Results of the AI1 Kalah Tournament

ABSTRACT

This report contains the results of the closed AI1 Kalah tournament. All teams that manage to pass the first stage will receive bonus points. The top ten teams receive additional bonus points. The tournament consists of multiple stages, where agents are disqualified if they don't perform well enough.

1. Stage "Sanity Test"

All agents are made to compete once against a random bot on a (6, 6) board. As the agent is allowed to make the first move, we know that they must be able to win the game, since this configuration of Kalah is solved. To pass this stage, one has to definitively win against the random bot, otherwise one is disqualified immediately.

1.1. Scores

Agent	Win	Loss	Draw	Score
Agent1337	**1**	**2**	**0**	**2**
sigmazero	**2**	**1**	**0**	**4**
EvaLuAtor	**2**	**1**	**0**	**4**
SimpleAlphaBeta	**3**	**0**	**0**	**6**
KGBAgent	**3**	**0**	**0**	**6**
262244	**3**	**0**	**0**	**6**
262449	**3**	**0**	**0**	**6**
Kalah Knight	**3**	**0**	**0**	**6**
MNI	**3**	**0**	**0**	**6**
watchthepartydie	**3**	**0**	**0**	**6**
258406	**3**	**0**	**0**	**6**

1.2. Game Log

Nr.	South Agent	North Agent	South	North	Diff.
1	**SimpleAlphaBeta**	**Random**	**56**	**16**	**40**
2	**262244**	**Random**	**55**	**17**	**38**
3	**Kalah Knight**	**Random**	**59**	**13**	**46**
4	**EvaLuAtor**	**Random**	**31**	**31**	**0**
5	**MNI**	**Random**	**45**	**27**	**18**
6	**262244**	**Random**	**53**	**19**	**34**
7	**watchthepartydie**	**Random**	**45**	**27**	**18**
8	**watchthepartydie**	**Random**	**44**	**28**	**16**
9	**MNI**	**Random**	**51**	**21**	**30**
10	**watchthepartydie**	**Random**	**48**	**24**	**24**
11	**Agent1337**	**Random**	**4**	**3**	**1**
12	**EvaLuAtor**	**Random**	**40**	**32**	**8**
13	**KGBAgent**	**Random**	**37**	**35**	**2**
14	**262244**	**Random**	**43**	**29**	**14**
15	**MNI**	**Random**	**46**	**26**	**20**
16	**258406**	**Random**	**55**	**17**	**38**

17	EvaLuAtor	Random	38	34	4
18	Agent1337	Random	53	19	34
19	Kalah Knight	Random	49	23	26
20	sigmazero	Random	33	39	-6
21	262449	Random	52	20	32
22	262449	Random	50	22	28
23	258406	Random	39	33	6
24	Agent1337	Random	5	4	1
25	262449	Random	50	22	28
26	SimpleAlphaBeta	Random	52	20	32
27	SimpleAlphaBeta	Random	54	18	36
28	sigmazero	Random	45	27	18
29	KGBAgent	Random	41	31	10
30	258406	Random	38	34	4
31	Kalah Knight	Random	46	26	20
32	sigmazero	Random	45	27	18
33	KGBAgent	Random	37	35	2

These agents were disqualified for failing to meet the necessary criteria for proceeding to the next round:

- Agent1337

2. Stage "Round Robin (6, 6)"

All agents play against all other agents, on both sides of a Kalah board. If one agent definitively manages to beat another agent, they are awarded two points, and the opponent is given no points. For a draw, both agents are granted a single point. The final score of this round is calculated by summing up the points for each game. Agents that suffered more losses than wins are disqualifed.

