

**DATA STRUCTURE AND ALGORITHM**

**FINAL ASSIGNMENT**

**Group name:**   Group 14

**Group members**: Tran Minh Quang (s3757281)

Tran Khang Duy - s3754942

Le Thanh Tai – s3760615

**Date:** 1/2021

Contents

[**Introduction** 3](#_Toc61912113)

[**Algorithm** **Explanation** 3](#_Toc61912114)

[**Algorithm Analysis** 5](#_Toc61912115)

[**Discussion** 6](#_Toc61912116)

[**Conclusion** 8](#_Toc61912117)

# **Introduction**

“Guess My Number” is a classic two-player game and it is an extended version of the famous mastermind game. It is a code breaking game where one side have a secret 4-digits code and other side will try to guess the code with limited number of guesses. After each guess, code maker will give back two numbers:

* The number of digits have correct value and correct position (strikes)
* The number of digits have correct value but are not placed in right position (hits)

In the final assignment, our team will study and implement an effective algorithm to break the code with minimum number of guesses. Basically, to get the secret code with the small number of guesses, we cannot use normal mindset to get the hint for each guess. However, we can rely on the computational strength of the computer and an optimal strategy to acquire the best the best strategy.

We will apply the algorithm invented by Donald E. Knuth [1], the original algorithm is made for the original mastermind game with 4 pegs and 6 colors. Hence, some modification has been made compare to the original algorithm to make it suitable for the “Guess My Number” and some improvement for optimization purpose.

# **Algorithm** **Explanation**

There exist several algorithms which can dealing with the game quite effectively, however, after some research and skim through all possible algorithms from different version of Knuth’s algorithm or genetic algorithm. We have decided to implement the Five-Guess Algorithm.

The idea is using Minimax (1) to optimize the potential guessing. We will have a list of potential numbers, then we pick one of them as a guess. After getting strikes and hits, we will remove all the numbers which do not result in the same answer from the last guess.

The procedures of the minimax algorithm:

Firstly, save all the potential numbers in a list. At the beginning when we have no hints, we will save all the numbers from 1000 to 9999.

list potentialNumbers

For i = 1000 to 9999:

potentialNumbers.add(i);

return potentialNumbers;

Generating a random number in the potential list and make a guess.

lastGuess = potentialNumber[random(0, potentialNumber(size))];

return lastGuess;

After getting the strikes and hits from the first guess, remove from the list the numbers that do not provide the same hits and strikes with that guess.

for i = 0 to potentialNumber.size():

If (getHitsAndStrikes(potentialNumber[i])! = getHitsAndStrikes(lastGuess))

Remove potentialNumber[i]

Repeat from step 2 until we get 4 strikes.

**Pseudo code:**

Function createAllPotentialNumbers():

list potentialNumbers

For i = 1000 to 9999:

potentialNumbers.add(i);

return potentialNumbers;

Function filterPotentialNumbers (potentialNumbers, lastGuess):

for i = 0 to potentialNumber.size():

If (getHitsAndStrikes(potentialNumber[i]) != getHitsAndStrikes(lastGuess)):

potentialNumber[i].remove();

list potentialNumbers = createAllPotentialNumber();

While ( number of strike != 4):

lastGuess = potentialNumbers[random(0; potentialNumbers.size()];

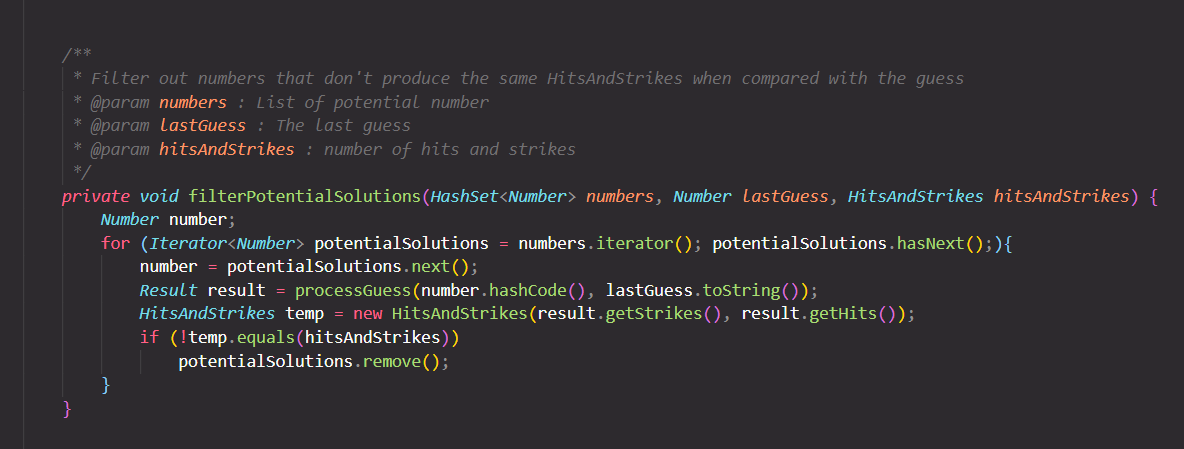
If (getStrikes(lastGuess) == 4)

