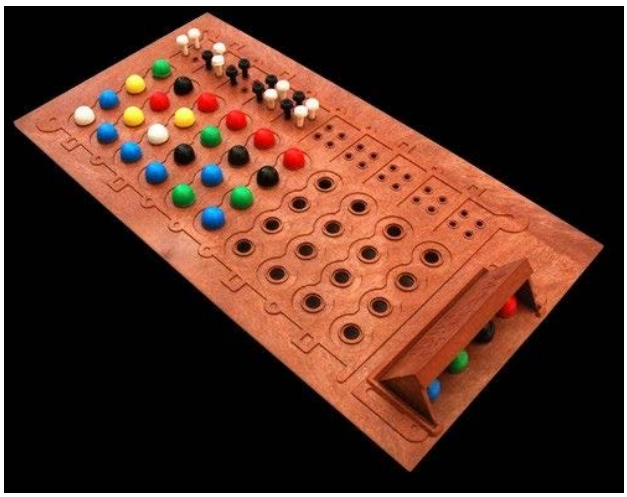
Dept. of Computer Science and Engineering
CS69011: Computing Lab 1

Assignment 3 **Maximum Marks: 30**

Mastermind is a popular mind game, requiring considerable logical reasoning skills. To learn the game, you may refer to the Wiki page: https://en.wikipedia.org/wiki/Mastermind_(board_game)
You may also play it online at: https://www.webgamesonline.com/mastermind/

Here is a description of the game from the Wiki-page with some simplifying assumptions:

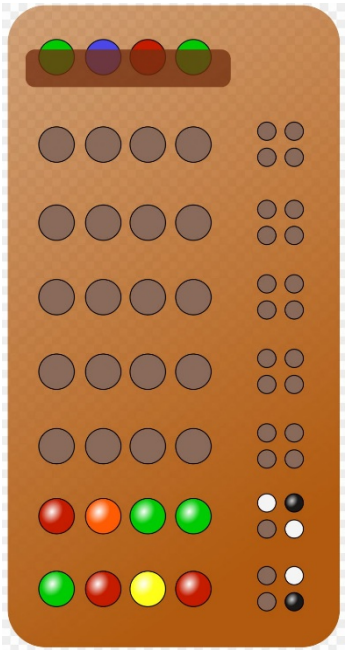


The game is played using a decoding board, with a shield at one end covering a secret row of four large holes, and ten additional open rows – each containing four large holes next to a set of four small holes.

There are *code pegs* of eight different colours with round heads, which will be placed in the large holes on the board.

There are also *key pegs*, some coloured black, some white, which are flat-headed and smaller than the code pegs; they will be placed in the small holes on the board.

One player becomes the codemaker, the other the codebreaker. The codemaker chooses a pattern of four code pegs. Duplicates and blanks are allowed, so the player could even choose four code pegs of the same colour. The chosen pattern is placed in the four holes covered by the shield, visible to the codemaker but not to the codebreaker. For example, the hidden code is green-blue-red-green in the picture below.



The codebreaker tries to guess the pattern. Each guess is made by placing a row of code pegs on the decoding board. Once placed, the codemaker provides feedback by placing from zero to four key pegs in the small holes of the row with the guess. A coloured or black key peg is placed for each code peg from the guess which is correct in both colour and position. A white key peg indicates the existence of a correct colour code peg placed in the wrong position. In the picture, the first guess, green-red-yellow-red, gets a black key peg for the first green and a white key peg for the red.

If there are duplicate colours in the guess, they cannot all be awarded a key peg unless they correspond to the same number of duplicate colours in the hidden code. For example, in the second guess, the player guesses red-orange-green-green, and is awarded a black peg for the rightmost green, and white pegs for the red and the other green. Here both green pegs receive a key peg (one black and one white) as the hidden code has two greens. If the hidden code was red-red-blue-blue and the player guesses red-red-red-blue, the codemaker will award two black key pegs for the two correct reds, nothing for the third red as there is not a third red in the code, and a single black key peg for the blue.

Our goal in this assignment is to write programs for the codemaker and the codebreaker as outlined below. The state of the game will be maintained by the codebreaker.

- ***Developing the codemaker***: The codemaker will randomly choose the hidden code and store it in an array. Thereafter, in each iteration, it will invoke the codebreaker for a new guess. After the codebreaker makes a guess, the codemaker will compare the guess with the hidden code and provide the codebreaker the number of black and white key pegs awarded.