2.1. Scores

Agent	Win	Loss	Draw	Score
sigmazero	0	16	2	2
KGBAgent	1	16	1	3
MNI	3	13	2	8
EvaLuAtor	7	10	1	15
262449	9	9	0	18
262244	12	6	0	24
Kalah Knight	12	4	1	25
watchthepartydie	13	5	0	26
258406	13	4	0	26
SimpleAlphaBeta	13	3	1	27

2.2. Game Log

Nr.	South Agent	North Agent	South	North	Diff.
1	watchthepartydie	KGBAgent	56	16	40
2	KGBAgent	Kalah Knight	17	55	-38
3	KGBAgent	watchthepartydie	20	52	-32
4	sigmazero	watchthepartydie	22	50	-28
5	sigmazero	262244	28	44	-16
6	watchthepartydie	262244	44	28	16
7	MNI	262449	28	44	-16
8	262244	watchthepartydie	39	33	6
9	watchthepartydie	Kalah Knight	31	28	3
10	KGBAgent	MNI	23	49	-26

11	**262244**	**sigmazero**	**53**	**19**	**34**
12	**Kalah Knight**	**262244**	**35**	**37**	**-2**
13	**258406**	**SimpleAlphaBeta**	**23**	**49**	**-26**
14	**262244**	**262449**	**40**	**32**	**8**
15	**Kalah Knight**	**EvaLuAtor**	**44**	**28**	**16**
16	**262449**	**sigmazero**	**50**	**22**	**28**
17	**258406**	**262449**	**37**	**35**	**2**
18	**258406**	**MNI**	**52**	**20**	**32**
19	**sigmazero**	**258406**	**28**	**44**	**-16**
20	**sigmazero**	**MNI**	**35**	**37**	**-2**
21	**sigmazero**	**Kalah Knight**	**29**	**43**	**-14**
22	**KGBAgent**	**EvaLuAtor**	**21**	**51**	**-30**
23	**262449**	**watchthepartydie**	**28**	**44**	**-16**
24	**MNI**	**262244**	**16**	**56**	**-40**
25	**EvaLuAtor**	**watchthepartydie**	**26**	**46**	**-20**
26	**KGBAgent**	**sigmazero**	**41**	**31**	**10**
27	**262449**	**Kalah Knight**	**37**	**35**	**2**
28	**SimpleAlphaBeta**	**sigmazero**	**56**	**16**	**40**
29	**262449**	**262244**	**38**	**34**	**4**
30	**EvaLuAtor**	**262244**	**19**	**53**	**-34**
31	**MNI**	**sigmazero**	**36**	**36**	**0**
32	**MNI**	**watchthepartydie**	**14**	**58**	**-44**
33	**Kalah Knight**	**sigmazero**	**55**	**17**	**38**
34	**Kalah Knight**	**258406**	**33**	**29**	**4**
35	**EvaLuAtor**	**258406**	**35**	**37**	**-2**
36	**Kalah Knight**	**MNI**	**61**	**11**	**50**
37	**262244**	**258406**	**32**	**40**	**-8**
38	**EvaLuAtor**	**KGBAgent**	**53**	**19**	**34**
39	**258406**	**Kalah Knight**	**29**	**43**	**-14**
40	**SimpleAlphaBeta**	**262449**	**41**	**31**	**10**
41	**SimpleAlphaBeta**	**KGBAgent**	**44**	**28**	**16**
42	**watchthepartydie**	**sigmazero**	**45**	**27**	**18**
43	**EvaLuAtor**	**SimpleAlphaBeta**	**29**	**43**	**-14**
44	**262244**	**EvaLuAtor**	**39**	**33**	**6**
45	**258406**	**262244**	**51**	**21**	**30**
46	**262244**	**MNI**	**41**	**31**	**10**
47	**SimpleAlphaBeta**	**258406**	**39**	**33**	**6**
48	**Kalah Knight**	**SimpleAlphaBeta**	**36**	**36**	**0**
49	**258406**	**KGBAgent**	**45**	**27**	**18**
50	**Kalah Knight**	**watchthepartydie**	**35**	**37**	**-2**
51	**Kalah Knight**	**262449**	**51**	**21**	**30**
52	**262449**	**EvaLuAtor**	**26**	**46**	**-20**
53	**262244**	**Kalah Knight**	**35**	**37**	**-2**
54	**SimpleAlphaBeta**	**MNI**	**49**	**23**	**26**
55	**258406**	**watchthepartydie**	**18**	**54**	**-36**
56	**KGBAgent**	**SimpleAlphaBeta**	**17**	**55**	**-38**
57	**262449**	**MNI**	**46**	**26**	**20**
58	**watchthepartydie**	**SimpleAlphaBeta**	**27**	**45**	**-18**
59	**EvaLuAtor**	**sigmazero**	**47**	**25**	**22**
60	**Kalah Knight**	**KGBAgent**	**50**	**22**	**28**
61	**watchthepartydie**	**EvaLuAtor**	**57**	**15**	**42**
62	**MNI**	**EvaLuAtor**	**36**	**36**	**0**
63	**262244**	**SimpleAlphaBeta**	**33**	**39**	**-6**

