**Mastermind Board Game**

**Overview**

Mastermind is a code-breaking board game designed for two players. The game is played using a decoding board, code pegs of at least 6 different colours, and key pegs which are coloured black or white and are smaller than code pegs. One player is the *codemaker* and must choose a pattern of 4 code pegs (such as red, red, green, blue) as their secret code for the game. The other player is the *codebreaker* and must try to guess the pattern within as few turns as possible.

A detailed explanation of the rules can be found on Wikipedia: <https://en.wikipedia.org/wiki/Mastermind_(board_game)>

**The Solving Algorithm**

In the classic version of mastermind, there are 4 pegs of 6 colours, and so there are 6^4=1296 total patterns/potential codes.

* Colours can be represented by numbers 1-6
* A code can therefore be represented by a string of 4 numbers: “1122” = “red-red-blue-blue”
* We can let S be the set of 1296 possible codes: S =[1111,1112,…,6665,6666]
* A response can be a tuple in the form: (*b, w*) - for example (2,1) is two black and one white peg.

If we guess “1122” and we get (0,0) as a response, then the size of S reduces from 1296 to 256, as there are only 256 codes that would give that exact response. Similarly, if we get (0,3) as a response then there are only 16 possible codes that it could be.

We can therefore split all of the codes in S up into different groups based on their response to our guess, as shown in the diagram below:

#///

We can use this to come up with an expected value of how many codes will be remaining after a certain guess. For each response, we should calculate the probability of that response occurring, multiplied by how many codes remain if the response occurs. We then add each result up to get the expected value for that guess. If we do this for each guess in S, we can then choose the guess that has the lowest expected value, as the guess with the lowest expected value is the one that on average leaves the smallest number of possible codes remaining (and is therefore the best guess).

**Note:** This is just a heuristic and isn’t the optimal solution: the heuristic assumes that fewer remaining codes will result in fewer guesses, however this is not always true.

**Mastermind Board Game**

**Overview**

Mastermind is a code-breaking board game designed for two players. The game is played using a decoding board (shown below), code pegs of at least 6 different colours, and key pegs which are coloured black or white and are smaller than code pegs.

One player is the *codemaker* and must choose a pattern of 4 code pegs (such as red, red, green, blue) as their secret code for the game.

The other player is the *codebreaker* and must try to guess the pattern within as few turns as possible.

Each guess that the *codebreaker* makes will be marked/evaluated by the *codemaker* using the key pegs.

A black key peg indicates a correct coloured peg in a correct position, and a white key peg indicates a correct coloured peg in the wrong position.

The *codebreaker* doesn’t know the link between key pegs and code pegs, and so they must move code pegs around and make different guesses in order to figure out the correct code.

The codebreaker can win the game by guessing the code, or lose the game by running out of guesses.

For example, if we have a code of (red – green – green – blue), and the codebreaker guesses (red – blue – black – green), then this would be marked with 1 black and 2 white key pegs:

Code : red – green – green – blue

Guess : red – blue – black – green

Mark: 1 black and 2 white

In the case above the codebreaker would know that one of the code pegs was correct, but would not know whether it was the red,blue,black or green one.

**The Solving Algorithm**

If we assume that the pattern consists of 4 pegs, and each peg can be 8 different colours, then we have 8^4=4096 different possible codes at the start of the game.

Let us take the guess red-red-blue-blue as an example:

If we look at this guess within the context of all of the possible codes

If we think of all of these codes as being in some set S, then we can create a function which evaluates a certain guess based on how much that guess reduces the number of codes in S on average. So in order to find the optimal guess, we just need to find the guess which reduces the number of possible remaining codes the most.

We use an algorithm that evaluates guesses based on the average number of codes removed from the set. However this might not be a good heuristic, as it is possible for there to be a set S1 that is smaller than a set S2, but that actually takes more moves to reduce. For example if S1 contains [1-1-1-a for a in range(8)], then it is of size 8 and takes up to 8 guesses, however S2 can contain 16 guesses that are all evenly split.