CS 11: MACHINE PROBLEM 2

PYTHON MASTERMIND

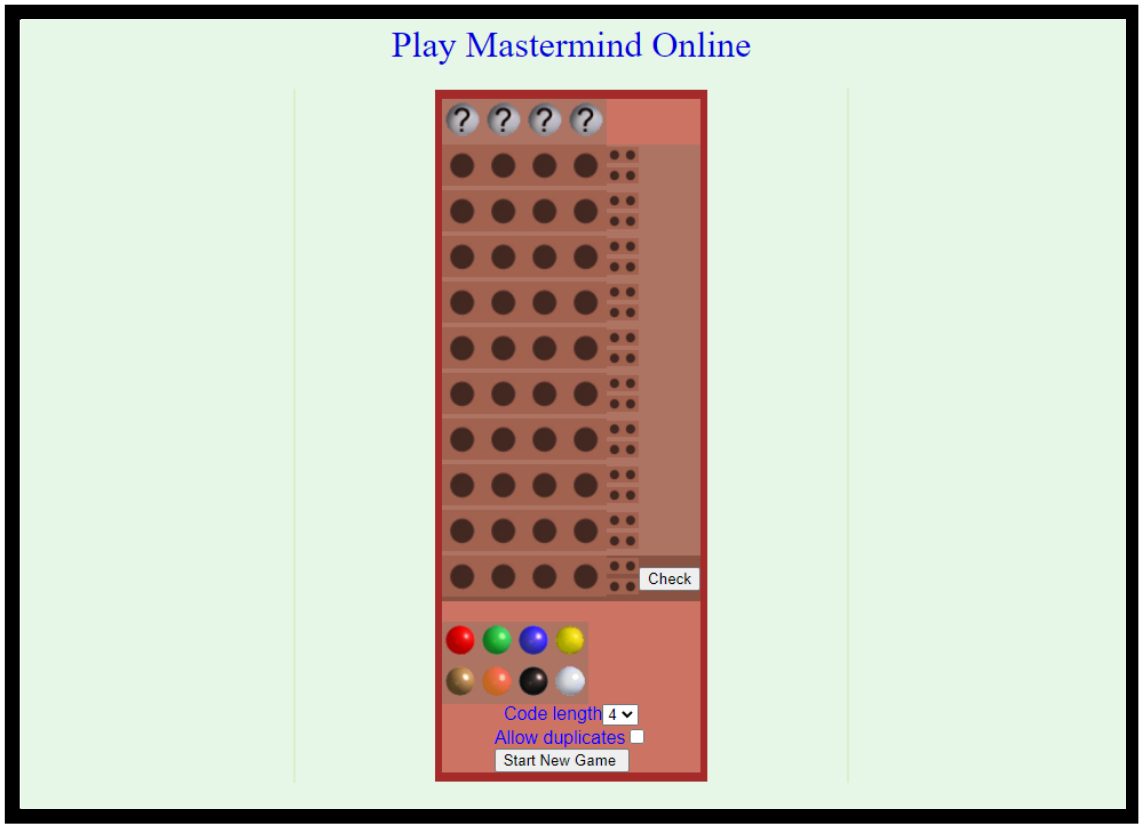
01/10/2022

Prepared For:
GABUD, Roselyn S.

Prepared By:
CALUBAYAN, Gabriel Aldrich S.
ODHUNO, Shane I.

DESCRIPTION

PYTHON MASTERMIND



Mastermind is a game around guessing a pattern of a randomly generated sequence of colors. The computer gives feedback after every incorrect guess that shows whether a guess has the correct colors in the correct positions or the correct colors but in the wrong positions. A player must use these clues to their advantage and guess correctly within 10 tries.

Knowing this, we converted these concepts into Python code and added features that improve the *"quality-of-life"* and correspondence to game mechanics. One of which is the Lifelines which will be elaborated in this document. Also, instead of colors, our game uses numbers instead.