64	**262449**	**KGBAgent**	**62**	**10**	**52**
65	**MNI**	**SimpleAlphaBeta**	**23**	**49**	**-26**
66	**262449**	**SimpleAlphaBeta**	**44**	**28**	**16**
67	**258406**	**sigmazero**	**55**	**17**	**38**
68	**KGBAgent**	**262244**	**23**	**49**	**-26**
69	**watchthepartydie**	**262449**	**49**	**23**	**26**
70	**SimpleAlphaBeta**	**Kalah Knight**	**27**	**45**	**-18**
71	**KGBAgent**	**258406**	**12**	**60**	**-48**
72	**watchthepartydie**	**258406**	**35**	**37**	**-2**
73	**EvaLuAtor**	**Kalah Knight**	**24**	**48**	**-24**
74	**MNI**	**258406**	**24**	**48**	**-24**
75	**sigmazero**	**262449**	**21**	**51**	**-30**
76	**EvaLuAtor**	**262449**	**48**	**24**	**24**
77	**SimpleAlphaBeta**	**262244**	**27**	**45**	**-18**
78	**SimpleAlphaBeta**	**EvaLuAtor**	**40**	**32**	**8**
79	**sigmazero**	**SimpleAlphaBeta**	**16**	**56**	**-40**
80	**EvaLuAtor**	**MNI**	**52**	**20**	**32**
81	**258406**	**EvaLuAtor**	**37**	**35**	**2**
82	**262244**	**KGBAgent**	**44**	**28**	**16**
83	**watchthepartydie**	**MNI**	**48**	**24**	**24**
84	**KGBAgent**	**262449**	**25**	**47**	**-22**
85	**SimpleAlphaBeta**	**watchthepartydie**	**33**	**27**	**6**
86	**262449**	**258406**	**34**	**38**	**-4**
87	**MNI**	**Kalah Knight**	**30**	**42**	**-12**
88	**MNI**	**KGBAgent**	**53**	**19**	**34**
89	**sigmazero**	**EvaLuAtor**	**33**	**39**	**-6**
90	**sigmazero**	**KGBAgent**	**36**	**36**	**0**

These agents were disqualified for failing to meet the necessary criteria for proceeding to the next round:

- sigmazero
- EvaLuAtor
- KGBAgent
- MNI

3. Stage "Round Robin (8, 8)"

All agents play against all other agents, on both sides of a Kalah board. If one agent definitively manages to beat another agent, they are awarded two points, and the opponent is given no points. For a draw, both agents are granted a single point. The final score of this round is calculated by summing up the points for each game. Agents that suffered more losses than wins are disqualifed.

3.1. Scores

Agent	Win	Loss	Draw	Score
SimpleAlphaBeta	**1**	**9**	**0**	**2**
258406	**1**	**8**	**0**	**2**
Kalah Knight	**4**	**6**	**0**	**8**
262449	**5**	**4**	**0**	**10**
watchthepartydie	**7**	**2**	**0**	**14**
262244	**9**	**1**	**0**	**18**

3.2. Game Log

Nr.	South Agent	North Agent	South	North	Diff.