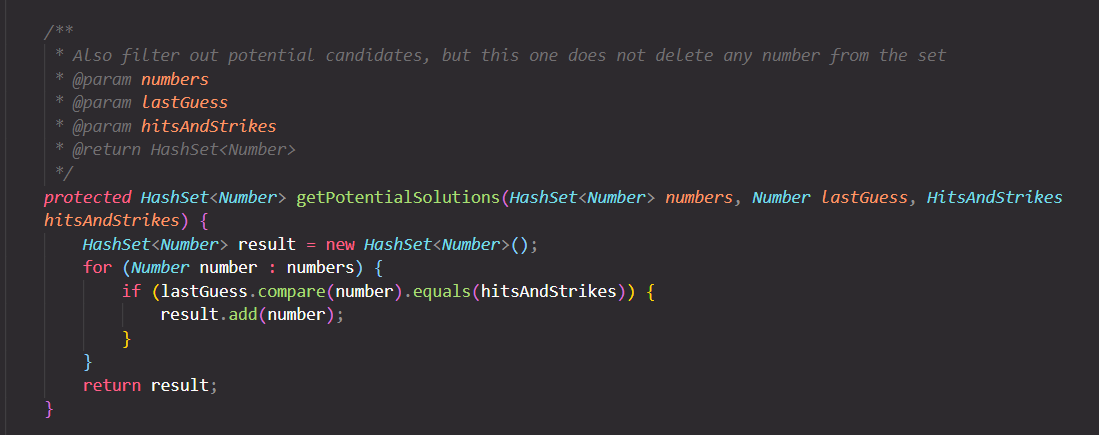
Break;

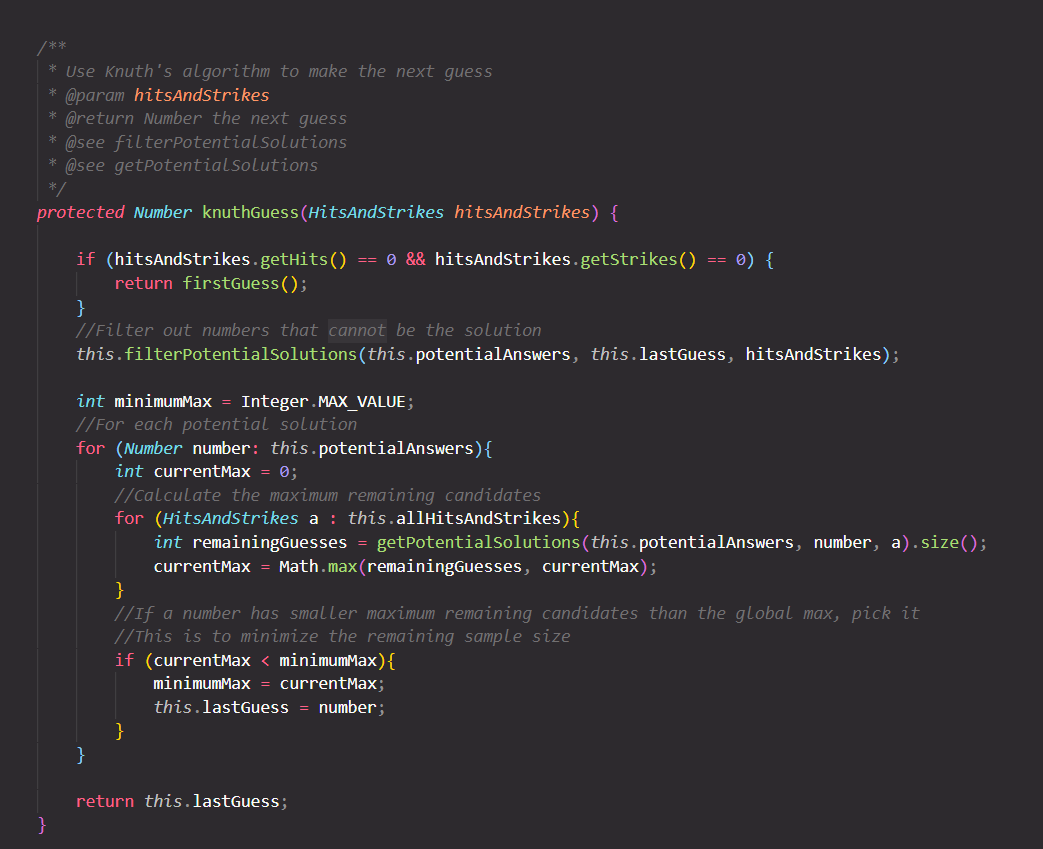
FilterPotentialNumbers(potentialNumbers, lastGuess);