The base game of Mastermind has various features and mechanics which we took into consideration in the different loops, conditionals, and functions we used in our Python program. This implementation document contains a breakdown of every block of code in our program as well as a summary at the end to give an overview of how the algorithm works.

INSTALLING

PYTHON MASTERMIND

Running Python Mastermind directly from our code requires Python to be installed into your device. Python can be installed from this link: **https://www.python.org/downloads/**

It is also advisable to have an integrated development environment (IDE) where you can run the provided code.

Python Mastermind uses Python 3.10.1, the latest version as of January 10, 2022.

CONTACTS

gscalubayan@up.edu.ph
Gabriel Aldrich S. Calubayan

siodhuno@up.edu.ph
Shane I. Odhuno

IMPLEMENTATION

WELCOME MESSAGE AND PRELIMINARIES

Lines 23-45

```
23  welcome = """
24  ============================================
25              Welcome to Mastermind
26  ============================================
27  """
28
29  print(welcome)
30
31  import random as rd
32  import sys
33
34  #main function that executes the game itself
35  def mainCode(theuserinp):
36      #stores patterns you've guessed already
37      history = []
38
39      #stores the revealed positions in the lifelines to prevent repeating clues
40      life_history = []
41
42      #number of guesses and lifelines
43      guesses = 10
44      life1 = 1
45      life2 = 1
```

Lines 23-29 prints out a welcome message indicating the start of the program. Line 31 calls the random module containing ***randint***, which randomly picks an integer from a specified range, and choice, which randomly picks an element from a list. Line 32 calls the **sys** module which contains the ***sys.exit()*** function that stops the program. All the mentioned functions will be used in the succeeding blocks of code.

From line 34, the entire algorithm is composed of three main parts:

- ***mainCode*** function (Lines 35-166) - contains the pattern generator, guessing system, lifelines, and Red and White clues.
- ***user_retry*** function (Lines 169-192) - a function that enables the user to retry the game after winning or losing.
- User input and game execution (Lines 195-212) - asks the user for input and establishes the flow of the game while utilizing the two functions mentioned.

The ***mainCode*** function starts with some preliminaries such as the history list (Line 37) that stores the guesses of the user, the ***life_history*** list (Line 40) that stores the clues that have been shown already, the number of guesses in Line 43, and the counters for Lifelines 1 and 2 (Lines 44-45).

IMPLEMENTATION

RANDOM PATTERN GENERATOR AND GUESS INPUT

Lines 48-59

```
48      length = int(theuserinp)
49      pattern = []
50      for i in range(length):
51          n = rd.randint(0,9)
52          pattern.append(n)
53
54      print('\n' + "Hidden code is of length " + str(length) + ".")
55      print("Total number of Guesses: 10")
56
57      while guesses > 0:
58          print("\n" + "Guess #" + str(11 - guesses))
59          guess = input("Enter guess: ")
```

Lines 48-52 generates a pattern of random length containing random integer elements. Line 48 stores the integer used on the ***mainCode*** function (see page 4) into the variable length which determines the length of the pattern. Lines 50-52 then uses a for loop (within length) to generate random integers (using ***randint*** within range 0-9) and stores it in a list called ***pattern***.

Line 54-55 prints the necessary information to the user before the game starts. This includes the pattern length printed in Line 54, and the number of guesses in Line 55.

The main while loop of the ***mainCode*** function starts at Line 57 which would run while the variable guesses is not zero. Line 58 shows the remaining guesses and Line 59 asks the user for a guess input every time. Lines 58-59 will be shown before every guess as long as the while loop runs.

