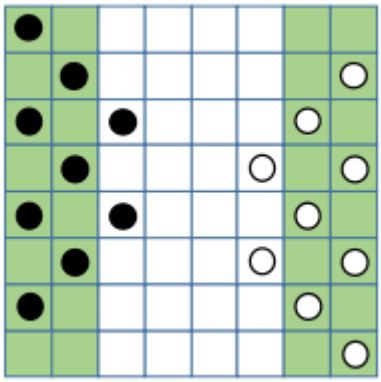
# Introduction to AI – Group Game Project

## I. Abstract

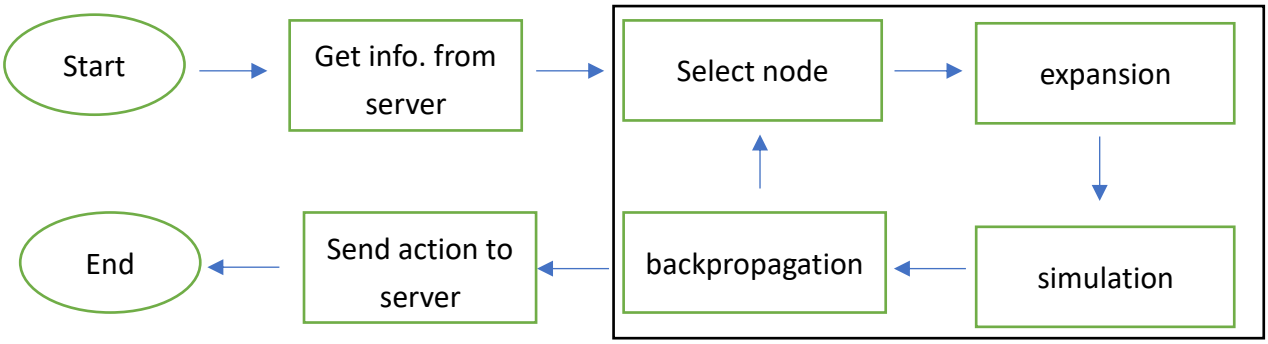


For this project, we set out to design an agent mastering a Chinese Checker like board game. Each team have 9 pieces, the goal is to hop to the 2 columns opposite to the starting side. When hopping over the enemy's piece, enemy's one will be excluded. If all the piece of one team reach the target area or already plays for 200 rounds, the game is ended.

## II. Hypothesis

We first come up with serval good proven to work strategies, handcraft rules, MCTS, TD learning and AlphaGo Zero. Then we found that given the computation power limit, Alphago Zero is out of the game. Then we look into MCTS because of its similarity to AlphaGo. Then we also found TD learning as an alternative, so we started to try MCTS and TD learning.

## III. Program Architecture and Flow Chart




MCTS

## IV. Program Approach

We write four classes and several functions to implement the agent.

| Class: Board | |
|---|---|
| moveOneStep | First check it is playing for which role(black/white) and provide three different ways to move - move up/down/right/left, hop over the piece of the enemy, hop over our piece. |

| | |
|---|---|
| performMove | Do a series of moveOneStep and check it is legal or not. If it is illegal, it would return back to previous step. |
| getBoardValues | Get the evaluation list of the board. |
| setBoardValues | Set a list to evaluate each piece value on the board. |
| checkInRegion | Check the pieces are in the target region. There are three cases - all of our pieces on the board reach the target region and all the enemy's pieces are eliminated, all of our pieces on the board reach the target region and there are enemy's pieces, not all of our/enemy's piece reach the region. |
| checkStatus | Check the status. There are four status - Our win, enemy win, draw, the game is still going. |

| Class: State | |
|---|---|
| getBoard | Return board state. |
| getPlayerNo | Return player number. |
| setPlayerNo | Set player number. |
| getOpponent | Return opponent number = 3 - player number.<br>1: black 2: white. |
| getVisitCount | Return visit count. |
| setVisitCount | Set visit count. |
| getWinScore | Return win score. |
| setWinScore | Set win score. |
| posPossibleState | Expand all the possible moving state and record it in the list. |
| getAllPossibleStates | Return the list that includes all the possible move for this step to next step and update the Board, PlayerNo, VisitCount, WinScore. |
| incrementVisit | Increase visit number by 1. |