1	**Kalah Knight**	**SimpleAlphaBeta**	**86**	**42**	**44**
2	**SimpleAlphaBeta**	**Kalah Knight**	**59**	**69**	**-10**
3	**SimpleAlphaBeta**	**262244**	**44**	**84**	**-40**
4	**262244**	**Kalah Knight**	**68**	**60**	**8**
5	**262244**	**262449**	**75**	**53**	**22**
6	**258406**	**262244**	**52**	**76**	**-24**
7	**262449**	**262244**	**62**	**66**	**-4**
8	**258406**	**watchthepartydie**	**61**	**67**	**-6**
9	**SimpleAlphaBeta**	**258406**	**69**	**59**	**10**
10	**watchthepartydie**	**262244**	**68**	**60**	**8**
11	**Kalah Knight**	**262244**	**62**	**66**	**-4**
12	**SimpleAlphaBeta**	**watchthepartydie**	**7**	**4**	**3**
13	**258406**	**262449**	**60**	**68**	**-8**
14	**258406**	**SimpleAlphaBeta**	**24**	**13**	**11**
15	**258406**	**Kalah Knight**	**73**	**55**	**18**
16	**262244**	**watchthepartydie**	**86**	**42**	**44**
17	**262244**	**SimpleAlphaBeta**	**85**	**43**	**42**
18	**Kalah Knight**	**262449**	**85**	**43**	**42**
19	**watchthepartydie**	**258406**	**73**	**55**	**18**
20	**262449**	**SimpleAlphaBeta**	**78**	**50**	**28**
21	**watchthepartydie**	**262449**	**62**	**47**	**15**
22	**watchthepartydie**	**SimpleAlphaBeta**	**70**	**58**	**12**
23	**262449**	**258406**	**70**	**58**	**12**
24	**262244**	**258406**	**80**	**48**	**32**
25	**Kalah Knight**	**258406**	**67**	**61**	**6**
26	**Kalah Knight**	**watchthepartydie**	**60**	**68**	**-8**
27	**262449**	**Kalah Knight**	**65**	**63**	**2**
28	**SimpleAlphaBeta**	**262449**	**57**	**71**	**-14**
29	**watchthepartydie**	**Kalah Knight**	**71**	**57**	**14**
30	**262449**	**watchthepartydie**	**51**	**77**	**-26**

These agents were disqualified for failing to meet the necessary criteria for proceeding to the next round:

- Kalah Knight
- 258406
- SimpleAlphaBeta

4. Stage "Round Robin (10, 10)"

All agents play against all other agents, on both sides of a Kalah board. If one agent definitively manages to beat another agent, they are awarded two points, and the opponent is given no points. For a draw, both agents are granted a single point. The final score of this round is calculated by summing up the points for each game. Agents that suffered more losses than wins are disqualifed.

4.1. Scores

Agent	Win	Loss	Draw	Score
262449	**0**	**4**	**0**	**0**
262244	**3**	**1**	**0**	**6**
watchthepartydie	**3**	**1**	**0**	**6**

4.2. Game Log

Nr.	South Agent	North Agent	South	North	Diff.
1	**262244**	**watchthepartydie**	**102**	**98**	**4**

2	**262449**	**262244**	**84**	**116**	**-32**
3	**262449**	**watchthepartydie**	**94**	**106**	**-12**
4	**262244**	**262449**	**108**	**92**	**16**
5	**watchthepartydie**	**262244**	**103**	**97**	**6**
6	**watchthepartydie**	**262449**	**101**	**99**	**2**

These agents were disqualified for failing to meet the necessary criteria for proceeding to the next round:

- 262449

5. Stage "Round Robin (12, 12)"

All agents play against all other agents, on both sides of a Kalah board. If one agent definitively manages to beat another agent, they are awarded two points, and the opponent is given no points. For a draw, both agents are granted a single point. The final score of this round is calculated by summing up the points for each game. Agents that suffered more losses than wins are disqualifed.

