MULTIPLAYER NIM AI GAME-ENGINE

Artificial Intelligence (CSL3090) Course Project

Group Members:

- Vikash Yadav (B20AI061)
- Swara Patel (B20AI044)
- Shubham Solanki (B20AI040)
- Harshith Reddy (B20AI018)

Link to GitHub Repository: **https://github.com/vcos611/nim-py**

▼ Abstract

- This is a small work in direction of Artificial Intelligence to beat multiplayer games.
- This project aims to extend the Minimax approach to multiplayer (more than 2 players) games. Minimax cannot be directly applied to multiplayer games, hence a new approach MaxN is introduced which tries to maximize the utility for current player at each layer. Another algorithm i.e. Paranoid algorithm which is based on the paranoid assumption that all other players are collectively playing against the current player hence it kind of transforms the MaxN approach to Minimax, where we treat all other players as a single opponent.
- Building the entire Search Space tree for any of these algos isn't feasible as it grows exponentially with depth. Hence we need a heuristic to evaluate intermediate states. Next, we introduce pruning in the Paranoid approach which is same as that in Minimax approach. Pruning cannot be performed in MaxN approach.

- Furthermore, we discuss another approach i.e. MP-Mix wherein algorithm switches between MaxN, Paranoid and Offensive strategy based on the the situation of the game. Offensive strategy means playing as to reduce the winning rate of a target player. This results in our agent playing offensively or defensively as needed and hence a more robust model comes out.
- We take the example of NIM game to explain all the above mentioned algorithms

▼ Brief Problem Statement

- The goal here is to develop AI agent for intelligently playing the multiplayer NIM game i.e. a number of piles with different number of sticks is given; players take turns and remove any number of sticks from a specific pile; the player to remove the last stick wins.
- For a 2-player NIM game, a simple MiniMax approach with Alpha-Beta Pruning would have worked perfectly, but here our problem is to extend that approach for multiple players. MaxN approach is based out of Minimax approach, but the problem with it is that pruning is not applicable with this approach; hence the need to find a good heuristic.
- Furthermore, there are situations where it is more appropriate to follow the MaxN strategy, while on other occasions the Paranoid strategy might seem to be the appropriate one; hence we explore another algorithm i.e. MPmix which switches between MaxN, Paranoid and Offensive strategy based on the situation of the game via value of defensive_threshold and offensive_threshold. Again, getting the appropriate values for threshold is a problem in itself.

▼ Literature Survey

- We refered [2] for implementing MaxN algorithm. Further we went through [3] to learn about pruning in multiplayer approaches and came to a conclusion that pruning only works well for minimax approach i.e. 2 player games. Hence pruning also works for Paranoid approach.
- Further most of our work is based on [1] which explains MPmix algorithm that tries to overcome disadvantages of each individual approach and switch to any of MaxN, Paranoid or Offensive based on which approach is most beneficial according to current state of the game.
- [4] is an extensive work in this direction of AI, reading it's introduction gave us more motivation and understanding of this problem.

▼ Motivation

- Game Playing has been a popular area of interest for the researchers in AI. It is a one of the many forms of trying to teach / program the computer to the things that humans do. The artificial intelligence technology used for such systems is becoming more and more important in solving real life problems too. Here we are limiting our discussion to adversarial games.
- Starting off with 2 player games, much research has gone into it and the Minimax algorithm with Alpha-Beta Pruning is wonderful. But the limitation of it being limited to single adversary hugely

reduces it's scope of usability. That is because in most real life scenarios, there are multiple adversaries. It may seem small difference but 2 person situations and situations involving 3 or more players have a qualitative difference which isn't so obvious as going from 2 to 3.

- Once the game has 3 or more players in it, it is no longer the case of strongest-person-winning. The other players can work in coalition to remove this strong threat to all. Such dependencies change the overall dynamics of the games.
- Thinking of strategies for solving multiplayers games such as NIM is a nice first step towards such problems. Many of the results and inferences we draw here are applicable to real life multiple adversary cases.

▼ Methodology

- The game is more or less just making the decisions about which row to pick the sticks from and how many sticks to pick. Any other game based on making decisions where the domain of decisions is defined can be played using this AI with just slight modifications.
- The choice of the decision rule is implemented by the algorithm which searches the game tree. The standard two-player decision rule is minimax and the standard and clever way to fasten it is alpha-beta pruning.
- Here for multiplayer we consider the following decision algorithms derived from the standard minimax:

`Here for each node the evaluation function returns a vector H of n values where H[i] estimates the merit of player i`

▼ The MAXN algorithm

This algo works based on the assumption that all the players make decisions just to greedily maximize their own chance of winning without giving any concern to that state of other players or how close other players are to winning.