Endwhile

The implementation of the algorithm is shown in the code snippet bellow:

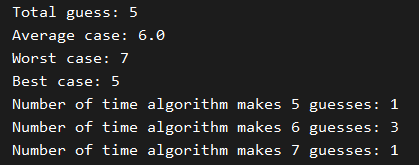




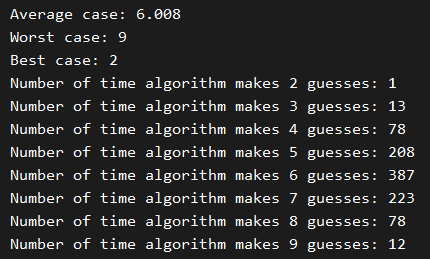


# **Algorithm Analysis**

In this algorithm analysis, we will focus on the optimization of the problem which is the number of guesses the algorithm use until fine the correct number. We will try to play the game multiple times to gather the average case, best and worst case of the algorithm. Firstly, let run 5 times and see the result:



As we can see, the average guesses the algorithm need is 6, the worst case is 7 and the best case is 5. Now, we will run it 1000 times to observe the statistic in number of guesses.



We can see that, after 1000 times, the average number of guess is still 6 which is similar to the number we have after only 5 runs. Therefore, we can conclude that the average number of guess of our algorithm is about 6 guessed. Referring to the best case and worst case, the best case show that it needs only two guessed to break the code, but it is rarely happen, while the worst case is 9 guess. Overall, the algorithm performs as expected with good efficiency. The graph below illustrated the distribution of number of guessed made. We can see that most of the time, the algorithm will need from 5 to 7 guessed to break the code.

# **Discussion**

For optimization, other optimization and special algorithm have been applied to increase the efficiency of the algorithm. Firstly, is the minimax algorithm to support in the guess process.

Minimax is the algorithm which act as a decision helper (happens in AI (Artificial Intelligence), game theory, statistic,) constructed for lowering the chances of getting the worst-case scenario as low as possible. In the game theory, this algorithm was specially used for concluding the optimal move for a player, while assuming the opponent also trying the best for the optimal move. Turn-based game are highly depending on this algorithm.

Formula of the algorithm:



Where vi : value function for player I.

