PROGRAMMING FUNDAMENTALS I - THEORICAL PROJECT 2022/23

Daniel Barchino Rodríguez-Caro

Juan María Bravo López

Andrés García López

Marcos Hurtado Morcillos

Index

Names, surnames and theory group of the 4 members of the group

The 4 members of the group are:

- Daniel Barchino Rodríguez-Caro – 1º A Student
- Juan María Bravo López – 1º A Student
- Andrés García López – 1º A Student
- Marcos Hurtado Morcillos – 1º A Student

Statement of the chosen problem

The statement of our chosen problem is:

Matermind game:

Mastermind is a game for two players, which is played on a board with B chips white and N black, small, and C colors, somewhat larger. One of the players, the "master", chooses a number F of colored tiles (F=4 in the original game), and puts a secret code hidden from the other player. This one, taking colored chips from the same set, proposes a combination, which is answered by the "master" with black and/or white chips. The master places as many black tiles as the player has placed coloured tiles; and as many white tiles as the player has placed correctly coloured tiles, but in a different place. The game ends when the player gets the correct combination (i.e. a combination with four black pieces), or there are no more possible combinations (depending on the size, although there are usually 10 combinations).

It is necessary to implement a program that allows you to play the Mastermind game, considering that the computer plays the role of "master", and that the number of colored pieces of the combination is an F value that is read by the keyboard. To do this, the program will start by initializing the board, for which the whole values B, N and C will be read from a text file, followed by the identifiers of the C colors, each one on a line. It will then ask the user to indicate the value of F and then randomly define the combination of F colors that the player will have to guess. Then the game starts. After each move, the program has to show the values entered by the player, along with the amount of black and/or white chips corresponding to the move. When finished, the program must show the winning combination, as well as all the attempts made by the player, along with the answers made by the program to each of the attempts.

Analysis of the problem and justification of the representation of the chosen data, as well as of the modular decomposition carried out

We are going to develop the virtual version of Mastermind game, so we are going to need:

- A way to generate the Master's Combination (the problem tell us that it is going to be built randomly).
- Get the attempt of the User (the User’s Combination).
- Compare the User's Combination with the Master's Combination.
- Tell the User the result of his try.

In order to do that, we have used the next data:

- **winCheck:** Initially “false”. It tells if the user has won.
- **turns:** Initially “10”. It stores the amount of turns to play.
- **amountPieces:** It stores the amount of pieces of the Master's Combination selected by the User.
- **colorList:** Array that stores the available colors.
- **masterCombination:** Array that stores the Master's Combination.
- **userCombination:** Array that stores the User's Combination.
- **answer:** Array that stores the answer that will receive the user. It values can be:
  - B (Black) means you correctly guessed one colored piece in its position.
  - W (White) means you correctly guessed one colored piece, but in a different position than in my combination.
  - X means that you got one piece wrong.
- **userRecord:** Array that stores the combinations tried by the User per attempt.
- **answerRecord:** Array that stores the answer received by the User per attempt.

This data is going to be used on the following modules:

- **main:** As it name said, it is the main module of the program.
- **initialMessage:** This method will print the rules to follow during the game.
- **askAmountPieces:** It will ask the user the amount of pieces wished for the Master's Combination.
- **colorLister:** It will read and store the colors from a document.
- **getMasterCombination:** It will get the Master's Combination randomly.
- **getUserCombination:** It will ask the User for its Combination and store it.
- **getAnswer:** It will compare the Master's Combination and the User's Combination in order to give a response to the User.

- **orderAnswer:** This method will order the output received by the User in order to show first the B's, then the W's and finally the X's.
- **isColor**: Module used by the module "getAnswer" in order to know if a color is on the Master's Combination.
- **vectorPrinter**: Method used to print the answer.
- **winCondition:** It will check if the User's Combination is the same than the Master's Combination, so, if the User has won the game.
- **finalMessage**: If the User has won the game, this method is going to be executed, printing the wining message.
- **recordPrinter:** Method that prints the records.

Pseudocode of the algorithm that resolves it

Pseudocode of the main() algorithm:

```
BEGIN
    initialMessage()
    winCheck ← false
    turns ← 10
    amountPieces ← askAmountPieces()
    colorList[] ← colorLister()
    masterCombination[] ← getMasterCombination(amountPieces, colorList)
    userCombination[] ← ARRAY [amountPieces]
    answer[] ← ARRAY [amountPieces]
    userRecord[][] ← ARRAY [turns][amountPieces]
    answerRecord[][] ← ARRAY [turns][amountPieces]
    FOR i = 0 WHILE i < turns INCREASE i ← 1
        DISPLAY "TURN " + turn number
        userCombination ← getUserCombination(amountPieces, colorList)
        answer ← getAnswer(masterCombination, userCombination, colorList)
        DISPLAY "Answer: " + vectorPrinter(answer)
        FOR j = 0 WHILE j < turns INCREASE j ← 1
            userRecord[i][j] ← userCombination[j];
            answerRecord[i][j] ← answer[j];
        END FOR
        winCheck ← winCondition(winCheck, answer, colorList)
        IF winCheck = true
            turns ← i + 1
        END IF
    END FOR
    finalMessage(winCheck, turns, masterCombination, userRecord, answerRecord)
END
```

Pseudocode of the algorithm initialMessage():

```
BEGIN
    DISPLAY “Welcome to Mastermind!”
    DISPLAY “In this game, you have to guess the Master's (me) Combination, that is
    composed of as many colored pieces as you want.”
    DISPLAY “You will have a maximum of 10 tries. Each try, an answer will appear
    telling you how well you did.”
    DISPLAY “B (black) means you correctly guessed one colored piece in its position."
    DISPLAY "W (white) means you correctly guessed one colored piece, but in a
    different position than in my combination."
    DISPLAY “X means that you got one piece wrong.”
    DISPLAY “"Each answer will have as many of these hints as the number colored
    pieces you chose.”
    DISPLAY “"Good luck and have fun!”
END
```

