Introduction to Artificial Intelligence (DS2020)

1. Name: DHRUVADEEP MALAKAR
Roll : 142201026
2. Name: DODDAPANENI UDITH
Roll : 142201012

<u>ConnectFour Game Implementation</u>

HEURISTIC / EVALUATION FUNCTION

The heuristic function (**func evl (s: str, player: int)**) takes two arguments.

1. ‘S’ represents the specific pattern in the board. These patterns can be vertical, horizontal or diagonal.
2. ‘Player’ takes an integer as its parameter which represents whether the player is the enemy or AI which we implemented.

The S takes input which are patterns which are identified by counting the number of occurrences of specific sequences of player pieces in the string ‘S’. For example, the string ‘00011110’ has the sequence ‘1111’ which represents four consecutive pieces of player 1 in a row, column or diagonal.

Evaluation Criteria

There are multiple different patterns such as ‘1111’, ‘1110’, ‘0111’, ‘0101’, etc which are given certain weightage such that ‘1111’ indicate an imminent victory so its assigned a high positive utility value of **1e9** while something like ‘0111’ or ‘1011’, etc are given a lower value than 1e9 of **43^2** then for pattern like ‘0011’ or ‘1001’ are given value of **43** each and single isolated node/piece is given a value of 1. We check if the player has any no. of instances with the pattern ‘1111’ then we automatically end the game. If not then we add the values of these patterns multiplying with the corresponding weightage.

Similarly, we now count for player 2 with patterns of ‘2’. When having four consecutive pieces of the opponent player ‘2222’ might indicate an imminent loss, so we assign it a very high negative heuristic value of **-1e9**. And similarly we calculate the patterns for player 2, and return the value for it accordingly.

Note: We are using player 1 and player 2 interchangeably, and when we call the evaluation function we are always referring to the given player.

Evaluation Function

For every move that we play as max player then as min player so we call evaluation function for every iteration and add that to the value of ai_player and player whom we are playing against. Also we check for patterns for all directions from row, column to diagonals.and at last return the value **ai_value - enemy_value** which will give us actual evaluation value.

The heuristic is biased towards recognizing specific patterns that are indicative of winning or losing positions. It does not consider all possible future moves or board configurations

TIME FUNCTION

The higher time that you can give the better the algorithm can perform and can generate more optimal and winning moves. The reason is for every move there is a branching factor of 7. Which increases the number of end nodes exponentially with a rate of **b^d** where b represents the branching factor and d represents the depth it traveled till now.

Tree Exploration with time

Knowing that the algorithm works in a time complexity of **O(b^m)** we can only explore a small part of the tree. The following is the number of nodes that can be traversed with limited time.

TIME	DEPTH LIMIT	TOTAL NODES EXPLORED
3 sec	**4**	**400**
5 sec	**5**	**2801**
10 sec	**5**	**2801**
40 sec	**6**	**19608**

AI VS AI

WITH OWN AI

While testing our code with its own AI against AI, we found out that AI makes the same kind of moves and at last either one of them wins or draws out completely, the AI or the machine. But in theory and also while testing we found out that the AI or machine who goes **first** has a higher probability of winning. While testing it multiple times over the course of lab, we found out that **100% probability** of winning if the ai goes first.

WITH OTHER PERSON'S AI
Also we tested our program with other classmates to test our AI, and overall when it goes first it has won almost all the times, while when it goes second, it draws the match around **30%** of the time and wins almost **70%** of the time.

WITH ONLINE AI
We have tested our program with online AI and found that if we go first using our AI it wins **10%** of the time and loses **90%** of the time. Hence the only way to actually win with 100% probability is by knowing all possible states it can reach from the start node which is expensively computative.