So here from the leaf nodes (that is the winning states) the tree is built in a bottom up way where the evaluation at each node is chosen from one of the child nodes which has the maximum chance of winning for current player (the player whose turn it is). `[Out of all H vectors of all children, the child with maximum H[turn] is chosen.]`

Values are propagated in max fashion at every level of the search tree.

▼ The Paranoid algorithm

This algo works based on the the assumption that all players other than the player whose turn it is are forming a coalition to defeat the current player.

So the levels at which **turn is of the root player** the child with the maximum evaluation for the root player is chosen `[Out of all H vectors of all children, the one with maximum H[root_turn] is chosen].`

And on the levels at which **turn is of other players** (not the root player) the child with the minimum evaluation for the root player is chosen `[Out of all H vectors of all children, the one with minimum H[root_turn] is chosen].`

Values are propagated in max fashion for turns of the root player and in min fashion for the turns of other players.

Here we are only concerned with H[root_player], so instead of H vector we just work with a single value, as all the other H[i] are not needed.

This algorithm is just like minimax and treating all other players combined as a single opponent. As a result, here alpha-beta pruning is applicable which allows faster and hence deeper searches of the tree.

▼ The Offensive Strategy

This is a building step for the MP-MIX algo. This isn't used generally for a player for the entire game though it can be used.

This strategy assumes that all other players play to optimize their individual chances while the root player plays in order to reduce the chances of one specific oppnent which is our target for offense.

Let's say root_player has to go offensive against a player x, then at all levels of other players (not root_player), maxn works `[Out of all H vectors, the one with maximum H[turn] is chosen] but for the levels of the root player the child with the minimum evaluation for the target player is chosen. [Out of all H vectors, the one with minimum H[x] is chosen]`.

▼ Mixing the algorithms: The MP-MIX algorithm

- All the above discussed algorithms have a fixed strategy, but following just one of these strategies throughout the entire game is not a very wise deicision. Hence we introduce this MP-mix approach where we change the strategy for decision based on the situation of the game.
- The MAXN should be the normal operating startegy for our agent when the game starts or when no single player is in a strong position, because of the assumption that everyone's goal is to win the game.
- When a our agent leading the game, it should make moves considering the fact that all other players will combinedly try to hinder its winning chance, so for a leading player, PARANOID startegy might be a wiser decision.
- When one of the player is in a very strong position, our agent will go offensive against it and play in the OFFENSIVE strategy so that the strong player does not win leaving room for our agent to win in the future.

How to mix?

- At each decision, first we find the evaluation for all the players. We then check if the agent is leading the game or not. If the agent is leading the game and leading by an amount > some threshold then agent should deploy PARANOID approach. If the agent is not leading and some other player is leading and leading by and amount > some other threshold, then

our agent should go OFFENSIVE against the leading player. If none of these conditions hold, then agent should play via MAXN approach considering that all players want to win.

- To check how much is the leading player leading, we see the difference in the heuristic values of the leading and the second leading player. This difference provides the most important information about the game's dynamics - a point where one leading player is too strong. To justify this, consider a situation wherethe leading edge between the first two players is rather small,but they both lead the other opponents by a large margin.This situation does not yet require explicit offensive movestowards one of the leaders, since they can still weaken eachother in their own struggle for victory, while, at the same time,the weaker players can narrow the gap.
- We determined the values of these 2 thresholds via simple trial and error procedure, though machine learning algorithms can be applied to find the same.Decreasing these thresholds will yield a player that is more sensitive to the game's dynam-ics and reacts by changing its search strategy more often.

▼ Pruning the Search Tree

- Going to the leaf node of the search tree and then backtracking each time is simply not feasible as the nodes expand exponentially, hence there is a need of some heuristic function that would return a score pertaining to some intermediate state of the game and thus the whole tree need not be traversed.
- Apart from this, alpha beta pruning cannot be applied on MaxN algorithm as multiplayer game is not zero-sum game instead it is constant-sum game. There exists some other pruning methods however we have not considered them.
- We have applied alpha beta pruning for the Paranoid agent since it is similar to the Minimax approach i.e. used for 2-player zero-sum game. Since Paranoid agent assumes that all other players except the current form a coalition against him and treats them altogether as a single player, the game is converted to a 2-player zero-sum game and hence alpha beta pruning can be applied here.

▼ Heuristics

There are 2 heuristics we used: (We devised the second one)

▼ Heuristic-1

We count the minimum number of moves in which this game can end minv (that is the number of non-zero piles) and the maxium number of moves in which this game can end maxv (that is the total number of sticks).

