Aerial Robotics Kharagpur, Task 2.1

Venkatesh Naresh

Abstract— **Task 2.1 is inspired from 'Wordle' a famous Internet phenomenon, where the player has to guess the correct word, and for every guess, hints are provided. The version here is different; given the number of digits, the player has to guess a hidden number. At every stage the player gets the information of how many guessed digits were in the correct position and how many guessed digits were present in the hidden number, but out of position. The solution discussed is a two step process. Firstly, all unwanted digits are removed from the guess. And secondly, strategic reordering of the digits take place, to finally get the hidden number. The solution is implemented using ROS (Robot Operating System) and the script is written in Python**

I. INTRODUCTION

Task 2.1 Part 1 is a game, where a hidden number is present, and a program which acts as the player of the game, needs to make guesses for the correct number. When the player program decides a guess and sends it, the checker program prints a number 999, if the guess is correct and -999 if the guess doesn't have the same digits as the answer. Otherwise, it prints 'You have D dollars and C cents' where D(referred to as dollars) denotes the number of digits guessed which are present in the answer and at the correct position and C(referred to as cents) denotes the number of digits guessed which are present in the answer but are out of position. The checker also sends a number $10D + C$ to the player program. Using this information the program has to try and guess the correct number. The game stops when the guess matches the answer. First, an initial guess is made and then a strategy is applied which is divided into two stages:

1) Removing the digits not present(referred to as unwanted digits) in the answer, add adding those which are present
2) Rearranging the digits, once all the unwanted digits are removed, to get the final answer

The solution has been implemented using ROS(Robot Operating System) and the scripts for the solution have been written in Python.

II. PROBLEM STATEMENT

Using ROS, we define two **nodes**: **Checker** and **Player**. Two topics, $/check1$ and $/guess\_part1$ have been defined, over which communication between Checker and Player takes place. Player acts as a publisher, and publishes the guess number, called $guess$ on the topic $/guess\_part1$, which is received by the Checker, as a subscriber. Since the number of digits is already known,say n, the Checker program generates a random number in the range $[10^{n-1}$, $10^n - 1]$ with **distinct digits** and assigns this value as the $answer$. The Checker program evaluates D and C from $guess$ and $answer$ and prints them to the console. It also publishes $N = 10D + C$ on the topic $/check1$, which is received by the Player as a subscriber. both N and $guess$ are messages of type $Int64()$. Once, the Player receives N, the problem is to understand the input and devise a strategy to reach $answer$. Subsequent guesses need to be made based on information received from Checker. Suppose a certain guess has n_d digits in the correct position, n_c digits in the answer but out of position and n_u digits which are not present in the answer, then we can say for any guess:

$$n_d + n_c + n_u = n$$

Our objective is to reach the state

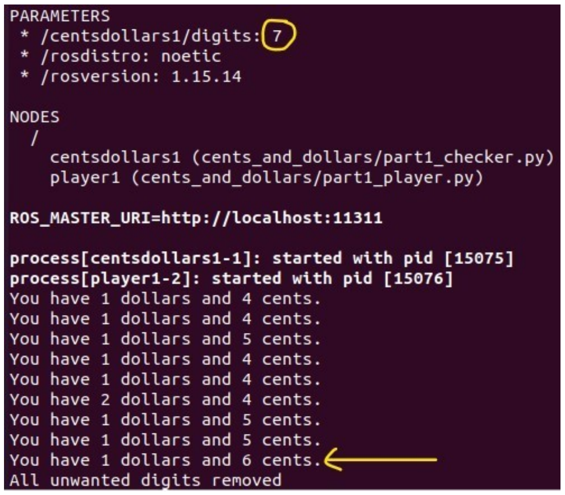
$$n_d = n, n_c = 0, n_u = 0$$

III. RELATED WORK

A possible approach is that, since number of digits are known beforehand, the program can run a loop to check every n digit number with distinct digits. This algorithm will surely yield the answer for smaller n. But for larger n , this method is not efficient, since there are $9 \cdot \frac{10!}{(11-n)!}$ n-digit numbers. Computational time will greatly increase for higher n . Hence, it is advisable to avoid this algorithm