| | |
|---|---|
| addScore | Add score to win score. |
| wuEvaluatePlay | Provide an evaluation function and return a score according to this state. If enemy's piece reaches the target region: -5. If our piece reaches the target region: +5. If kill the enemy's piece: +10. If our piece is killed: -10. |

| Class: Node | |
|---|---|
| getState | Return object state class (Class State). |
| setState | Set state to given state class (Class State). |
| getChild | Get list of child node classes (List of Class Node). |
| setChild | Set list of child node classes (List of Class Node). |
| getParent | Get parent node class (Class Node). |
| setParent | Set parent node class (Class Node). |
| getRandomChildNode | Get the all possible state and randomly select one child node to return. |

| Class: Tree | |
|---|---|
| getRoot | Return root node (Class Node) class. |
| setRoot | Set root node class as given (Class Node) node. |

| Funtion | detail |
|---|---|
| uctValue | (nodeWinScore/nodeVisit)+1.41*math.sqrt(math.log(totalVisit) / nodeVisit ) |
| findBestNodeWithUCT | Check out all the child node and find out the best one by using the uctValue. |
| selectPromisingNode | Find the expanding node. |

| | |
|---|---|
| backPropogation | Using backpropogation method to update the parent node from the simulation child node. |
| checkChess | Check the location of the piece. |
| displayChildInfo | Print out the score and number of the child node. |
| findNextMove | Run the MCTS algorithm - Selection, Expansion, Simulation, Backpropogation until the time is out. Return the best move. |
| oppoentMove | match the enemy's move and get into the corresponding child node. |
| GetStep | Check the role(black/white) and give different parameter to do findNextMove. |
| main | Get the information sent by the server and run Getstep and SendStep or oppoentMove by turns. |

## V. Discussion/Experiment

We design several playing strategies to help us to check this method is good or not.

1. Defense strategy: Back all the piece to the last position and wait for the enemy's piece coming. If the enemy's piece gets into the target area, hop over and eliminate it.

Our MCTS can move two pieces next to each and get into the target area, so it won't be eliminated by the enemy. Or another case is to consume the enemy's piece to let it cannot defense the area well, and then we can move the piece into the target area.

2. Stable strategy: Do not move the piece until the enemy's piece reach the dangerous zone. If the enemy's piece gets into the dangerous zone, try to eliminate it.

Our MCTS can destroy the formation and move our piece across the weak area. Then we can move the piece into the target area.

3. Attack strategy: Rush the piece to the target area, when the enemy's piece in the dangerous zone, do not move the piece until the enemy's come and eliminate it.

Our MCTS can move two pieces next to each and get into the dangerous area, so it won't be eliminated by the enemy. It will also destroy the formation and our piece can move across the weak area.

## VI. Result

Show the step when playing the games.

```
step 13 ,it's turn to 1
~  0  1  2  3  4  5  6  7
0  O  -  -  -  -  -  -  -
1  X  -  -  -  -  -  -  O
2  X  -  -  -  -  O  -  O
3  X  X  -  -  -  O  -  -
4  -  -  -  -  -  -  -  -
5  -  O  -  -  -  -  -  -
6  -  -  -  -  -  -  O  -
7  -  -  -  -  -  -  -  O
step 13 ,it's turn to 2
~  0  1  2  3  4  5  6  7
0  O  -  -  -  -  -  -  -
1  X  -  -  -  -  -  -  O
2  X  -  -  -  -  -  O  O
3  X  X  -  -  -  O  -  -
4  -  -  -  -  -  -  -  -
5  -  O  -  -  -  -  -  -
6  -  -  -  -  -  -  O  -
7  -  -  -  -  -  -  -  O
```