DRAFT OF AI Colosseum Survival GAME

Algorithms ,Functions and Data structures used

1. **Adjacency list**
2. **Directed graph**
3. **Heuristic evaluation Function**
4. **Breadth first search algorithm**
5. **Min max algorithm**

Firstly,We get an Adjacency list of all the available points that are found within the maximum number of steps a player is allowed to reach,we get the list for both the Adversary and our student agent,this permits us to have a perfect state of the game .

So now have a road map of where all the agents can go to,I now save them in 2 lists enemy_moves and my_moves.Using the length of this lists,I now create my heuristic function.

Heuristic evaluation_Function:

This function is devised by the lengths of the different lists which contain the moves of the agents.So we have three states which are winning ,loosing and Draw.

The agent with the highest number of moves is Winning ,while the user with the lowest number of moves is loosing.The third state now comes in when both users have equal number of steps.

We will use a minmax algorithm to determine our best move,since we already have an idea of who is winning and who is loosing.Before implementing the algorithm ,we first have to analyse our best move using a graph structure.So we now pass our Adjacency list in a directed graph tree and using a Breadth first search algorithm,we get all the future moves of the nodes we have considering the barrier positions to be placed if we take that move and we store it in our dictionary all_lengths,now for all the future moves of each nodes, we make a nested dictionary(last_moves) that will contain every node and its future moves,taking into consideration its barrier position to be placed.

We now have all the possible future moves our agent can make,using our Heuristic evaluation Function,we can now know if we are winning or loosing.If we are winning,we get our number of moves and compare to our dictionary containing our future number of moves.for the highest move in our future number of moves that doesn't have 2 or more barriers attached to that point,we choose that move and if we are loosing,we compare our future moves with that of our adversary and choose the highest number of moves.

The idea here is that we want to always be at a position favouring our agent,that is our agent should always be winning.For some issues beyond our control, our algorithm is not able to locate some barriers on the board and then chooses a random move,also when our agent does found any suitable move,we make our agent to make a random move.

So our agent is always on the defence rather than on offence,it is always looking for a way to escape to the route that has the highest number of moves and henceforth win.

When we run the function –autoplay ,our agent is able to beat the adversary with a score of at least :

INFO:Player A win percentage: 0.976 (0.07534 seconds/game)

INFO:Player B win percentage: 0.029, (0.00133 seconds/game)

Limitations and improvements:

Though our agent has an almost perfect score against the random agent,it can be improved ,below are some limitations and possible improvements which can be made:

- Takes more time to play :
    - Since it has a lot of processes going on,it can be improved using threads so as to make the processes run in the background.
- Always on the defense and not on the offence:
    - Our agent is constantly looking for better moves,but then it can be improved by getting the best next move of the opponent and blocking its agent.This can be done using a depth first or Breadth first search on its moves.
- Uncertain Future of adversity:
    - Comes back to the point that our player is always on the defense rather than offense,so our player is always analyzing his moves only.So we can improve our step function by analyzing the game play of our adversary and get a better move to play.

Nevertheless, an implementation of the α-β pruning algorithm will be quite favourable,allowing us to dicard the worst moves our agent has and focus on the good ones hence increasing our chances of winning.