IV. INITIAL ATTEMPTS

The solution which has been implemented consists of an initial guess, and repeatedly modifying the initial guess. The first step is to remove the unwanted digits, i.e. decrease n_u to zero, so that $n_d + n_c = n$. This is achieved by iterating over the digits of $guess$ and replacing the digit with a digit not present in $guess$ currently. If there is an increase in the value of $n_d + n_c$, we maintain the change. Otherwise, we revert back to the previous value of $guess$. Similarly we do for every digit in $guess$ until $n_d + n_c = n$.



```
PARAMETERS
 * /centsdollars1/digits: 7
 * /rosdistro: noetic
 * /rosversion: 1.15.14

NODES
  /
    centsdollars1 (cents_and_dollars/part1_checker.py)
    player1 (cents_and_dollars/part1_player.py)

ROS_MASTER_URI=http://localhost:11311

process[centsdollars1-1]: started with pid [15075]
process[player1-2]: started with pid [15076]
You have 1 dollars and 4 cents.
You have 1 dollars and 4 cents.
You have 1 dollars and 5 cents.
You have 1 dollars and 4 cents.
You have 1 dollars and 4 cents.
You have 2 dollars and 4 cents.
You have 1 dollars and 5 cents.
You have 1 dollars and 5 cents.
You have 1 dollars and 6 cents.
All unwanted digits removed
```

Fig. 1. Demonstrating execution of step 1(removing unwanted digits)

Once, all unwanted digits are removed, we move on to the next stage i.e. reordering the digits to get the final answer. Initially I thought of swapping consecutive digits and observe the change in n_d . But I wasn't able to figure out a definite algorithm of how to swap and till when. So I left that idea, and started working on a new algorithm, which clearly lays out conditions of how and when to swap

V. Final Approach

We set **global** variables $dollars$, $cents$,$new\_dollars$, new_cents,old_1,old_2.We also define $new\_dollarcent$,which stores the value published by the Checker. We define three functions $try\_number()$,$diff()$ and $revert\_back()$. $try\_number(num)$ takes an n-digit number as input then publishes it as a guess. After a delay of 0.1 seconds, the values of $new\_dollars, new\_cents$ get updated since the depending on value of n,since the Checker publishes message on $/check1$. If $n < 10$ then

$$new\_dollars = new\_dollarcent//10\,(floor\,division)$$

$$new\_cents = new\_dollarcent\%10$$

If $n = 10$ then,

$$new\_dollars = \frac{new\_dollarcent - 10}{9}$$

$$new\_cents = \frac{100 - new\_dollarcent}{9}$$

The $diff()$ function returns **two** values: change in new_dollars i.e. change in n_d and change in new_cents i.e change in n_c and also sets $dollars$, $cents$ to $new\_dollars$, new_cents **respectively**. Hence, $dollars$, $cents$ store the **latest** values of n_d and n_c . $revert\_back()$ function is used to reset $dollars$, $cents$ to their previous values.

We define two functions $remove\_unwanted\_digits()$ and $reorder\_digits()$. Firstly,$remove\_unwanted\_digits()$ is called, which convert $guess$ to a string then to a $Python\ list()$ of characters. Then the list is iterated over by index k ; the digit at k is replaced by a digit from the set of all digits 0 to 9 which are not present in the $guess$. If the process results in a positive change in $n_d + n_c$, then the value of $guess$ is updated. Otherwise, $revert\_back()$ is called to reset $dollars$ and $cents$. Eventually, all unwanted digits from the number is removed.

