Artificial Intelligence – Project 2 Tic Tac Toe - Report

[Pranay Patodi] – [G22035334]

[Siva Teja Koppineedi] – [G41333490]

[Jeet Shah] – [G41262537]

[Ashvi Soni] – [G38103241]

Work Distribution

Name	Task Performed
Pranay Patodi	• Created initial structure of code • Implemented minimax function • Implemented Alpha Beta pruning • Implemented cache function • Worked on optimising code and runtime of code • Did R&D on evaluation function, heuristic functions. • Played games with opponent teams • Worked on project report • Testing
Siva Teja K	• Developed API's for communicating with server • Developed Evaluation Function(Win check) • Developed heuristic function(get legal moves) • Worked on optimising the code structure • reduced time complexity of minimax function • Monitoring the games while playing with opponents with the team • Worked on Project Report • Testing
Jeet Shah	• Worked on initial structure of code • Implemented depth condition • Worked on research and development of heuristic function(get legal moves) and Win Check function • Worked on optimising code structure • Played games with other teams • Worked on project report • Testing
Ashvi Soni	• Brainstormed possible implementation methodologies • Worked on initial structure of code • Worked on research of evaluation function of the code • Worked on optimizing code

	• Worked on refactoring of the code • Monitoring the games while playing against opponents • Worked on project report • Testing

Files :

game_engine.py
This file contains our main function which is used to play games. It contains a while loop which is used to run the script until someone wins or the game is drawn. It contains functions to get response from api and convert it into json format.

bestMove.py
This file is used to predict the best move that is the row and col indexes where we should place our marker. It is used to call minimax.py

minimax.py
This file is our main file which contains our heuristic function, legal move function and minimax with alpha beta pruning with cache implementation function.

wincheck.py
This file contains the code by which we can find if the game is win or draw. It checks in row, col , diagonal and reverse diagonal elements.

API
This folder contains api files to create game, get board, make a move and get a move from server.

Agent :

Minimax Algorithm with alpha beta pruning :
Minimax is a recursive algorithm which is used to choose an optimal move for a player assuming that the opponent is also playing optimally. Alpha beta pruning allows us to search much faster and even go into deeper levels in the game tree. It cuts off branches in the game tree which need not be searched because there already exists a better move available.

Alpha is the best value that the maximizer currently can guarantee at that level or above.Beta is the best value that the minimizer currently can guarantee at that level or above.

To optimize the evaluation further, we have also included depth condition. Which means a node will not be explored after a certain depth.

Evaluation Function:

We have prioritized nodes to explore in three ways. First, it will check if an opponent has two or more consecutive markers in any rows, columns and diagonals. If there exists such a condition, then we will add those empty positions to legal moves and return. If such a condition doesn't exist, then we will play aggressively by checking if the same condition exists for our team or not. If it exists then we will add those empty positions to legal moves and return. And if that condition also doesn't exist then we will check the opponent's markers position and check other, upto eight, positions and then empty positions will be added to legal moves. If the first move is ours we will pass our marker at 6,6 coordinate.

Runtime:

For the first move we are giving a fixed position on the board and after that we are getting board from the server and passing the board to get the best move from our algorithm. We are finding the legal moves that are the empty position nearby the "X" and "O" marker. Once we get all the legal moves then we use our heuristic function to find best positions and then we are passing those best positions to the minimax algorithm which returns the best score value of the corresponding best position.

```
Best Legal Moves : [[3 6]
 [5 6]
 [5 3]
 [4 6]
 [4 3]]
Execution time : 5.281316757202148
5 6
{"moveId":63395,"code":"OK"}

[['_' '_' '_' '_' '_' '_' '_' '_' '_' '_' '_' '_' '_']
 ['_' '_' '0' '_' '_' '_' '_' '0' '_' '_' '_' '_' '_']
 ['_' '_' '_' 'X' '0' '0' 'X' '_' '_' '_' '_' '_' '_']
 ['_' '_' '_' '0' 'X' 'X' '_' '_' '_' '_' '_' '_' '_']
 ['_' '_' 'X' '_' 'X' 'X' '_' '_' '_' '_' '_' '_' '_']
 ['_' '_' '_' '_' '0' 'X' '0' '_' '_' '_' '_' '_' '_']
 ['_' '_' '_' '_' '_' 'X' '0' '_' '_' '_' '_' '_' '_']
 ['_' '_' '_' '_' '_' '0' '_' '0' '_' '_' '_' '_' '_']
 ['_' '_' '_' '_' '_' '_' '_' '_' '_' '_' '_' '_' '_']
 ['_' '_' '_' '_' '_' '_' '_' '_' '_' '_' '_' '_' '_']
 ['_' '_' '_' '_' '_' '_' '_' '_' '_' '_' '_' '_' '_']
 ['_' '_' '_' '_' '_' '_' '_' '_' '_' '_' '_' '_' '_']]
```

How to run the game :

Create Game :
To create game you need to run “API/CreateGameAPI.py”. In this api you need to pass opponent team id with board size and target length which is the number of consecutive marker.

Play Game:
To play game you need to run “game_engine.py”. In this file you need pass game id, opponent team id and run the script. It will execute until someone wins or its a draw.

```
# Only calculate a score if the board is not already in our cache.
def minimax_score_with_cache(board, depth, ism, alpha, beta, target_len,team_A,team_B):
    board_cache_key = str(board)
    if board_cache_key not in cache:
        best = minimax(board, depth, ism, alpha, beta, target_len,team_A,team_B)
        cache[board_cache_key] = best
    return cache[board_cache_key]
```

```
# This is our Heuristic Function...
def get_legal_moves(board,team_A,team_B):
    #this will return the possible opponent moves where he has already 2 elements in a row,column or diagonal
    x_y = get_legal_moves_from_opponent(board,team_B)
    x_y2 = get_legal_moves_from_opponent(board, team_A)
    x_y.update(x_y2)
    if x_y=={}:
            # this will return the possible  moves of team_A where its neigbouring elements are  's
            il,jl=np.where(board==team_A)
            for index in range(0, len(il)):

  if (ism):

      # Maximising move
      best = -1000
      for i in range(0, len(legal_moves)):
          curr_move = legal_moves[i]
          board[curr_move[0], curr_move[1]] = team_A
          best = max(best, minimax(board, depth + 1, not ism, alpha, beta,target_len,team_A,team_B))
          if (depth > 9):  # Depth heuiristic
              return best
          alpha = max(alpha, best)
          board[curr_move[0], curr_move[1]] = '_'
          if (alpha >= beta):
              break
      return best
  else:

      # Minimising opponent move
      best = 1000
      for i in range(0, len(legal_moves)):
          curr_move = legal_moves[i]
          board[curr_move[0], curr_move[1]] = team_B
          best = min(best, minimax(board, depth + 1, not ism, alpha, beta,target_len, team_A,team_B))
          if (depth > 9):  # Depth heuiristic
              return best
          beta = min(beta, best)
          board[curr_move[0], curr_move[1]] = '_'
          if (alpha >= beta):
              break
      return best
```