- ***Developing the codebreaker***. The black and white key pegs received in response to each guess enhances the knowledge of the codebreaker. A smart codebreaker will not make guesses, where one or more pegs are wrong based on its existing knowledge. For example, if the guess, red-red-red-red, received no key pegs, then it knows that there are no red pegs in the hidden code, and therefore no future guess should include a red peg. Therefore, a key requirement for a codebreaker is to find a guess that does not contradict its existing knowledge. We shall use a Boolean Satisfiability Solver (SAT-solver) to find guesses that are consistent with the existing knowledge.

Boolean SAT solvers check the satisfiability of logical formulas over Boolean variables. The first step in modelling a constraint satisfaction problem in SAT is to choose the variables modelling the system. Suppose we encode positions and colours as follows:

Positions: 1 (Left), 2, 3, 4 (Right)
Colour codes: R – red, B – blue, G – green, Y – yellow, O – orange, P – purple, W – white, K – black

Suppose the Boolean variable, X3G, represents a green peg in the 3rd position. There are 4 x 8 = 32 such variables, out of which only 4 will be true. If the hidden code is red-blue-red-green, then X1R, X2B, X3R, and X4G are true, and the rest are false. The codebreaker aims to deduce the four that are true and eliminate the others.

We can model various constraints using these variables. A few interesting ones are explained below.

- The code has one or more yellow pegs: $X1Y \vee X2Y \vee X3Y \vee X4Y$
- The code has a green peg, but not at the 3rd position: $X1G \vee X2G \vee X4G$
- We were awarded a white peg for a yellow peg in the 2nd position: $X1Y \vee X3Y \vee X4Y$
- The code has exactly one yellow peg:

$$[\, X1Y \wedge \neg\, (X2Y \vee X3Y \vee X4Y)\,]$$
$$\vee\, [\, X2Y \wedge \neg\, (X1Y \vee X3Y \vee X4Y)\,]$$
$$\vee\, [\, X3Y \wedge \neg\, (X1Y \vee X2Y \vee X4Y)\,]$$
$$\vee\, [\, X4Y \wedge \neg\, (X1Y \vee X2Y \vee X3Y)\,]$$

Sometimes we may use additional variables to implement more complex requirements. For example, consider the constraint: *we were awarded a white peg, either for the yellow peg in the 2nd position or the blue peg in the 3rd position.* We can encode this with 3 clauses:

- Clause-1: $Z1 \Leftrightarrow (\, X1Y \vee X3Y \vee X4Y\,)$
- Clause-2: $Z2 \Leftrightarrow (\, X1B \vee X2B \vee X4B\,)$
- Clause-3: $Z1 \oplus Z2$

SAT solvers require you to prepare the clauses in a standard format. We shall provide you a tutorial on how to use a SAT solver at the beck end.

The statement of the exercise is given in the next page.

Part 1: ***Codes without repetition of colours*** (20 marks)

In this part of the assignment the codemaker can only choose hidden codes without repetition of colours. Write the following:

1. A `main()` function which acts as the codemaker. It initially sets up the hidden code and shows it to us (we can see the code, because the computer will also act as the codebreaker).

2. In each iteration it invokes the codebreaker, evaluates it's score in terms of white and black key pegs, and shows us both the guess as well as the score. A sample output would look like the following:

	R G B Y		// Hidden Code
Iteration 1:	R W Y G	B-1, W-2	// Guess and score
Iteration 2:	Y W G O	W-2	// Guess and score
Iteration 3:	R G K Y	B-3	// Guess and score
Iteration 4:	R G B Y	B-4	

3. The codebreaker function, `codebreaker()`, which maintains a set of constraints on the variables and uses a SAT solver to find a new guess that does not have any contradictions with the clauses generated from the previous iterations. When it receives the scores (in terms of the white and black key pegs) from the codemaker, it generates suitable clauses and pushes them into the SAT solver.

Part 2: ***Codes with repetition of colours*** (10 marks)

Repeat the same exercise in the setting where the codemaker is allowed to choose hidden codes with repetition of colours. Also think of heuristics for choosing the next guess which has the potential to reduce the number of iterations. Can such a strategy allow some choices which are conflicting with previous knowledge?

Along with the codes for each part of the assignment you are required to submit a single PDF document describing the types of Boolean constraints added by the codebreaker and how they are encoded (with an example for each).