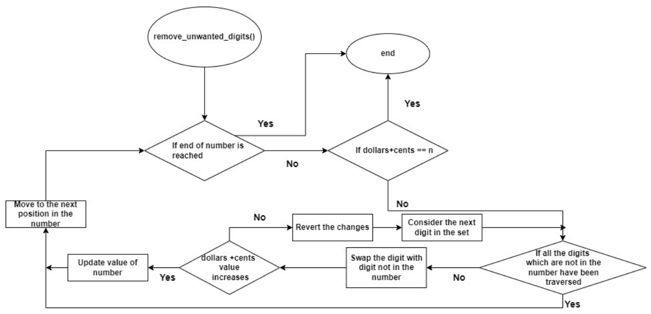

remove_unwanted_digits()
end
Yes
Yes
If end of number is reached
No
If dollars+cents == n
Move to the next position in the number
No
No
Revert the changes
Consider the next digit in the set
Update value of number
dollars +cents value increases
Yes
Swap the digit with digit not in the number
No
If all the digits which are not in the number have been traversed
Yes

Fig. 2. Flowchart representing working of $remove\_unwanted\_digits()*$

Now, $reorder\_digits()$ is called.Consider position i of a digit Y within $answer$. Let's consider cases for corresponding digit in $guess$.

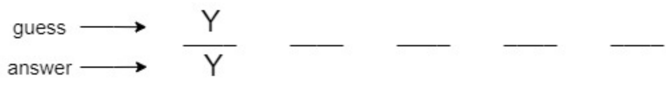


Fig. 3. Case 1*

Suppose Y is present at the correct place in $guess$, then on swapping digit at **any other** position with position i, there will be a decrease in the value of n_d of either **-1** or **-2**, since, we are taking a digit out of its correct position.

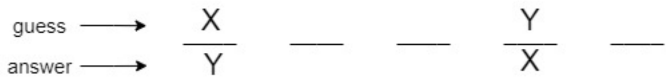


Fig. 4. Case 2*

Suppose, within $guess$ there is a digit $X \neq Y$ present at position i , but digit Y is present at correct position of X,say j, then on swapping digits at other positions with i, there will be **exactly one** such swapping (i with j) which causes an increase of **+2** in n_d , since two digits have come in their correct positions

If both the above cases are not satisfied, then there is only one possible case Suppose correct position of X is j and position of Y in $guess$ is k , then, on swapping i with other positions, change in n_d is non-positive **except at** j **and** k , where a change of **+1** will be observed. Further after swapping i with j , and after updating the value of $guess$,

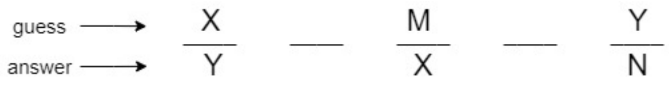


Fig. 5. Case 3*

when i is swapped with k , a change of **+1** is observed. This **fixes** the correct digit at position i . The above principle is used to reorder the digits and get the correct answer.

1) A loop is started which iterates over positions of $guess$
2) For position i in $guess$, another loop is started which swaps digit at position i with digits at indices greater than i
3) For each such i , we check which one of the 3 cases is satisfied. Accordingly, swapping takes place, $guess$ is updated, and correct digit is placed at position i.
4) Finally, we terminate the loop, when the Checker publishes 999 as the message

Using this strategy, we can successfully guess the answer.

VI. Results and Observation

The strategy has been successfully implemented as a python script, and can be run using ROS. The program works as intended for numbers upto 10 digits.

VII. Future Work

The issue which I faced was that, I was unable to test the code for all numbers. For example, if I consider $n = 8$, there are $9 \cdot 10^7$ numbers which are to be tested, which might take hours.There may be a possible recursive approach to this problem statement, compared to the iterative approach I have taken. There may also an algorithm which removes unwanted digits and fixes correct positions simultaneously in contrast to the two-step process proposed above.

CONCLUSION

The task was to implement a strategy, in the form of a program, to guess a hidden number, using hints regarding the number. The ROS interface was used to implement this solution using a pair of publisher-subscriber nodes. The nature of the problem statement demanded the player to make an advantageous change at almost every step. This concept may be used in developing algorithms where the cost of performing a task is to be minimized, like finding the shortest allowable route between two points, or finding an optimal way to win a game.

References

*Figures 2,3,4 and 5 were made by the Author using app.diagrams.net