ARTIFICIAL INTELLIGENCE

PROGRAMMING ASSIGNMENT 2

RAJVI SAMPAT

2018B4A70820G

CONNECT4

Connect4 is a game in which a given board consists on n rows and m columns. When a coin is dropped into a column of the board, it falls into the lowest available row. Players take alternate turns in dropping coins into the board->move. If a player makes a straight line of 4 coins, either horizontal, vertical or diagonal, that player wins.

In the given program, the connect4 game is represented as a class with attributes board, player1, player2, winner and player (the current player). It has methods to initialize a new game, update the board, change the turn, find the winner etc. The board itself is a class consisting of the actual game board represented as a 2d numpy array and the number of rows filled in each column. The players are given values '1' and '-1' respectively as this simplifies the code to calculate the winner. If the game is still in progress, the winner is assigned value 0, and if there is a tie, the winner is assigned value 3.

MONTE CARLO SEARCH TREE

Monte Carlo Tree search is an efficient algorithm to solve games with large branching factors. Since the action space is very large, the computational complexity is brought down by building a search tree. It is an online method, and the agent learns while playing the game. It keeps a search tree whose nodes consist of states of the game. The value of a given node is calculated by using its Upper Confidence Bound Value.

$$UCB(n) = \frac{V(n)}{N(n)} + c * \sqrt{\frac{logN(Parent(n))}{N(n)}}$$

Where V(n) = Number of Wins after selecting the node

N(n) = Number of times the node is visited

N(parent(n)) = Number of times the parent is visited

C=Exploitation coefficient

The first term corresponds to the exploitation, i.e. takes the best values known. The second term gives the confidence bound. The larger the confidence bound, the more the exploration will take place. The tuneable parameter c controls the amount of exploitation vs exploration and is called the exploitation coefficient.

The algorithm performs many simulations of the game and keeps updating the values of the node. The algorithm has 4 steps, namely Selection, Expansion, Simulation and Backpropagation.

Selection: The best possible actions are chosen by following a path down the tree taking the node with maximum UCB value. This is done till a leaf node of the tree is reached

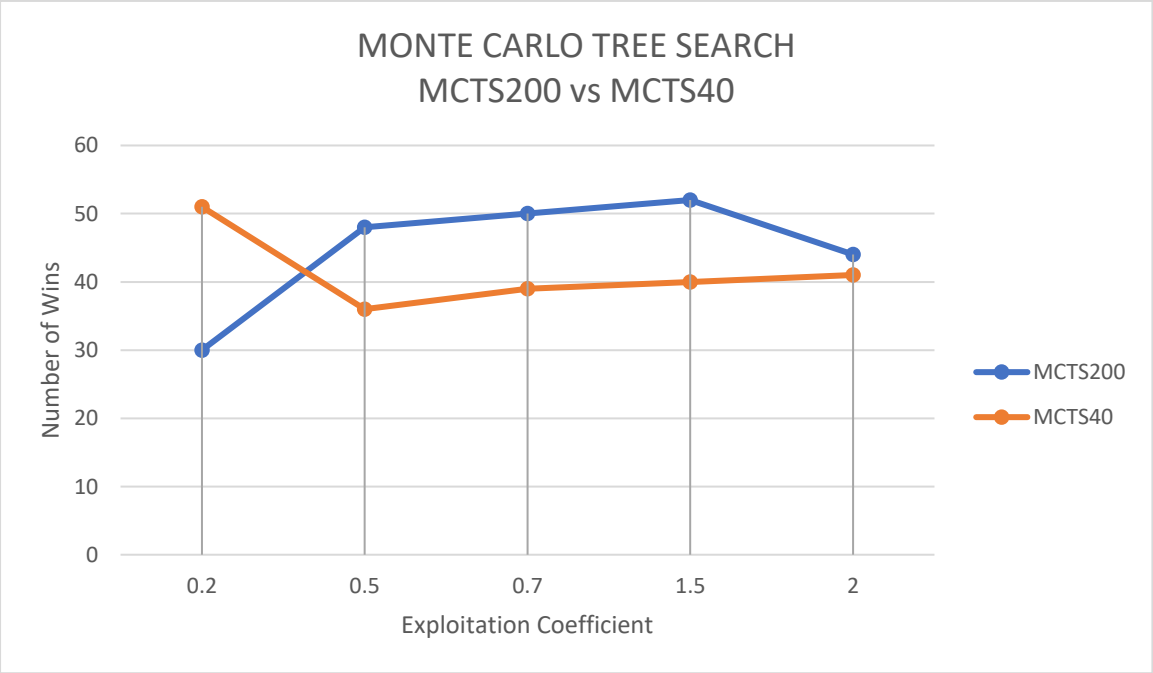
Expansion: The tree is then expanded by adding the child nodes of the current leaf node to the tree and then choosing a child node at random in the path. Here, the child nodes are those which can be found by adding a coin to one of the unfilled columns in the current state of the board.

Simulation: Once the tree is expanded, the algorithm simulates the rest of the game by choosing moves at random, without adding the obtained states to the tree. This is done till a terminal state is reached, ie either one of the players win or there is a tie.

Backpropagation: Depending on the result of the game, a reward is given to the agent. If it wins, the number of wins is incremented and if it loses, the number of wins is decremented so that a larger effect is obtained on the UCB value. This is done for all the nodes encountered in the path followed. Thus, the UCB value of a node is updated based on how it performs in the game and reinforcement learning is used to make the agent learn the best possible moves.

This process is repeated n times for an agent MCTS(n).

The game is then played between two such agents MCTS200 and MCTS40. The game is played for 100 iterations for a given value of 'c'. It is observed that for a smaller value of c, MCTS40 wins but a larger value of c favours MCTS200. This is because a larger value of c implies more exploration. Since MCTS200 has more simulations, it is able to run many iterations after exploring and get a more optimized UCB value for the nodes. However, since MCTS40 has lesser simulations, keeping more explorations may cause it to be more randomized as it can't make use of these explorations well. A value of c=1.5 is most beneficial. A larger value will cause too much exploration and randomize the process.



Q LEARNING

Q learning is an off policy Temporal Difference Control method. The Q values are updated independent of the policy being followed to select the actions. The algorithm is run offline, by first training the agent for many episodes. At each step, the agent chooses the action by following an epsilon-greedy approach. It takes a random possible move with probability<epsilon and the move with the best Q value with probability epsilon. The Q value is then updated based on the reward observed in the next state, the max q value of the next state.

In this program, a hash table stores the q values are stored in a dictionary which maps the hash of the state to the q value. The agent is trained against a MCTS agent with n=4 for 200 iterations. If the agent wins, a reward of 10 is given, if it loses, a reward of -10 is given and a reward of -5 is given for draw to ensure that the agent learns fast. Additionally, the q values are initialised to a random values in range (0,5) for non terminal nodes and to 0 for terminal nodes.