Paula Bezares
20393015
Eshaan Mittal
20620983

LAB 1: ELEMENTARY SEACH ALGORITHMS

INTRODUCTION

The first practical exercise in the field of Artificial Intelligence focuses on the implementation of the basic AI search algorithm known as A*. The objective of this exercise is to achieve a checkmate position using the white pieces in the game of chess. It is important to note that in this scenario, the black pieces are restricted from making any moves, and the initial chessboard configuration consists solely of both kings and a single white rook.

To accomplish this task, we will make use of four distinct classes that have been provided to us by the professors of the course. These classes include **aichess.py**, **chess.py**, **pieces.py**, and **board.py**. Our primary area of modification will be the **aichess.py** class, specifically within the *AStarSearch* method.

Through this practical, by focusing on the strategic game of chess, we will explore the essential concepts of artificial intelligence, problem representation, and algorithmic decision-making, through the successful implementation of the A* algorithm to navigate the chessboard and achieve checkmate.

WORK DONE

We initially started by using the given *BreadthFirstSearch* algorithm as a starting point and modified it so that the fundamentals of the A* algorithm are applied, such as the heuristics of each position. We start by adding the initial position into a PriorityQueue, called *frontera*, which contains all positions that we may potentially explore, and into the dictionary of visited states, saving the following values: g, h, parent, depth. G is the number of movements already made, H is the value from the current position to the end (checkmate), parent is the previous position of the board, and depth is the depth of the current state in the tree of all possible states. The next step of algorithm is to find all possible states that follow and add them to the priority queue based on their "f"[1] value, given that it is not a position that has already been previously explored. However, there is a case where we do add these previously visited states into the PriorityQueue, which is when the state now has a lower "f" value than that which was found before for the same state, causing an override of the given state as we have found a more optimal route to it. From there, we change the state of the board to the next position that has the smallest "f" value. Before repeating this for the next move, we check whether checkmate has been achieved or not, and if yes, whether there is still a position in the queue that could possibly provide a smaller "f" value to the solution.

[1] F is the value given by adding the cost to reach the current position (g) and the heuristic value/cost to reach solution from current position (h)

Paula Bezares
20393015
Eshaan Mittal
20620983

Once the algorithm has finished executing, the list of moves from start to finish is shown, along with the depth of the solution which in this case is 6. The path taken is:

[[[7, 0, 2], [7, 4, 6]], [[0, 0, 2], [7, 4, 6]], [[6, 3, 6], [0, 0, 2]], [[5, 2, 6], [0, 0, 2]],

[[4, 2, 6], [0, 0, 2]], [[3, 3, 6], [0, 0, 2]], [[2, 4, 6], [0, 0, 2]]].

CHANGE OF STARTING POSITION

For this question, we have changed the starting position of the white king from square [7,4] to the square [7,7]. After executing the code, we see that it also provides the correct solution without having to alter any code. However, it will obviously provide a different path to the solution but the same depth as before:

[[[7, 0, 2], [7, 7, 6]], [[0, 0, 2], [7, 7, 6]], [[6, 6, 6], [0, 0, 2]], [[5, 5, 6], [0, 0, 2]],

[[4, 4, 6], [0, 0, 2]], [[3, 3, 6], [0, 0, 2]], [[2, 4, 6], [0, 0, 2]]].

OBSERVATIONS & CHANGES MADE

During the project we made the following changes to the code. Our first change to the given code was to make the given list *frontera* into a PriorityQueue so that all addition to *frontera* would be automatically ordered from smallest to greatest given their “f” value. Meanwhile, a normal list would need to be reordered in every iteration of the algorithm. Furthermore, we provided a return value to the method *AStarSearch* so that it returns the depth of the solution, allowing the value to be printed in the main.

The most important change made to the code was adding a new method called *reorder_dict_keys*. This method has been created so that the string used to search through the dictionary will always have the format of [king, rook]. This change was made due to key errors being caused by the order of king and rook changing depending on which piece had moved.

We also noticed that the algorithm has the pieces move in a zig-zag pattern, which we see comes from the method *getListNextStatesW*. This is due to the order that the next possible moves are found, where it must be starting by finding the king moves from top left to top right, causing the king to move diagonally left in the beginning.

CONCLUSION

In conclusion, this practical exercise in the field of Artificial Intelligence has provided us with an opportunity to delve into the intricacies of AI search algorithms, particularly the A* algorithm, within the context of chess. By applying this algorithm to achieve checkmate with a limited chessboard configuration, we have gained valuable insights into problem representation and algorithmic decision-making. This exercise has showcased the effectiveness of AI techniques in addressing strategic challenges and highlighted the potential for further exploration in the realm of artificial intelligence.