IMPLEMENTATION

LIFELINE#1: NUMBER CLUE WITHOUT POSITION

Lines 61-78

```
61          #lifeline#1
62          if guess == "Lifeline#1":
63              if guesses > 1:
64                  if life1 > 0:
65                      guesses -= 1
66                      life1 -= 1
67                      digit = rd.choice(pattern)
68                      pos = pattern.index(digit) + 1
69                      while pos in life_history:
70                          digit = rd.choice(pattern)
71                          pos = pattern.index(digit) + 1
72                      life_history.append(pos)
73                      print("Hidden code contains digit " + str(digit))
74                      print("Note: Total number of guesses is reduced by 1.")
75                  else:
76                      print("You have already used Lifeline#1.")
77              else:
78                  print("You don't have enough guesses for Lifeline#1.")
```

Lines 62-72 make it so that when the user types “Lifeline#1”, the computer would print out a number in the pattern (from Lines 48-52) but not its position. As a consequence, the number of ***guesses*** and ***life1*** (Lines 64-65) would both be decreased by one. Upon using Lifeline#1, ***life1*** will be 0, making it so that this lifeline is only used once.

Line 67 picks out a random digit from the pattern (from Lines 49-52) using the choice function (from the ***randint*** module in Line 31) and stores it in the variable called ***digit***. Line 68 then stores the index of digit in the pattern in the variable called ***pos***. Note that 1 is added to ***pos*** because indexing in lists starts at 0, and printing out the value would tell the user the ordinal position of the digit.

Lines 69-71 then checks whether the digit picked from the pattern has been shown already by scanning the elements of ***life_history*** (from Line 40). If the index of the picked digit is already in ***life_history***, the computer picks a new digit from the pattern and stores its index in pos. When the picked digit is not in ***life_history***, it is printed out in Line 73 and its index is then appended to ***life_history*** (in Line 72) so that it would not be picked again. Line 74 informs the user that the number of guesses has been reduced by one. A conditional in Lines 63, 75 and 76 makes sure that Lifeline#1 is only used once and informs the user if try to repeat. In a similar manner, Lines 63, 77, ans 78 prevents the use of lifelines when the user runs out of guesses.

IMPLEMENTATION

LIFELINE#2: CLUE WITH POSITION, REVEAL, AND ISNUMERIC

Lines 80-107

```
80          #Lifeline#2
81          elif guess == "Lifeline#2":
82              if guesses > 2:
83                  if life2 > 0:
84                      guesses -= 2
85                      life2 -= 1
86                      digit = rd.choice(pattern)
87                      pos = pattern.index(digit) + 1
88                      while pos in life_history:
89                          digit = rd.choice(pattern)
90                          pos = pattern.index(digit) + 1
91                      life_history.append(pos)
92                      print("Hidden code contains digit " + str(digit) + " at position " + str(pos))
93                      print("Note: Total number of guesses is reduced by 2.")
94                  else:
95                      print("You have already used Lifeline#2.")
96              else:
97                  print("You don't have enough guesses for Lifeline#2.")
98
99          elif guess == "Reveal":
100             print("GAME OVER!")
101             print("Code: " + str(pattern))
102             user_retry()
103
104         #checks if input is all numbers
105         elif not guess.isnumeric():
106             print ("Invalid guess." + "\n" + "Code only uses symbols 0, 1, 2, 3, 4, 5, 6, 7, 8, and 9")
107             print("Do not put letters, spaces, or any other symbols.")
```

Lines 71-91 make it so that when the user types “Lifeline#2”, the computer would print out a number in the pattern (from Lines 48-52) and its position. As a consequence, the number of guesses and life2 (Lines 84-85) would both be decreased by two and one, respectively. This uses the same concepts explained in Lifeline#1 (Page 6). The only differences is in Line 82 checking if there are enough guesses, Line 84 where guesses are decreased by 2, in Line 92 where the computer also prints ***pos***, and Line 93 informing the user that the number of guesses has been decreased by two.

We also added a feature in Lines 99-96 which checks if the user typed “Reveal” in the guess. If so, the computer would print “GAME OVER!” at Line 94, and the pattern (from Lines 48-52) at Line 95. Typing “Reveal” would also end the game, but gives the user a chance to retry the game by calling the ***user_retry*** function at Line 96. This is either for debugging purposes or if the user wants to quit.