Pseudocode of the algorithm askAmountPieces():

```
BEGIN
    DO
        DISPLAY “First, tell me the amount of pieces to guess (higher than 1):”
        amountPieces ← INPUT
    WHILE amountPieces ≤ 1
    RETURN amountPieces
END
```

Pseudocode of the algorithm colorLister():

```
BEGIN
    READ colors.txt
    FOR i = 0 WHILE i < LENGTH colorList[] INCREASE i ← 1
        colorList[i] ← LINE i FROM colors.txt
    END FOR
    RETURN  colorList[]
END
```

Pseudocode of the algorithm getMasterCombination():

```
BEGIN
    FOR i = 0 WHILE i < amountPieces INCREASE i ← 1
        colorRandom ← RANDOM number (0-8)
        colorPicked ←  colorList[colorRandom]
        masterCombination[i] ← colorPicked;
    END FOR
    RETURN masterCombination
END
```

Pseudocode of the algorithm getUserCombination():

```
BEGIN
    userCombination[] ← ARRAY [amountPieces]
    FOR i = 0 WHILE i < amountPieces INCREASE i ← 1
        DO
            DISPLAY “Tell me the color of the element to guess (type in CAPITAL
                LETTERS without quotation marks "B" (black), "W" (white), "R" (red),
                "P" (purple), "G" (green), "Y” (yellow), "O" (orange), "C" (cyan) ): ”
            userCombination[i] ← INPUT
        WHILE userCombination[i] ≠ element in colorList[]
    END FOR
    RETURN userCombination
END
```

Pseudocode of the algorithm getAnswer():

```
BEGIN
    answer[] ← ARRAY [LENGTH masterCombination]
    FOR j = 0 WHILE j <  LENGTH masterCombination INCREASE j ← 1
        IF masterCombination[j] = userCombination[j]
            answer[j] ← B
        ELSE
            answer[j] ← isColor()
        END IF
    END FOR
    orderAnswer(answer, colorList)
    RETURN answer
END
```

Pseudocode of the algorithm orderAnswer():

```
BEGIN
    copyAnswer[LENGTH answer]
    FOR i = 0 WHILE i <  (LENGTH answer – 1) INCREASE i ← 1
        FOR l = 0 WHILE l < LENGTH answer INCREASE l ← 1
            copyAnswer[l] ← answer[l];
        END FOR
        IF answer[i] ≠ B
            IF ≠ W
                FOR j =  LENGTH answer – 1 WHILE j > l DECREASE J ← 1
                    IF answer[j] = B
                        answer[i] ← answer[j]
                        answer[j] ← copyAnswer[i]
                        j ← i
                    ELSE IF answer[j] = W
                        answer[i] ← answer[j];
                        answer[j] ← copyAnswer[i];
                        j ← i;
                    END IF
                END FOR
            ELSE
                FOR k = LENGTH answer – 1 WHILE k > i DECREASE k ← 1
                    IF answer[k] = B
                        answer[i] ← answer[k];
                        answer[k] ← copyAnswer[i];
                        k ← i;
                    END IF
                END FOR
            END IF
        END IF
    END FOR
    RETURN answer
END
```

Pseudocode of the algorithm isColor():

```
BEGIN
    answer ← X
    FOR k = 0 WHILE k <  LENGTH masterCombination INCREASE k ← 1
        IF masterCombination[k] = userCombination[position]
            answer[j] ← W
    END FOR
    RETURN answer
END
```

Pseudocode of the algorithm vectorPrinter():

```
BEGIN
    FOR j = 0 WHILE j < LENGTH vector INCREASE j ← 1
        DISPLAY “vector[j]”
    END FOR
END
```

Pseudocode of the algorithm winCondition():

```
BEGIN
    condition ← 0
    FOR i = 0 WHILE i < LENGTH answer INCREASE j ← 1
        IF answer[i] = B
            condition ← condition + 1
        END IF
    END FOR
    IF condition ≥  LENGTH answer
        winCheck ← true
    END IF
    RETURN winCheck
END
```

Pseudocode of the algorithm finalMessage():

```
BEGIN
    IF winCheck = true
        DISPLAY “You win!”
        DISPLAY “Master's Combination:”
        vectorPrinter(masterCombination)
        DISPLAY “Attempts | Answers”
        recordPrinter(turns, userRecord, answerRecord)
    END IF
END
```

Pseudocode of the algorithm recordPrinter():

```
BEGIN
    FOR i = 0 WHILE i < turns INCREASE i ← 1
        FOR j = 0 WHILE j < LENGTH userRecord[i] INCREASE j ← 1
            DISPLAY “userRecord[i][j]”
            IF j = LENGTH userRecord[i] – 1
                FOR k = 0 WHILE k < LENGTH answerRecord[i] INCREASE k ← 1
                    DISPLAY “answerRecord[i][k]”
                END FOR
            END IF
        END FOR
    END FOR
END
```

Lessons learned

During the development of the project, we have learnt:

- The utility of dividing the code into methods in order to:
  - Have to code clearest and tidiest.
  - Reuse the code done before for different parts of the projects.
- To make the method for a general purpose in order to reuse them later.
- To get the data from different types of inputs (in our case, from a file and from the keyboard).
- To read information from a file.
- To work with a huge amount of information at the same time. Because at the theoretical classes and at the laboratories, we have worked with a tiny amount of variables but now we needed more variables in order to solve the project.
- To listen our project mate`s opinion and discuss what is the best solution.