Then in the range minv to maxv we count the number of number_of_moves_to_end_the_game where player i would win.

And this count divided by the total number of number_of_moves_to_end_the_game is our heuristic for that player.

▼ Heuristic-2

The above heauristic is based on the assumption that equal number of games end in all number_of_moves_to_end_the_game which isn't true. So in this heuristic we assume that to be a logarithmic distribution rather than a uniform one. Means we assume that the number of games that would end in a given number of moves are related logarithmically. The weights of this logarithmic mapping we found using dry running on case and fitting a logarithmic curve to the points.

▼ Growing Depth Strategy

- Taking max_depth to be some high value as 7 or 9 takes a long amount of time for the agent to make a decision, though that decision is good but still computationally expensive. On the other hand, if we take max_depth to be as low as 3, it will lead to degradation in the performance of AI agent as it will not be able to go deep down and find the actual path which is performing well. Heuristic helps in some extent, however it is not perfect.
- Here the growing depth strategy finds a sweet spot between both of these, i.e. initially when number of stick is comparitively more, we start with max_depth = 3 and prune the tree accordingly. Now as the game proceeds further and number of sticks reduce, we increase max_depth a little bit, and then pruning takes place accordingly.
- This strategy works well as initially when number of sticks is comparitively more, takng the correct decision does not make much sense. Once the number of sticks is fewer, now if max_depth is increased, one can reach the leaf node and ultimately choose the best path hence the outcome.

▼ Experiment Settings

1. First, we simulated 2 player games for 200 times where each opponent took turns in being first i.e. Opponent 1 played first for 100 times and Opponent 2 played first for next 100 times. The simulations were performed between:
   - Random Agent vs MaxN
   - Random Agent vs Paranoid
   - Random Agent vs MPmix
   - MaxN vs Paranoid
   - MaxN vs MPmix
   - Paranoid vs MPmix

   The hyperparameters for each of the engines were set to default values i.e.

- max_depth = 5 for each engine
- pruning = True for Paranoid engine
- thresh_off = 0.3 and thresh_def = 0.3 for MPmix

The significance of running these simulations was to check correctness of implemented algorithms

2. Next, we simulated the game among 3 Random Agents and 1 AI Engine for MaxN and Paranoid to get the optimal value of depth. We ran it 100 times for both the engines for max_depth in [3,5,7].

   Further, we also ran the simulation 100 times with grow_depth = 1 for both the engines which implements our novel algorithm of increasing depth with decrease in number of sticks so as to reduce time complexity.

3. Next, we simulated the game 100 times among 3 Random Agents vs MPmix with default settings for other hyperparams for each of the 9 possible combinations of thresh_def and thresh_off so as to choose the optimal one. Values for both of the thresholds belonged to [0.2,0.3,0.4].
4. After getting the optimal threshold values for MPmix, we again ran simulation 100 times among 3 Random Agents and a MPmix engines with threshold values set to the those obtained above for choosing the optimal values of max_depth from [3,5,7]. Same simulation was also performed for grow_depth=1 100 times.
5. After getting optimal values for all the hyperparams for each engines through above simulations, now we ran simulation among 3 Random Agents and 1 AI Engine i.e. Paranoid, MaxN or MPmix with a different heuristic function.
6. Finally, after deciding the best heuristic function we ran the simulation among:
   a. 3 MaxN with its best heuristic function vs 1 MPmix with its best heuristic function
   b. 3 Paranoid with its best heuristic function vs 1 MPmix with its best heuristic function
   c. 2 MaxN with worst heuristic vs 1 MaxN with best heuristic
   d. 1 MaxN with best heuristic vs 1 MaxN with worst heuristic vs Random Agent

   The purpose of this experiment is
   - to identify if MPmix outperforms MaxN and Paranoid engines
   - the better heuristic is actually better or not

▼ Results

Abbreviations Used-

- R: Random Agent
- P: ParaNoid
- M: MaxN
- X: MPmix

A row corresponds to a game with number of columns equal to number of players/engines.

