Learning to Play Mastermind Well Using the Anti-Mind with Feeback Algorithm

Jose Barahona da Fonseca

Department of Electrical Engineering and Computer Science
Faculty of Sciences and Technology
New University of Lisbon
2829-516 Caparica, Portugal,
Email: jbfo@fct.unl.pt

***Abstract*—In a previous work we developed the Anti-Mind algorithm. The Anti-Mind program simulated a good player of the Mastermind game, discovering the secret code defined by the human operator (a sequence of four integers in the interval [0 5]) very quickly. Then we used the algorithm of Anti-Mind to help and correct a human operator trying to discover the secret code defined by the computer resulting in the Anti-Mind with Feedback algorithm. In this paper, we revisited this work and developed another faster implementation of the Anti-Mind with Feedback algorithm which has the drawback that it does not know the set of next good guesses, it just compares each guess with the previous moves and accepts it if it is *coherent* with all the previous moves. Nevertheless, we introduced an option to generate the set of good guesses, i.e., the guesses that are *coherent* with all the previous moves. This implementation allows generalizing the Mastermind game to more than four digits and more than six colours. We begin to define rigorously what we mean by a guess *coherent* with a previous move, next we define what is a good guess and, then, we enunciate five hypotheses about the Anti-Mind algorithm namely one that guarantees that if we always play a good guess we will find the code in a finite bounded number of guesses. We propose a strategy to play Mastermind with the maximization of repetions at the beginning of the game which reduces the *cognitive overload* to play well and validate it with the Anti-Mind with Feedback algorithm. Finally we compare the Anti-Mind algorithm with the Ant-Mind with maximization of repetitions of the guesses through intensive simulations and conclude that the original Anti-Mind algorithm has a better average performance in terms of the number of guesses to break the secret code.**

Keywords–artificial intelligence; mastermind game; anti-mind algorithm; anti-mind with feedback algorithm

I. INTRODUCTION

It seems that the Mastermind game was invented by M. Meyerowitz in 1973 [1]. It is a two players game, the code maker and the code breaker, and the code breaker must find the secret code in at most 10 guesses. The secret code has four colours that may be of one of six colours. In each move the code breaker makes a guess of four colours and the code maker answers with the number of white pegs, *cpe*, the number of correct digits in wrong position, and the number of black pegs, the number of correct digits in right position, *cpc*. In this work the six colours will be represented by the integers in the interval [0 5]. Our algorithm was named Anti-Mind because it was originally created to break the code created by the human player. Then we created a variation of this algorithm where the human plays as the code breaker and the Anti-Mind algorithm is used to generate in each guess the set of good guesses, i.e., the guesses coherent with the previous moves. If the guess does not belong to the set of good moves, then, it is considered a bad move and the user is asked if he wants to see the set of good moves and, then, he must make another guess [2]. In this paper, we show how the human player can improve his skills using the Anti-Mind with Feedback algorithm. When it remains only one hypothesis that must be the secret code and the user is asked to guess the code without any more information. From the use of this algorithm resulted a simplified faster algorithm and a strategy of playing Mastermind.

Our simulations point to a worst case performance of the Anti-Mind algorithm of 9 guesses and Donald E. Knuth [3] showed through exhaustive simulations that his strategy guarantees a maximum of 5 guesses. He showed that the best first guess to guarantee this worst case performance is 1122. After the work of Knuth many proposals were published, e.g. [4]-[7], but no one beat the worst case performance of Knuths strategy of 5 guesses, nevertheless some works improved the average performance over all possible secret codes. Nevertheless, our algorithm is much faster than previous algorithms, since at each stage it selects randomly the next guess from the set of good guesses that are coherent with previous moves. In this sense, the Anti-Mind algorithm can be considered a stochastic algorithm. On the contrary Knuth's algorithm for each good guess generates the next set of good moves for each of the 15 possible combinations of *cpc* and *cpe* and chooses the good guess that *minimizes the maximum number of next good guesses* [3].

This paper is organized as follows. In Section 2, we describe how a good guess is defined based on the concept of the *coherence* of a guess with a previous *move*. In Section 3, we describe the Anti-Mind algorithm. In Section 4, we describe the Anti-Mind with Feedback algorithm. In Section 5, we show how the Anti-Mind with Feedback algorithm can be used to learn to play mastermind well, in Section 6, we present a faster version of the Anti-Mind with feedback algorithm, in Section 7, we present exhaustive simulation results of the Anti-Mind algorithm and the Anti-Mind algorithm with maximization of the number of repetitions of the guesses and in Section 8, we present the conclusions and possible vectors of evolution of our work.