Line 99 checks if the guess input is composed of all numeric characters (012346789) by using the ***.isnumeric()*** function that returns a Boolean value. If it is False, the computer would inform the user of the invalid guess in Line 100, and tell them to not use letters, spaces, and symbols in Line 101. The computer would then ask for another guess without decreasing the number of guesses.

IMPLEMENTATION

WIN, LENGTH, AND HISTORY CHECKERS

Lines 109-124

```
109         else:
110             #converts input into list with integer elements
111             guess = [int(x) for x in str(guess)]
112 
113             #when user guesses correctly
114             if guess == pattern:
115                 print("YOU WIN!!!")
116                 print(winmsg)
117                 guesses == 0
118                 user_retry()
119 
120             #checks if input is of valid length or if it has been guessed already
121             if len(guess) != length:
122                 print ("Invalid guess." + "\n" + "Code is of length " + str(length))
123             elif guess in history:
124                 print ("Invalid guess." + "\n" + "You guessed that already.")
```

If the user does not use Lifelines or Reveal, and inputs a guess containing only numeric characters (0123456789), the user's input in guess (from Line 59) is converted into a list with integer elements in Line 111. The following lines of code in succeeding sections would also execute under the else statement in Line 109.

Lines 114 compares the user's guess to the generated pattern (from Lines 48-52). If the guess and pattern match, the computer prints out "YOU WIN!!" in Line 115, and ***winmsg*** (from Line 4) in Line 116. Guessing correctly ends the game. Line 118 also calls the ***user_retry*** function to let the user choose to retry the game or not.

Line 121 checks if the length of the user's guess input is the same as the length of the pattern (from Lines 48-52). If not, the computer would inform the user of the invalid guess, and the length of the generated pattern in Line 122.

Line 117 checks if the guess is in ***history*** (from Line 37). If so, the computer would inform the user of the invalid guess in Line 124.

In both cases, the computer would then ask for another guess without decreasing the number of guesses.

IMPLEMENTATION

CLUES FOR WRONG VALID GUESSES: R CLUE

Lines 126-145

```
126             #when user guesses wrong
127             else:
128                 guesses -= 1
129 
130                 #right color and position
131                 red = 0
132 
133                 #right color, wrong position
134                 white = 0
135 
136                 #creates a separate list copy of the pattern and user guess
137                 dummy_pattern = pattern[:]
138                 dummy_guess = [int(x) for x in guess]
139 
140                 #loop to set up clues for the player move
141                 for index1, input1 in enumerate(dummy_guess):
142                     if input1 == dummy_pattern[index1]:
143                         red += 1
144                         dummy_pattern[index1] = "checked solution"
145                         dummy_guess[index1] = "checked user"
```

If the user does not use Lifelines or Reveal, inputs a guess of valid length using only numeric characters that has not been guessed already, and the guess is wrong, the computer would print out two clues: R and W. The R clue reveals how many numbers are correct and are in the correct position. The W clue reveals how many numbers are correct, but not in the correct position.

For starters, Line 128 would decrease the number of guesses (from Line 43) for every wrong valid guess. The values of R (red) and W (white) will then be initialized. Next, the program will create a copy, that is a list, of both the generated pattern (Line 137) and the user guess (Line 138).

Afterwards, in Line 141, a for loop will be created that will iterate through and return both the actual elements of the newly created list copy of the user guess (*input1*) as well as their respective indices (*index1*). Then, it is just a matter of checking if the number in the user guess is equal to the number in its respective, relative position in the generated pattern (Line 142). If yes, the *red* count (Line 131) will increment by one (Line 143). The purpose of Lines 144 and 145 is to go through each element in both created dummy lists and replace them with the text "checked solution" and "checked user" respectively for debugging purposes to determine if the number has been checked to pass the above condition.