5.1. Scores

Agent	Win	Loss	Draw	Score
watchthepartydie	**0**	**2**	**0**	**0**
262244	**2**	**0**	**0**	**4**

5.2. Game Log

Nr.	South Agent	North Agent	South	North	Diff.
1	**262244**	**watchthepartydie**	**159**	**129**	**30**
2	**watchthepartydie**	**262244**	**125**	**163**	**-38**

These agents were disqualified for failing to meet the necessary criteria for proceeding to the next round:

- watchthepartydie

6. Final score

The top ten agents are as follows:

1 262244 (Score: 58)

2 watchthepartydie (Score: 52)

3 Kalah Knight (Score: 39)

4 SimpleAlphaBeta (Score: 35)

5 262449 (Score: 34)

5 258406 (Score: 34)

6 EvaLuAtor (Score: 19)

7 MNI (Score: 14)

8 KGBAgent (Score: 9)

9 sigmazero (Score: 6)

Congratulations to all participating teams!

The remaining scores are:

10 Agent1337 (Score: 2)

7. About the Agents

This section contains the abridged, partly spellchecked, sometimes translated and slightly editorialized (where *needed*) contents of the ABOUT file, if it was submitted.

7.1. sigmazero (262621)

Idea

The idea was MCTS with AlphaZero

Implementation

The agent is implemented in Python. The goal of this work consisted of creating an agent that could play the game of Kalah as strongly as possible. The most naive method (though very effective) method is to implement the min-max algorithm, as was described in the lecture. There are limited opportunities for improvements there, however.

The easiest and most important improvement is to implement alpha-beta pruning, which entails an enormous speedup (ca. *8). The second optimization is caching already expanded game states, which proved ineffective, however. We are left with two options:

1. Adjust the heuristic function
2. Implement move ordering to optimize alpha-beta pruning

(2.) can entail an enormous performance boost, since better moves are tried first, which reduces the size of the search tree. We get a ca. *3 improvement.

Concerning (1.), one could try different mathematical functions, e.g. to prefer states with many seeds in the pits before one's home, since they are not as easy to steal. Alternatively, one could prefer positions whithout big numbers of seeds in one's pits. All of those complex functions bear the risk of not being sensible on a high level of play. They are also computationally expensive. The report for the Kalah Challenge 2022[1] mentions that teams played around with different heuristic functions, which could only provide limited improvements. It also mentions that not a single team implemented a different algorithm, like Monte Carlo Tree Search (MCTS). I want to change that with this work.

Monte Carlo Tree Search

The MCTS algorithm is significantly more complex, but it offers more opportunities for optimization.

We started by trying to implement an ordinary MCTS algorithm. Contrary to minmax, the algorithm does not evaluate states by a heuristic, but by contiunally expanding the game positions all the way to the end of the game. There are two possibilities for improvement:

1. The choice of expanded nodes. We used the upper confidence bound for trees (UCT), because it is well-researched and applicable to many MCTS use cases.
2. The choice of an effective policy in the rollout phase. The simplest variant of MCTS uses a random policy. With that, a MacBook M1 is able to perform 40,000 iterations of MCTS in 5 seconds. Nethertheless, a min-max algorithm with depth 2 beats MCTS with this policy easily.

Thereupon, we tried implementing a greedy algorithm as the policy, with a heuristic of

South - North

We reached ca. 8000 MCTS iterations, though this algorithm is still worse than minmax. The report mentioned that a combined MCTS and minmax could give good results. We implemented an agent that switched from MCTS to minmax after the number of available seeds dropped to a specific number (< 20). This did not improve our results.

