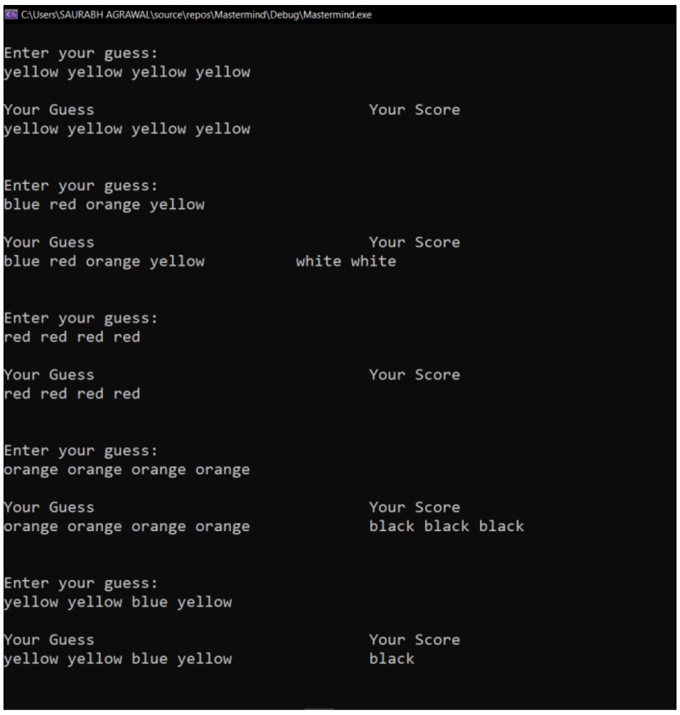
```
Enter your guess:
yellow red purple green

Your Guess                              Your Score
yellow red purple green                 black black white


Enter your guess:
yellow red purple red

Your Guess                              Your Score
yellow red purple red           black black black


Enter your guess:
yellow blue purple red

Your Guess                              Your Score
yellow blue purple red          black black black black

Congratulations. You WON! :)


MASTER MIND!
Press 1 to start game:
Press any number to exit.

4

C:\Users\SAURABH AGRAWAL\source\repos\Mastermind\Debug\Mastermind.exe (process 20012)
Press any key to close this window . . .
```

Case 2(Losing Case):



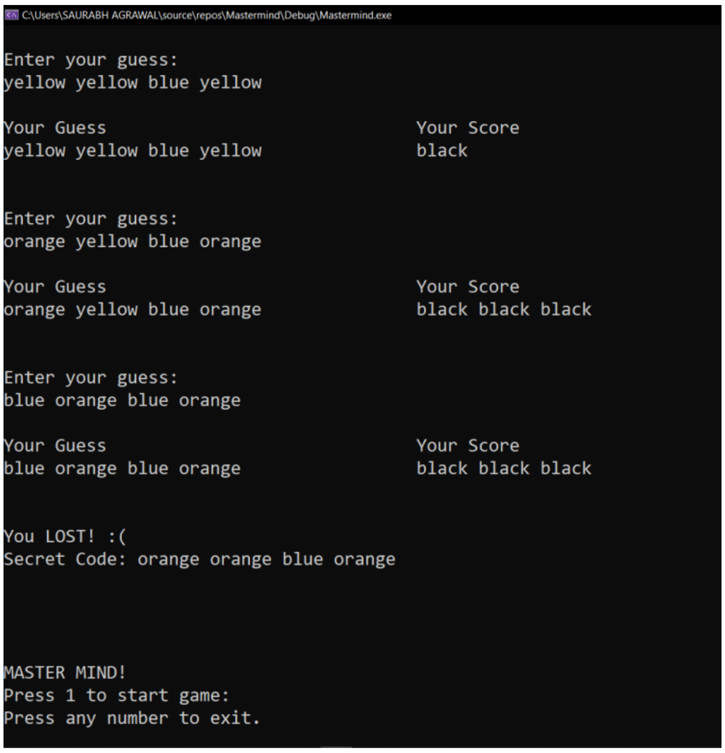
C:\Users\SAURABH AGRAWAL\source\repos\Mastermind\Debug\Mastermind.exe

```
MASTER MIND!
Press 1 to start game:
Press any number to exit.

1
The secret code may consist of any of the following colors:
1.Red
2.Blue
3.Green
4.Yellow
5.Orange
6.Purple

Enter your guess:
purple green yellow blue

Your Guess                              Your Score
purple green yellow blue                white


Enter your guess:
purple purple purple purple

Your Guess                              Your Score
purple purple purple purple


Enter your guess:
green green green green

Your Guess                              Your Score
green green green green
```



