Practical 1: Elementary Search Algorithms

The precursor game of Chess was Chaturanga, dating from the 6th century in India. Chaturanga translates into "the four divisions": infantry, cavalry, elephantry and chariotry, which, already in modern chess, transpired into the four well-known pieces known in modern chess: pawn, knight, bishop and rook. Also, the game geometrical arrangement follows the typical battle lines, in which each army forms one battle line to confront each other. Although the game was first created by in India, chess was first incorporated to European culture through the Arabs during the Sassanid Empire and the Calafat of Cordoba.

Ever since, its derivative game, chess, became a popular game of strategy, in which search and anticipation are key elements for a potentially winning strategy. This became obvious when in 1997, an IBM computer beat wold champion Garry Kasparov. In the end, it is a matter of projecting the evolution of the game sufficiently into the future to be able to attain the best strategy.

The goal of this practical is to familiarize yourself with elementary chess, and with the basic search algorithms of AI. To this end, your task consists of implementing an AI search algorithm to attain specific goals on a chess board.

The Simulator
The simulator consists of a hierarchy of four classes: aichess, chess, board, piece. The latter three implement the dynamics of a chess game, which may be run by two players. Please, run the main on the class Chess to this end. Finally, the Aichess class is a capsule of the former three, with the purpose of implementing AI algorithms to analyse and alter the dynamics of the chess game.

Documentation
The python code provided is documented, and provides a straightforward structure for you to study and analyse. Although you do not have to understand every single detail, you need to build sufficient intuition about the structure of the underlying classes to be able to solve the problems listed next.

In this practical we will view chess as a game of search (of the check-mate board position starting from an origin position), which you will have to find out with a search algorithm. To facilitate your implementation work, the AIClass has several variables of interest, listed next:

- CurrentState. The list of positions and pieces of the White pieces on the board.
- ListNextStates. The list of states that hang from the current state in the tree.
- ListVisitedStates. The list of already visited states in the tree.
- PathToTarget. The sequence – python list of states from the origin to the target state yielded by your algorithm.

It is fundamental that you implement the **state** as a list of elements containing the positions and types of pieces you move. Each position has a numeric identifier you may find in the initialization of the board class. For example, the initial position could be: [[7,0,2], [7,4,6]].

This indicates that a white rook is in board position [7,0] and the king on position [7,4]. As you explore the decision tree, you will have to modify this state.

Finally, the Aiclass already has an implementation of the getListNextStateW(*thisState*) function, which returns the list of states you may potentially reach from *thisState.*

Your Work
During this first game, you are provided with a specific starting board configuration (pre-programmed): it contains a black king, on position [0,4] and two white pieces, the movement of which you will have to program: a king on position [7,4] and a rook on position [7,0]. Keep in mind that in this exercise, the black piece do not move, and that only you can move the white pieces. Your goal is to find the configuration state such that it is a check-mate to the black king.

To this end, you are instructed to find that state by implementing a basic algorithm. Please do provide a list of the visited states from the origin to the target (3p) and the minimal depth necessary to reach the target (1p).

1. A* Search. 3p

Now, initialize the board with the white king on position [7,7] and rerun your code.

2. Does your algorithm work in precisely the same way? Did you have to change anything? 3p