II. DEFINITION OF A MASTERMIND GOOD GUESS

The main idea behind the Anti-Mind algorithm is the *coherence* between a guess and a previous *move*. In Definition 1, we define what is a *move*.

Definition 1: A *move* is the triplet (*guess*, *cpc*, *cpe*) where *cpc* and *cpe* result from the comparison between the *guess* and the secret code.

In Definition 2, we define what is the *coherence* between a *guess* and a previous *move*.

Definition 2: Consider a previous move, *move*, with *cpc* correct digits in right position and *cpe* correct digits in wrong position and a guess, *guess*. Considering that the comparison between the *guess* and the *move* resulted in *cpc_i* and *cpe_i*, then, the *guess* is coherent with *move* if (1) is true.

$$cpc = cpc\_i \text{ AND } cpe = cpe\_i \tag{1}$$

When a guess is coherent with all previous moves, then, we say that it is a *good guess*. On the contrary if the guess is not coherent with at least one previous move, then, we say that it is a *bad guess*. In this sense, we can say that *playing Mastermind well* happens when all guesses are *good guesses*.

So, we can define rigorously the set of good guesses as all combinations that are coherent with all previous moves. This is expressed by (2).

$$\begin{aligned} &set\_good\_guesses = \{\forall combination : \\ &\forall move, cpc = cpc\_i \text{ AND } cpe = cpe\_i\} \end{aligned} \tag{2}$$

Once obtained the set of good guesses, we can verify if a given guess is a good guess, testing if it belongs to the set of good guesses. Alternatively, we can compare the *guess* with all previous moves, and if it is coherent with all of them, then, it is a *good guess*.

III. The Anti-Mind Algorithm

After each guess, the Anti-Mind algorithm obtains the new set of good guesses and selects randomly one of them as the next guess. This is much less computationally expensive than Knuth's algorithm but the Anti-Mind algorithm has a greater worst performance of 8 guesses. In this sense, we can say that the Anti-Mind algorithm is a stochastic algorithm. So we can enunciate five hypotheses that we will show in a near future work that characterize the behaviour of the Anti-Mind algorithm based on emperical data like intensive simulations.

Hypothesis 1: The Anti-Mind algorithm always finds the secret code in less than 9 guesses if the codemaker did not make any error in *cpc* and *cpe* for all previous moves.

Hypothesis 2: When the Anti-Mind algorithm finds the secret code in 8 guesses, if the codemaker did not make any error in *cpc* and *cpe* for all previous moves, the eigth *good* guess is always the secret code. In appendix we show some *enough information* games with 8 guesses.

Hypothesis 3: If the codemaker makes errors in *cpc* and/or *cpe* in at least one previous move, then, the Anti-Mind algorithm always reaches a situation of an empty set of good guesses in less than 9 guesses.

Hypothesis 4: If the game reaches a situation where there is only one good guess, then, this good guess must be the secret code, if the codemaker did not make errors in the previous moves.

Hypothesis 5: Obtaining the new set of good guesses is equivalent to obtain all combinations/guesses that are coherent with all previous moves

The result of hypothesis 2 is the main idea behind our previous work of the Anti-Mind with an unlimited number of lies [8]. When it reaches the conclusion that there is at least one lie in previous moves, reaching the situation of an empty set of good guesses, then, it begins to test the hypothesis of one lie, removing one previous move from the set of previous moves and generating each time the set of good guesses. If it reaches a point where all manners of removing a previous move resulted in an empty set of good guesses, then, it begins to remove two previous moves and so on until it reaches a point where the set of good guesses has a cardinal 1. Then the removed previous moves are the moves with lies and the good guess must be the secret code [8].

Hypothesis 1 can be proved by exhaustive computer search, where for each possible secret code are generated all possible Mastermind games with good guesses and saved the number of guesses when the secret code is found. Since there are only 1296 possible Mastermind secret codes, this computer search is not prohibitive in terms of runtime.

The result of hypothesis 4 is the main idea behind the second version of the Anti-Mind with feedback algorithm where we do not have all good guesses in memory, and generate them whenever it is asked, comparing all combinations with the previous moves and printing the guesses that are coherent with all previous moves- see section 6.