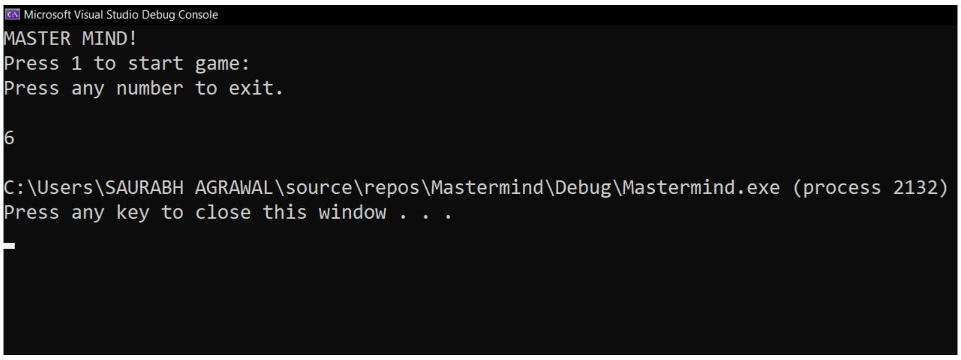
C:\Users\SAURABH AGRAWAL\source\repos\Mastermind\Debug\Mastermind.exe

```
Enter your guess:
yellow yellow yellow yellow

Your Guess                              Your Score
yellow yellow yellow yellow


Enter your guess:
blue red orange yellow

Your Guess                              Your Score
blue red orange yellow          white white


Enter your guess:
red red red red

Your Guess                              Your Score
red red red red


Enter your guess:
orange orange orange orange

Your Guess                              Your Score
orange orange orange orange             black black black


Enter your guess:
yellow yellow blue yellow

Your Guess                              Your Score
yellow yellow blue yellow               black
```



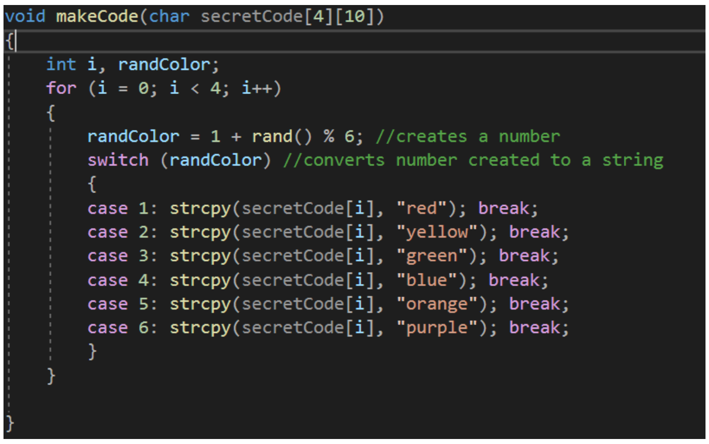
```
C:\Users\SAURABH AGRAWAL\source\repos\Mastermind\Debug\Mastermind.exe

Enter your guess:
yellow yellow blue yellow

Your Guess                                Your Score
yellow yellow blue yellow                 black


Enter your guess:
orange yellow blue orange

Your Guess                                Your Score
orange yellow blue orange                 black black black


Enter your guess:
blue orange blue orange

Your Guess                                Your Score
blue orange blue orange                   black black black


You LOST! :(
Secret Code: orange orange blue orange



MASTER MIND!
Press 1 to start game:
Press any number to exit.
```

Case 3(Code exits):



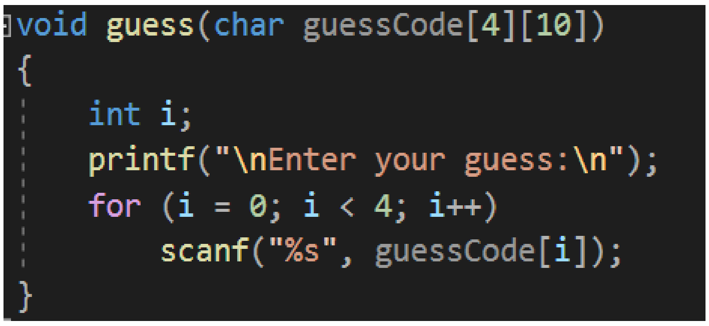
```
Microsoft Visual Studio Debug Console
MASTER MIND!
Press 1 to start game:
Press any number to exit.

6

C:\Users\SAURABH AGRAWAL\source\repos\Mastermind\Debug\Mastermind.exe (process 2132)
Press any key to close this window . . .
```