IMPLEMENTATION

CLUES FOR WRONG VALID GUESSES: W CLUE, AND GAME OVER

Lines 147-166

```
147                 for index1, input1 in enumerate(dummy_guess):
148                     for i, p in enumerate(dummy_pattern):
149                         if p == input1:
150                             white += 1
151                             dummy_pattern[i] = "checked solution"
152                             break
153 
154                 print(str(red) + "R" + " - " + str(white) + "W")
155                 print("You have " + str(guesses) + " guesses left.")
156 
157                 #if there are no guesses left
158                 if guesses == 0:
159                     print("GAME OVER! Try again.")
160                     print("Code: " + str(pattern))
161                     #asks user if they want to try again.
162                     #if yes, game starts over with a new pattern
163                     guesses = 10
164                     user_retry()
165 
166         history.append(guess)
```

For the white clues, the principle is very similar. In Line 147, a for loop was created yet again that will once again iterate through and return both the actual elements of the newly created list copy of the user guess (***input1***) as well as their respective indices (***index1***).

This time, however, an additional for loop will be added underneath (Line 148) that will get both the elements (***p***) as well as the respective indices (***i***) of the generated pattern this time instead of the user guess. Now, with this, Line 149 simply checks if there exists a number in the user guess that is also in the generated pattern. If yes, the white count (Line 134) will then iterate by one.

Similar to before, after all this, the numbers that have been checked will be changed to the text "changed solution" for debugging purposes. Lines 154 and 155 will then print the number of clues for each color as well as the remaining number of guesses, respectively.

When the user runs out of guesses, a "GAME OVER!" message and the generated code would be printed in Lines 159-160. The computer would then run the ***user_retry*** function (Line 164) to let the user retry the game based on their input.

IMPLEMENTATION

USER RETRY FUNCTION

Lines 168-192

```
168  #function that asks the user to retry if so desired
169  def user_retry():
170      retry = input('\n' + "Retry? (Y/N): ")
171      if retry == "Y":
172          diff_option = (input("Do you want to set the difficulty? (Y/N): "))
173          if diff_option == "Y":
174              difficulty = input('\n' + """Please choose among "Easy", "Medium", and "Hard" to determine the length of the pattern: """)
175              if difficulty == "Easy":
176                  mainCode(4)
177              elif difficulty == "Medium":
178                  mainCode(6)
179              elif difficulty == "Hard":
180                  mainCode(8)
181              else:
182                  print('\n' + """Kindly only choose among "Easy", "Medium", and "Hard" difficulties. Please restart and try again.""")
183          elif diff_option == "N":
184              difficulties = [4, 6, 8]
185              mainCode(rd.choice(difficulties))
186          else:
187              print('\n' + """Kindly only choose among "Y" or "N". Please restart and try again.""")
188      else:
189          print()
190          print("Thank you for playing!")
191          print("By Calubayan and Odhuno")
192          sys.exit()
```

Lines 168-192 gives the user the ability to retry the game after a win or a loss. Line 170 asks the user if they want to retry. If so, Line 171-187 executes, which restarts the game. If the user types "N", the computer would print a thank-you message (Lines 190-191) and ends the program using the ***sys.exit*** function in Line 192.

Line 172 asks the user if they want to set the difficulty. If the user neither types "Y" or "N", the computer informs the user of the invalid input, and instructs them to restart the game (Line 187).

Another nested-if executes if the user types "Y", and asks the user to choose between "Easy", "Medium", and "Hard" (Line 174). These difficulties correspond to 4, 6, and 8, respectively, which sets the length of the pattern generated in the ***mainCode*** function executed in Lines 176, 178, or 180.

If the user types "N", the computer randomly picks between 4, 6, and 8 and runs the mainCode function based on that choice (Lines 183-185). If the user neither types "Y" or "N", the computer also informs the user of the invalid input, and instructs them to restart the game (Line 182).