IV. First Version of the Anti-Mind with Feedback Algorithm

In the Anti-Mind with feedback algorithm the human is the codebreaker and the computer the codemaker. At each move, the new set of good guesses is generated, and when the guess does not belong to this set, the user is asked to enter another guess. There is an option that allows the user to see all the good guesses. If it is reached a situation where only one good guess remains, the user is asked to enter the secret code without any more information, since hypothesis 3 guarantees that the remaining good guess must be the secret code. In Table I we present a game of this type.

In Algorithm 1, we describe the Anti-Mind algorithm in detail.

V. Using the Anti-Mind with Feedack Algorithm to Find a Good Playing Strategy

Since we think in terms of symbolic expressions, and taking into account our cognitive limitations, it is easier to begin with a guess with repetitions like 0111. Now if the answer is cpc=3, cpe=0, we can conclude that there exist three 1s in the last three positions, OR one 0 in the first position AND two 1s in two of the last three positions. This symbolical logical expression corresponds to the set of 20 good guesses presented in Table II.

But if for the same secret code, the first guess was 0123, the answer would be cpc=2, cpe=0, which means a much more

Algorithm 1 Anti-Mind with feedback algorithm

$n_digits \Leftarrow$input('Number of Digits=')
$dig_max \Leftarrow$input('Maximum Digit=')
$n\_good\_guesses \Leftarrow (dig\_max + 1)^{n\_digits}$
$secret\_code \Leftarrow generate\_s\_c(n\_digits, dig\_max)$
$set\_good\_guesses \Leftarrow gen\_all\_gs(n\_digits, dig\_max)$
$counter \Leftarrow 0$
while $n\_good\_guesses > 1$ **do**
  $counter \Leftarrow counter + 1$
  $guess \Leftarrow input('Guess =')$
  $cpc \Leftarrow calc\_cpc(secret\_code, guess)$
  $cpe \Leftarrow calc\_cpe(secret\_code, guess)$
  if $cpc == n\_digits$ **then**
    display('You Found It')
    break
  end if
  $flag\_bel \Leftarrow see\_if\_bel(guess, set\_good\_guesses)$
  if $flag\_bel$ **then**
    display_cpc_cpe(cpc,cpe)
    $set\_good\_guesses \Leftarrow calc\_new\_set(guess, cpc, cpe, set\_good\_guesses)$
  else
    display('Bad Move')
    in=input('Want to See Good Guesses?')
    if $in == 1$ **then**
      display_g_g(set_good_guesses)
    end if
  end if
end while
if $n\_good\_guesses = 1$ **then**
  $flag \Leftarrow 0$
  while $1 - flag$ **do**
    $guess$=input('Secret Code=')
    $flag \Leftarrow (guess == secret\_code)$
  end while
end if

complex symbolic expression: 0 and 1 are in right position OR 0 and 2 are in right position OR 0 and 3 are in right position OR 1 and 2 are in right position OR 1 and 3 are in right position OR 2 and 3 are in right position. This complex symbolical logical expression corresponds to a much greater set of 96 good guesses. In Table III we show these good guesses.

So in terms of *cognitive overload* it seams better to play in the beginning with repetitions.

VI. Second Version of the Anti-Mind with Feedback Algorithm

In the first version of the algorithm we first generate all combinations, and for each guess and cpc and cpe we generate a new set of good guesses, and we decide if the new guess is good, testing if the guess belongs to the set of good guesses. For a genarilized version of Mastermind with more digits and a greater maximum digit, this can take a lot of time in the first moves. So, inspired in this strategy, we created a new version of the algorithm where we only compare the guess with previous moves, and accept it if it is coherent with all previous moves. This algorithm does not have the information of the number of good guesses and it is impossibile the detection of an *enough information* situation, but it is much faster and we can play more difficult generalized Mastermind games inside Matlab environment. In Algorithm 2, we describe in detail this second version of the Anti-Mind with feedback algorithm.