TIME COMPLEXITIES



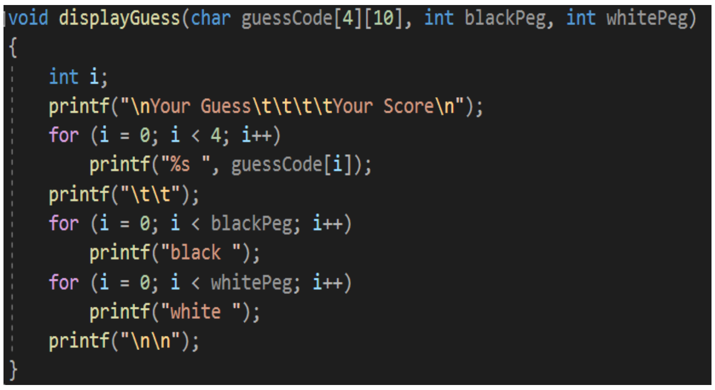
```
void makeCode(char secretCode[4][10])
{
    int i, randColor;
    for (i = 0; i < 4; i++)
    {
        randColor = 1 + rand() % 6; //creates a number
        switch (randColor) //converts number created to a string
        {
        case 1: strcpy(secretCode[i], "red"); break;
        case 2: strcpy(secretCode[i], "yellow"); break;
        case 3: strcpy(secretCode[i], "green"); break;
        case 4: strcpy(secretCode[i], "blue"); break;
        case 5: strcpy(secretCode[i], "orange"); break;
        case 6: strcpy(secretCode[i], "purple"); break;
        }
    }

}
```

The adjacent piece of code is responsible for generating the random color sequence. The time complexity of the function depends on the for loop used in the function and therefore it is **O(4)** where 4 is the number of iterations of the loop.



```
void guess(char guessCode[4][10])
{
    int i;
    printf("\nEnter your guess:\n");
    for (i = 0; i < 4; i++)
        scanf("%s", guessCode[i]);
}
```

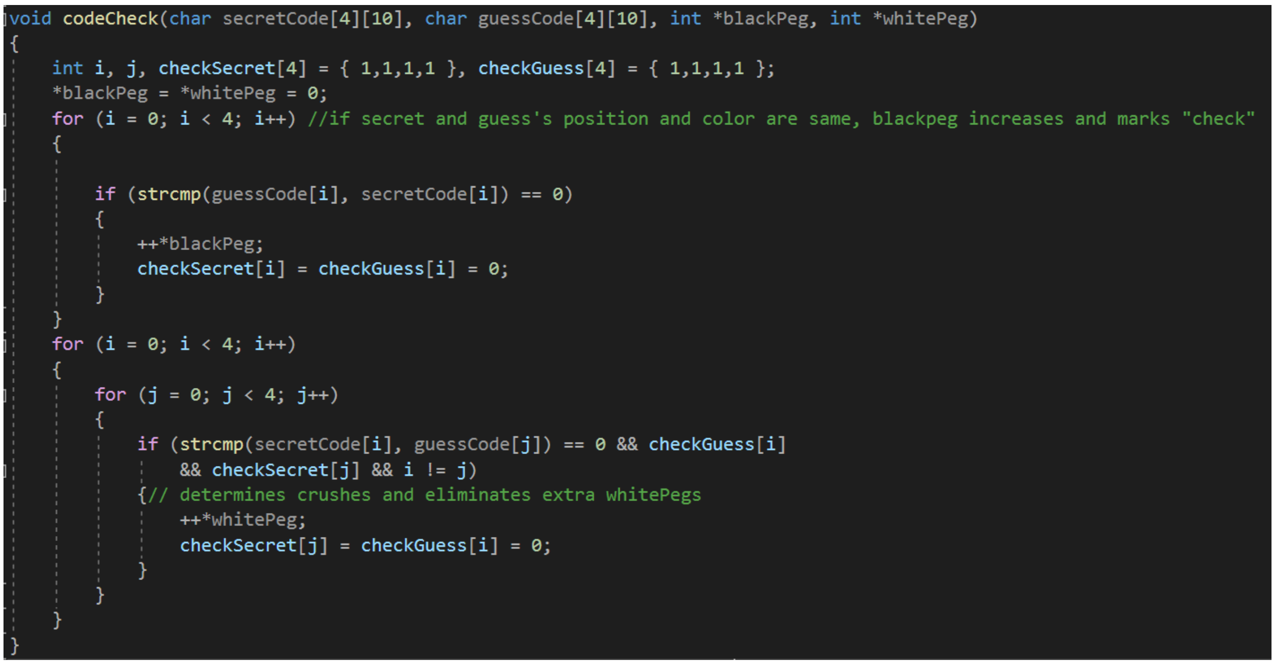
The adjacent piece of code is responsible for taking the user's guess. The time complexity of the function depends on the for loop used in the function and therefore it is **O(4)** where 4 is the number of iterations of the loop.



```
void displayGuess(char guessCode[4][10], int blackPeg, int whitePeg)
{
    int i;
    printf("\nYour Guess\t\t\tYour Score\n");
    for (i = 0; i < 4; i++)
        printf("%s ", guessCode[i]);
    printf("\t\t");
    for (i = 0; i < blackPeg; i++)
        printf("black ");
    for (i = 0; i < whitePeg; i++)
        printf("white ");
    printf("\n\n");
}
```

The adjacent piece of code is responsible for displaying the score/hint(s) that helps the user to know the accuracy of their guess. The time complexity of the function depends on the three for loops used in the function.