IMPLEMENTATION

USER INPUT AND GAME EXECUTION

Lines 194-212

```
194  #asks the user for the desired length of the pattern
195  print("Before playing, please refer to the User Manual for the Instructions. ")
196  diff_option = (input("Do you want to set the difficulty? (Y/N): "))
197
198  if diff_option == "Y":
199      difficulty = input("""Now, please choose among "Easy", "Medium", and "Hard" to determine the length of the pattern: """)
200      if difficulty == "Easy":
201          mainCode(4)
202      elif difficulty == "Medium":
203          mainCode(6)
204      elif difficulty == "Hard":
205          mainCode(8)
206      else:
207          print('\n' + """Kindly only choose among "Easy", "Medium", and "Hard" difficulties. Please restart and try again.""")
208  elif diff_option == "N":
209      difficulties = [4, 6, 8]
210      mainCode(rd.choice(difficulties))
211  else:
212      print('\n' + """Kindly only choose among "Y" or "N". Please restart and try again.""")
```

After the two functions—***mainCode*** function (Lines 35-166) and ***user_retry*** function (Lines 169-192)—have been defined, Lines 194-212 will execute, which will be the first thing the user sees after the welcome message (Line 29). Line 196 would ask the user if they want to set the difficulty. If the user types "Y", the computer asks the user to pick between "Easy", "Medium", and "Hard" in Line 199. The next lines of code operates similarly to Lines 173-180 in the ***user_retry*** function and runs the ***mainCode*** function to start the game.

If the user types "N", the computer would also randomly pick between 4, 6, and 8 and runs the ***mainCode*** function based on that random choice (Lines 201, 203, or 205). By the way, the selection of the length of the patterns being between 4, 6, and 8 is an attempt to stay true to the original Mastermind Game in https://webgamesonline.com/mastermind/index.php which also generates a pattern between the lengths of 4, 6, and 8 only.

If the user neither types "Y" or "N", the computer also informs the user of the invalid input, and instructs them to restart the game (Line 207).

SUMMARY

PYTHON MASTERMIND

The algorithm is basically composed of one big function (*mainCode*), a *user_retry* function, and code that asks for the difficulty and starts the game. *mainCode* starts off by generating a random pattern based off of the length picked by the user, or a randomly generated length. A while loop then executes (as long as guesses isn't zero) which asks the user for a guess, makes sure the guess is valid, checks if the user wants to use a lifeline, and most importantly, when the guess is correct. When the guess is incorrect, *mainCode* compares the guess with the pattern using for loops, and gives out the red and white clues. It does this 10 times, or until the user guesses correctly. Regardless if the user wins or loses, the *user_retry* function will be called to give the user the option to retry the game.

The Mastermind game looks simple from a bird's-eye view. But there is a lot that happens behind the scenes, so much so that it warrants over 200 lines of code. The Red and White clues, the lifelines, and our additional features such as the duplicate guess checker, retry feature, and difficulty picker really complicate things which was quite discouraging from the start.

Little by little, if statements got nested into other if statements, which also got nested under while and for loops, and so on. Production started with creating a random pattern generator, and a simple while loop that asked the user for a guess ten times. A checker was then added to compare each guess to the generated pattern. This was basically the bare-bones of the Mastermind game. It was only the matter of adding the clues, lifelines, and other features.

The red and white clues were the trickiest to implement. Almost every lesson in CS 11 was utilized in making this algorithm: plotting a draft pseudocode, string and list manipulation, loops, conditionals, functions, and a LOT of debugging. This is an all-around program that would really challenge and teach students the basics of programming and problem solving, in a fun way.

```
24 ============================================
25             Welcome to Mastermind
26 ============================================
```

REFERENCES

ONLINE ARTICLES

GeeksforGeeks. (2020, August 21). Randomly select n elements from list in Python. https://www.geeksforgeeks.org/randomly-select-n-elements-from-list-in-python/

Generating random number list in Python. (n.d.). Tutorialspoint. https://www.tutorialspoint.com/generating-random-number-list-in-python

Mulani, S. (2020, July 30). Exit a Python program in 3 easy ways. AskPython. https://www.askpython.com/python/examples/exit-a-python-program

Play Mastermind online. (n.d.). Webgamesonline. https://webgamesonline.com/mastermind/index.php