TABLE I. Example of a Game with 8 Guesses

```
anti_mind_real(4,5,1, 1)
Number of Possible Good Guesses=1296
move 1=0111
cpc=0
cpe=1
Number of Possible Good Guesses=308
move 2=1222
cpc=1
cpe=0
Number of Possible Good Guesses=90
move 3=1333
cpc=1
cpe=0
Number of Possible Good Guesses=20
move 4=1444
cpc=0
cpe=1
Number of Possible Good Guesses=6
move 5=4320
cpc=2
cpe=2
Number of Possible Good Guesses=3
move 6=4302
cpc=1
cpe=3
Number of Possible Good Guesses=2
move 7=4023
cpc=1
cpe=3
*ENOUGH INFORMATION**
Secret Code=4230
*You Found It! in 8 Guesses, with 0 bad Guesses and 0 hints **
```

TABLE II. Set of Good Guesses After Guess 0111

0 0 1 1	0 1 0 1	0 1 1 0	0 1 1 2	0 1 1 3	0 1 1 4
0 1 1 5	0 1 2 1	0 1 3 1	0 1 4 1	0 1 5 1	0 2 1 1
0 3 1 1	0 4 1 1	0 5 1 1	1 1 1 1	2 1 1 1	3 1 1 1
4 1 1 1	5 1 1 1				

TABLE III. Set of Good Guesses After Guess 0123

0 0 0 3	0 0 2 0	0 0 2 2	0 0 2 4	0 0 2 5	0 0 3 3
0 0 4 3	0 0 5 3	0 1 0 0	0 1 0 1	0 1 0 4	0 1 0 5
0 1 1 0	0 1 1 1	0 1 1 4	0 1 1 5	0 1 4 0	0 1 4 1
0 1 4 4	0 1 4 5	0 1 5 0	0 1 5 1	0 1 5 4	0 1 5 5
0 2 2 0	0 2 2 2	0 2 2 4	0 2 2 5	0 3 0 3	0 3 3 3
0 3 4 3	0 3 5 3	0 4 0 3	0 4 2 0	0 4 2 2	0 4 2 4
0 4 2 5	0 4 3 3	0 4 4 3	0 4 5 3	0 5 0 3	0 5 2 0
0 5 2 2	0 5 2 4	0 5 2 5	0 5 3 3	0 5 4 3	0 5 5 3
1 1 1 3	1 1 2 1	1 1 2 2	1 1 2 4	1 1 2 5	1 1 3 3
1 1 4 3	1 1 5 3	2 1 2 1	2 1 2 2	2 1 2 4	2 1 2 5
2 2 2 3	2 3 2 3	2 4 2 3	2 5 2 3	3 1 1 3	3 1 3 3
3 1 4 3	3 1 5 3	3 2 2 3	3 3 2 3	3 4 2 3	3 5 2 3
4 1 1 3	4 1 2 1	4 1 2 2	4 1 2 4	4 1 2 5	4 1 3 3
4 1 4 3	4 1 5 3	4 2 2 3	4 3 2 3	4 4 2 3	4 5 2 3
5 1 1 3	5 1 2 1	5 1 2 2	5 1 2 4	5 1 2 5	5 1 3 3
5 1 4 3	5 1 5 3	5 2 2 3	5 3 2 3	5 4 2 3	5 5 2 3

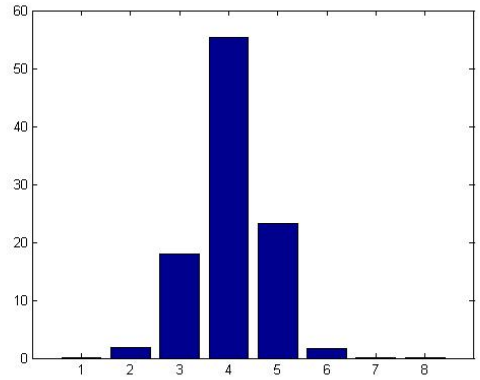
VII. Simulation Results

To evaluate the performance of the Anti-Mind algorithm, we made an exhaustive simulation over all possible secret codes, with 1000 runs for each secret code. In this simulation, the computer is the codemaker and the codebreaker. In Figure 1 we show the distribution of runs by the number of guesses to find the secret code. Then we repeat the simulation maximizing the number of repetitions of each guess. In Figure 2 we show