1. 2-player games among different agents with default settings
   Below tables show games won by each agent(out of 200)

D	P
4	196

P	M
5	195

D	X
0	200

M	X
71	129

D	M
1	199

P	X
0	200

2. Varying max_depth from 3 to 7 and setting grow_depth=1 for MaxN and Paranoid
   Below tables show games won by each agent(out of 100)

max_depth	R	R	R	M
3	3	13	35	49
5	7	11	32	50
7	1	19	33	47
grow_depth=1	7	14	23	56

max_depth	R	R	R	P
3	14	38	11	37
5	9	26	28	37
7	7	19	19	55
9	4	20	22	54
grow_depth=1	11	23	36	30

   In above table, we checked for max_depth = 9 also, because, we wanted to test the pessimist nature of Paranoid agent that increases with max_depth

3. Varying thresh_def and thresh_off from 0.2 to 0.4 and calculating results for all possible combinations for MPmix agent
   Below tables show games won by each agent(out of 100)

thresh_def	thresh_off	R	R	R	X
0.2	**0.2**	3	14	28	55
0.2	**0.3**	4	14	21	61
0.2	**0.4**	3	15	42	40
0.3	**0.2**	5	16	37	42

thresh_def	thresh_off	R	R	R	X
0.3	**0.3**	8	18	27	47
0.3	**0.4**	8	13	23	56
0.4	**0.2**	4	17	29	50
0.4	**0.3**	6	10	33	51
0.4	**0.4**	5	18	23	54

4. Varying max_depth from 3 to 7 and setting grow_depth=1 for MPmix
   Below tables show games won by each agent(out of 100)

max_depth	R	R	R	X
3	4	15	37	44
5	5	9	33	53
7	4	10	30	56
grow_depth=1	4	16	28	52

5. Varying heuristic functions for each agent i.e. MaxN, Paranoid and MPmix with best settings obtained from above tables i.e.
   MaxN: grow_depth=1
   Paranoid: max_depth = 7
   MPmix: thresh_def = 0.2, thresh_off = 0.3, max_depth = 7
   Below tables show games won by each agent(out of 100)

heuristic	R	R	R	M
1	9	17	36	38
2	2	19	24	55

heuristic	R	R	R	P
1	7	13	28	52
2	8	19	24	49

heuristic	R	R	R	X
1	1	20	26	53
2	8	9	30	53

6. Challenging various agents with their best heuristic function
   Below tables show games won by each agent(out of 100)

M2	M2	X1	M2
26	11	38	25

P1	P1	X1	P1
8	27	54	11

M1	M1	M2
42	10	48

M2	M1	R
72	18	10

Conclusions

1. When opponent is Random Agent, all 3 AI agents i.e MaxN, Paranoid and MPmix perform extremely well, i.e. they barely lose; which implies that our agents work correctly.
   Next, MaxN outperforms Paranoid and MPmix outperforms MaxN; which implies that MPmix is the best all of 3 agents and Paranoid as an individual agent is the worst.
2. It was thought that as max_depth is increased i.e. the agent would perform even better but from results it is evident that for MaxN, it performs significantly better at max_depth = 5 than max_depth = 3. On increasing max_depth further, no significant improovement is observed.
   However, the grow_depth setting worked even better for MaxN algorithm (increasing max_depth based on number of total sticks).
   Similar thing is evident for Paranoid agent for max_depth = 7 after which no significant improvement is noticed.
3. On trying different values of thresholds for MPmix agent, it is evident from results that the combination where thresh_def = 0.2 and thresh_off = 0.3 outperforms all other combos.
4. On varying max_depth for MPmix agent with threshold set to the ones mentioned above, no significant improvements are evident after max_depth = 7.
5. It was thought that heuristic 2 would significantly outperform heuristic 1 since it takes into consideration the distribution i.e. number of games corresponding to each value, however, from the results it is evident that heuristic 2 performed well only for MaxN agent. Its performance dropped for Paranoid agent and there was no change observed for MPmix agent.
6. Next, MPmix agent was simulated against 3 MaxN and 3 Paranoid agents and it won each time, indicating that switching among the agents based on the cuurrent state of the game leads to a better agent.
   Also, MaxN was simulated against another MaxN where in both had different heuristic function. Here the one with the better heuristic i.e. heuristic 2 won.

▼ References

[1] https://www.researchgate.net/publication/220814706_Mixing_Search_Strategies_for_Multi-Player_Games

[2] https://www.aaai.org/Papers/AAAI/1986/AAAI86-025.pdf

[3] https://faculty.cc.gatech.edu/~thad/6601-gradAI-fall2015/Korf_Multi-player-Alpha-beta-Pruning.pdf

[4] https://webdocs.cs.ualberta.ca/~nathanst/papers/multiplayergamesthesis.pdf

▼ Roles of each Individual

Vikash Yadav : Literature Survey, Coding the Library, Running the Simulations, Writing the report

Swara Patel : Literature Survey, Coding the Library, Running the Simulations, Writing the report

Shubham Solanki : Literature Survey, Writing the report

Harshith Reddy : Literature Survey, Setting up the experiments