i:  Index of the main player that the calculation solves for

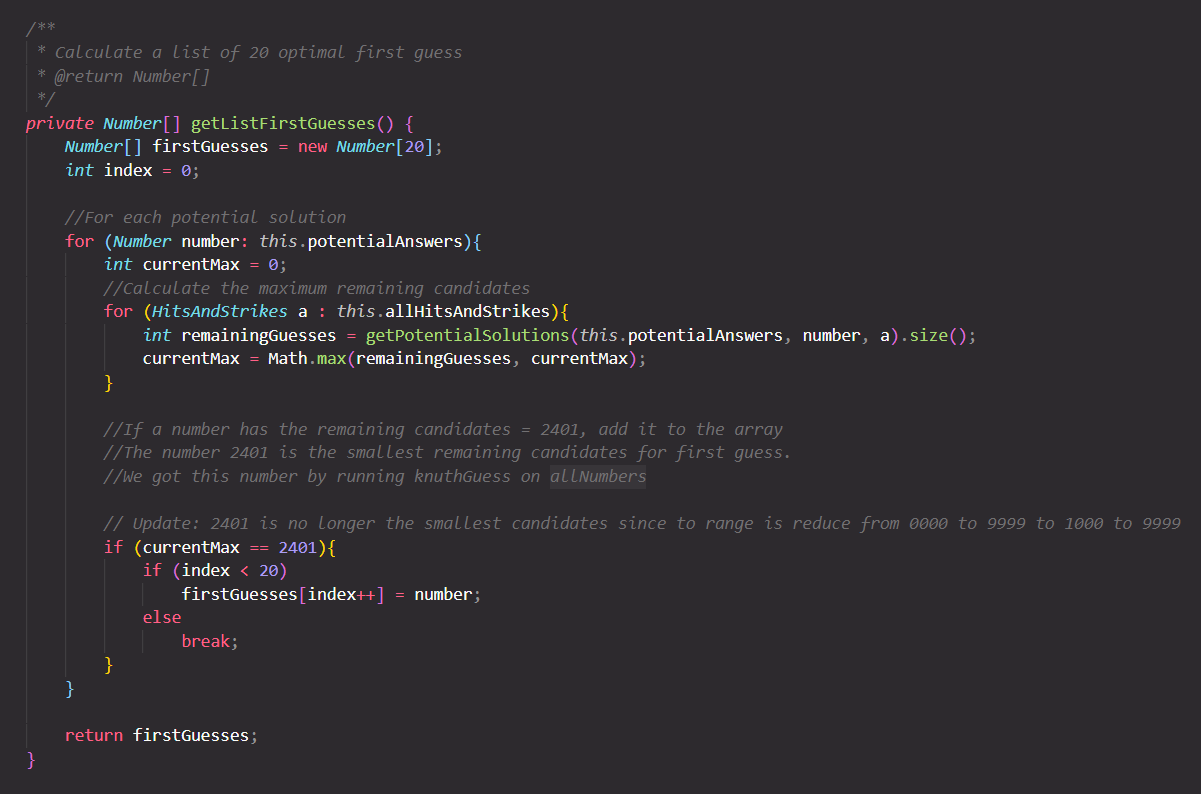
-i: all other competitive players except i.

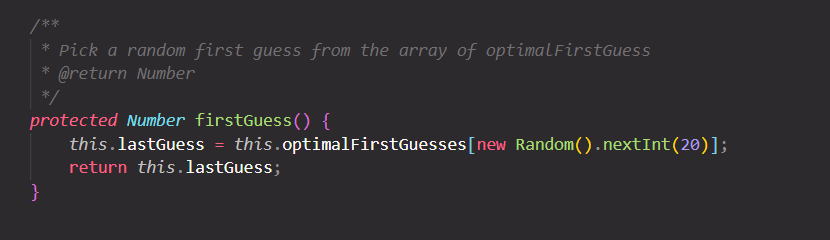
ai: the action which has been executed by the main player.

a-i: all the actions executed by other players except i.

Secondly, instead of using 1D and 2D array to store the number, we use the hash set and custom object to increase the computational time and efficiency of the algorithm. Hash table itself is the data structure where the data stores inside it is linked with each other in certain rules. Array is the main format for storing data of hash table, and since each of the element of the data will received their own unique index, any access to these data will be much faster. And because of these, the size of the data will not matter, cause the insertion and the search function will be execute fast regardless the size. In the hashing table, the most important part is to has the data values into the index that belong to the array.

Finally, another noticeable modification in the algorithm is the way we handle the first guess. In the original algorithm, the first guess is fixed to a number such as 1122 in mastermind(4,6), however, in the context of this game when the population is large, a fixed first guess is not optimized for the algorithm. Hence, we have filter from the list 20 optimal first guess and pick one of them randomly.





# **Conclusion**

In conclusion, we have successfully developed an algorithm to break the code based on Knuth’s five-guess algorithm. The algorithm needs 6 guesses in average to break the code which can be consider optimal. Also, the data structure hash set is applied to increase efficiency and optimization of the algorithm.