Algorithm 2 Second version of Anti-Mind with feedback

$flag\_not\_found \Leftarrow 1$
$n\_digits \Leftarrow$ input('Number of Digits=')
$dig\_max \Leftarrow$ input('Maximum Digit=')
$secret\_code \Leftarrow generate\_s\_c(n\_digits, dig\_max)$
$counter \Leftarrow 0$
while $flag\_not\_found$ **do**
  $counter \Leftarrow counter + 1$
  $guess \Leftarrow input('Guess =')$
  $cpc \Leftarrow calc\_cpc(secret\_code, guess)$
  $cpe \Leftarrow calc\_cpe(secret\_code, guess)$
  $cpc\_o(counter) \Leftarrow cpc$
  $cpe\_o(counter) \Leftarrow cpe$
  $guess\_o(counter) \Leftarrow guess$
  if $cpc = n\_digits$ **then**
    display('You Found It')
    break
  end if
  for i=1:counter-1 **do**
    if $1 - flag\_bad\_guess\_o(i)$ **then**
      $cpc\_i \Leftarrow calc\_cpc(guess, guess\_o(i))$
      $cpe\_i \Leftarrow calc\_cpe(guess, guess\_o(i))$
      $flag\_bad\_guess \Leftarrow 1 - (cpc\_i = cpc) + 1 - (cpe\_i = cpe)$
      if flag_bad_guess **then**
        break
      end if
    end if
  end for
  $flag\_bad\_guess\_o(conter) \Leftarrow flag\_bad\_guess$
  if flag_bad_guess **then**
    display('Bad Guess!')
    in=input('Do you want to see the good guesses?')
    if $in == 1$ **then**
      combination=zeros(1,n_digits)
      $flag\_end \Leftarrow 0$
      while 1-flag_end **do**
        for i=1:counter-1 **do**
          $cpc\_i \Leftarrow calc\_cpc(combination, guess\_o(i))$
          $cpe\_i \Leftarrow calc\_cpe(combination, guess\_o(i))$
          $flag\_coher \Leftarrow 1 - (cpc\_i = cpc\_o(i)) + 1 - (cpe\_i = cpe\_o(i))$
          if flag_coher==0 **then**
            break
          end if
        end for
        if flag_coher **then**
          display(combination)
        end if
        $combination \Leftarrow gen\_next\_comb(combination)$
        $flag\_end \Leftarrow all\_combs(combination)$
      end while
    end if
  end if
  $flag\_not\_found \Leftarrow 1 - (cpc = n\_digits)$
end while

the results of this simulation. Comparing the two figures, we can say that the results of the second simulation are worse than the results of the first simulation, since we have a greater percentage of runs with 5 guesses. This way we can say that playing mastermind *well*, with repetitions, has a lower performance than the Anti-Mind algorithm performance. So we confirm that the computer *thinks* better than the human [2].



□

FIGURE 1. NUMBER OF GUESSES IN PERCENTAGES DISTRIBUTION OF THE ANTI-MIND ALGORITHM.

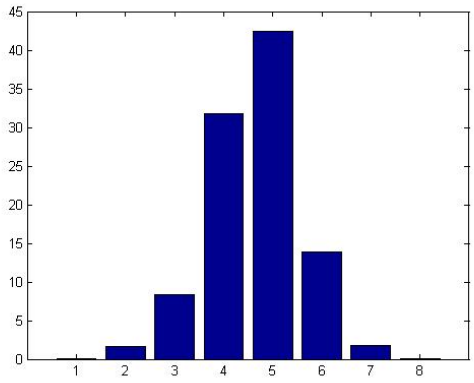


FIGURE 2. NUMBER OF GUESSES IN PERCENTAGES DISTRIBUTION OF THE ANTI-MIND ALGORITHM WITH MAXIMIZATION OF REPETITIONS OF GUESSES.

VIII. CONCLUSIONS AND FUTURE WORK

We showed how to define rigorously what it is meant by playing Mastermind well and how the Anti-Mind algorithm can be used to learn playing well, resulting in the Anti-Mind with feedback algorithm. In the near future, we plan to demonstrate the five hypotheses enunciated in this work as theorems.

REFERENCES

[1] "Games Gift Guide," Games and Puzzles, vol. 20, pp. 16-17, December 1973.

[2] J. Barahona fa Fonseca, "Anti-Mind and Anti-Mind with Feedback: an Example where the Computer Thinks better than the Human," Proc. Congress in Cognitive Neurosciences, University of Evora, Nov. 2003, pp. 55-61.

[3] D. E. Knuth, "The Computer as Mastermind," Journal of Recreational Mathematics, vol. 9, 1976, pp. 1-6.

[4] V. Chvtal, "Mastermind," Combinatorica, vol. 3, 1983 pp. 325-329.

[5] M.M. Flood, "Mastermind Strategy," Journal of Recreational Mathematics, vol. 18, 1985, pp. 194-202.