The time complexity of the first loop will be O(4). Since the number of iterations of the two other loops depend on user's guess, we can further classify the time complexities of the loop as best or worst time complexities. Since the highest number of iterations that the loops can have is 4, therefore the worst time complexity of the function will be **3*O(4)** (i.e. O(4)+O(4)+O(4)) and the best time complexity will be **O(4)** since the least number of iterations that the loops can have is 0.



```
void codeCheck(char secretCode[4][10], char guessCode[4][10], int *blackPeg, int *whitePeg)
{
    int i, j, checkSecret[4] = { 1,1,1,1 }, checkGuess[4] = { 1,1,1,1 };
    *blackPeg = *whitePeg = 0;
    for (i = 0; i < 4; i++) //if secret and guess's position and color are same, blackpeg increases and marks "check"
    {

        if (strcmp(guessCode[i], secretCode[i]) == 0)
        {
            ++*blackPeg;
            checkSecret[i] = checkGuess[i] = 0;
        }
    }
    for (i = 0; i < 4; i++)
    {
        for (j = 0; j < 4; j++)
        {
            if (strcmp(secretCode[i], guessCode[j]) == 0 && checkGuess[i]
                && checkSecret[j] && i != j)
            {// determines crushes and eliminates extra whitePegs
                ++*whitePeg;
                checkSecret[j] = checkGuess[i] = 0;
            }
        }
    }
}
```

The function above is responsible for matching the random color sequence generated by the program with the guessed colors entered by the user. The time complexity of the first for loop used for the function is **O(4)** and that of the second(nested) for loop is **O(4*4)** (since the time complexity of a nested loop is O(m*n) where m and n are the number of iterations of each loop). Therefore the time complexity of the function is **O(4) + O(4*4)**.

REFERENCES

[1] Glass, B. D., Maddox, W. T., & Love, B. C. (2013). Real-Time Strategy Game Training: Emergence of a Cognitive Flexibility Trait. *PLoS ONE*. https://doi.org/10.1371/journal.pone.0070350

[2] Ferragina, P., & Manzini, G. (2000). Opportunistic data structures with applications. In *Annual Symposium on Foundations of Computer Science - Proceedings*. https://doi.org/10.1109/sfcs.2000.892127

[3] McKelvey, R. D., & Palfrey, T. R. (1995). Quantal response equilibria for normal form games. *Games and Economic Behavior*. https://doi.org/10.1006/game.1995.1023

[4] Van Der Walt, S., Colbert, S. C., & Varoquaux, G. (2011). The NumPy array: A structure for efficient numerical computation. *Computing in Science and Engineering*. https://doi.org/10.1109/MCSE.2011.37

[5] Yannakakis, G. N., Hallam, J., & Lund, H. H. (2008). Entertainment capture through heart rate activity in physical interactive playgrounds. *User Modeling and User-Adapted Interaction*. https://doi.org/10.1007/s11257-007-9036-7

[6] Goodrich, M.T., Tamassia, R. and Goldwasser, M.H., 2014. *Data structures and algorithms in Java*. John Wiley & Sons.

[7] Preiss, B.R., 2008. *Data structures and algorithms with object-oriented design patterns in C*++. John Wiley & Sons.

[8] Sahni, S., 2005. *Data structures, algorithms, and applications in Java*. Universities Press.

[9] Drozdek, A., 2012. *Data Structures and algorithms in C*++. Cengage Learning.

[10] Samet, H. and Webber, R.E., 1988. Hierarchical data structures and algorithms for computer graphics. I. Fundamentals. *IEEE Computer Graphics and applications*, *8*(3), pp.48-68.

[11] Galil, Z. and Italiano, G.F., 1991. Data structures and algorithms for disjoint set union problems. *ACM Computing Surveys (CSUR)*, *23*(3), pp.319-344.

[12] Lafore, R., 2017. *Data structures and algorithms in Java*. Sams publishing.

[13] Yamaguchi, K., Kunii, T., Fujimura, K. and Toriya, H., 1984. Octree-related data structures and algorithms. *IEEE Computer Graphics and Applications*, (1), pp.53-59.

[14] Galil, Z. and Giancarlo, R., 1988. Data structures and algorithms for approximate string matching. *Journal of Complexity*, *4*(1), pp.33-72.

[15] Gomes, A., Voiculescu, I., Jorge, J., Wyvill, B. and Galbraith, C., 2009. *Implicit curves and surfaces: mathematics, data structures and algorithms*. Springer Science & Business Media.