[6] M.M. Flood, "Sequential Search Sequences with Mastermind Variants-part 1," Journal of Recreational Mathematics, vol. 20, 1988, pp. 105-126.

[7] M.M. Flood, "Sequential Search Sequences with Mastermind Variants-part 2," Journal of Recreational Mathematics, vol. 20, 1988, pp. 168-181.

[8] J. Barahona da Fonseca, "The Anti-Mind with Unlimited Number of Lies as a First Step to Detective Reasoning Modeling," Proc. ICIEIS 2014 Conference, ICIEIS Press, Nov. 2014, pp. 200-205.

APPENDIX

Enough Information Games with 8 Guesses

```
>> anti_mind_real(4,5,1, 1)
Number of Possible Good Guesses=1296
move 1=0111
cpc=0
cpe=1
Number of Possible Good Guesses=308
move 2=1222
cpc=1
cpe=0
Number of Possible Good Guesses=90
move 3=1333
cpc=1
cpe=0
Number of Possible Good Guesses=20
move 4=1444
cpc=0
cpe=1
Number of Possible Good Guesses=6
move 5=4320
cpc=2
cpe=2
Number of Possible Good Guesses=3
move 6=4302
cpc=1
cpe=3
Number of Possible Good Guesses=2
move 7=4023
cpc=1
cpe=3
**ENOUGH INFORMATION**
move 8=4230
**You Found It! in 8 Guesses, with 0 bad Guesses and 0 hints **

Number of Possible Good Guesses=1296
move 1=2201
cpc=0
cpe=2
Number of Possible Good Guesses=222
move 2=5320
cpc=3
cpe=0
Number of Possible Good Guesses=8
move 3=5322
cpc=2
cpe=0
Number of Possible Good Guesses=7
move 4=5520
cpc=2
cpe=0
Number of Possible Good Guesses=4
move 5=5310
cpc=2
cpe=0
Number of Possible Good Guesses=3
move 6=4320
cpc=3
cpe=0
Number of Possible Good Guesses=2
move 7=0320
cpc=3
cpe=0
**ENOUGH INFORMATION**
move 8=3320
**You Found It! in 8 Guesses, with 0 bad Guesses and 0 hints **

Number of Possible Good Guesses=1296
move 1=2241
cpc=1
cpe=1
Number of Possible Good Guesses=230
move 2=2532
cpc=0
cpe=2
Number of Possible Good Guesses=34
move 3=5121
cpc=0
cpe=2
Number of Possible Good Guesses=10
move 4=1345
cpc=0
cpe=2
Number of Possible Good Guesses=6
move 5=4254
cpc=1
cpe=0
Number of Possible Good Guesses=3
move 6=3213
cpc=3
cpe=0
Number of Possible Good Guesses=2
move 7=3210
cpc=2
cpe=2
**ENOUGH INFORMATION**
move 8=0213
**You Found It! in 8 Guesses, with 0 bad Guesses and 0 hints **

Number of Possible Good Guesses=1296
move 1=0214
cpc=1
cpe=0
Number of Possible Good Guesses=108
move 2=1111
cpc=0
cpe=0
Number of Possible Good Guesses=81
move 3=0300
cpc=0
cpe=1
Number of Possible Good Guesses=29
move 4=3444
cpc=1
cpe=0
Number of Possible Good Guesses=9
move 5=3232
cpc=3
cpe=0
Number of Possible Good Guesses=4
move 6=3233
cpc=2
cpe=0
Number of Possible Good Guesses=2
move 7=3252
cpc=3
cpe=0
**ENOUGH INFORMATION**
move 8=3222
**You Found It! in 8 Guesses, with 0 bad Guesses and 0 hints **

Number of Possible Good Guesses=1296
move 1=1243
cpc=0
cpe=2
Number of Possible Good Guesses=312
move 2=2551
cpc=1
cpe=0
Number of Possible Good Guesses=50
move 3=3450
cpc=0
cpe=2
Number of Possible Good Guesses=12
move 4=4001
cpc=1
cpe=1
Number of Possible Good Guesses=4
move 5=2024
cpc=0
cpe=1
Number of Possible Good Guesses=3
move 6=0331
cpc=3
cpe=0
Number of Possible Good Guesses=2
move 7=0131
cpc=2
cpe=2
**ENOUGH INFORMATION**
move 8=0311
**You Found It! in 8 Guesses, with 0 bad Guesses and 0 hints **
```