Therefore we made the decision to implemented the well-known AlphaZero algorithm for this game. There are open source templates[2] for other games, particulary Othello, where this algorithm delivers very good results. For this project, we adjusted the template for the game of Kalah and we explored an architecture for the neural net. That was

[1] Editor's note: `https://wwwcip.cs.fau.de/~oc45ujef/ai/kalah22.pdf`

[2] `https://github.com/suragnair/alpha-zero-general`

based exclusively on fully connected residual blocks, since seeds have little local, but rather global dependencies in Kalah.

For effective training of the algorithm, we implemented parallel processing, although with errors. We did not have access to the correct hardware. Training on the computers in the CIP-Pools produced errors and was strangely slower than on a MacBook. The model was trained 40 times with different parameters (MCTS-simulations, arena-compare(50-400), cpuct (1; 1.5), update-threshold (0.52-0.6), temp-threshold (15-35)), with neural net architectures as well as parallel and linear computation. The Fritz Cluster of the HPC at FAU was requested, but denied.

The parameters of the original paper for the game of Go and Othello were not used, since they would have required a lot more resources. The game of Kalah is simpler in many ways, so simpler parameter constructions should be justified. In most cases the model improves initially, but then hits a plateau where it does not keep improving. Maybe with more computational resources a better result would have been achievable, but the current model barely manages to beat a random agent.

In total, we spent around 60-70 hours on this project. All to build an agent, that is significantly worse than the provided minmax algorithm. It is possible that bugs that hinder performance dramatically exist in the implementation, or more training would have been necessary. Sadly the algorithm only works with a board size of 8.

7.2. SimpleAlphaBeta (257394)

Idea

I wanted to implement the Agent in a more performant language leading to this pure C++ implementation. I was thinking about going for a Monte Carlo driven approach but decided that this would likely require a neural network to function properly, which would require extensive self play, which i did not have the resources for. Instead I pivoted to a very basic MiniMax approach.

Implementation

I started up with a pure minimax limited to a certain search dept. Once this worked i wanted to make sure that the agent does not waste any time it has untill the game is solved so i started working on the time component. The controller now measures the latency using the initial "ok" message and keeps flags determening if the calculation should be interrupted. This lead to a dynamic search dept that always uses the entire available search time (if board not solved). The next step would be implementing alpha beta pruning for more efficient search using a basic store difference eval function, which i later adapted a little to also include the difference of stones in pits. The next step was adding move ordering and a quiescenceSearch.

The following was implemented but did not work as intended:

- completely reworked the board class to now offer undo unfunctionality to prevent the countless board copys
- added bitmasks and highly optimized the get legal moves function in the board
- introduced a transposition table using zobristHashes in a static sized vector, this increased the search depth by almost 5 steps
- I tried learning weights for a better eval function but this did not work, so i scrapped it.

Very early on my agents keep beating the ones provided on the training server consistently, so i switched to comparing them against each other. Literal dozens of hours later i once again let one of now really advanced agents play on the server and to my horror it performed worse than the rather simple one in the beginning. About a week later and with the deadline approaching i could not figure out what has happened and why the agents got worse all of a sudden, so i gave up, scrapped everything and came back to on of the earlier iterations that i know worked against the agents provided on the server. (honestly i lost track by now and i am not sure anymore)

7.3. EvaLuAtor (257864)

Idea

The general approach was to create a game-winning agent with the least amount of time and effort possible â basically, a lazy implememtation that still somehow manages to win a few rounds.

Implementation

Our agent is implemented in Python using the provided template. We began by implementing the MinMax algorithm with alpha-beta pruning and then made a few adjustments to the evaluation function to improve performance.

7.4. KGBAgent (261360)

Idea

Minmax agent with alpha-beta pruning and cache.

Implementation

The agent is written in Java.

7.5. (262244)

Idea

Our general approach was to implement a MinMax search with alpha-beta pruning and move ordering.

Implementation

The agent is written in Python using the provided template. First we implemented MinMax search with alpha-beta pruning and move sorting. We then implemented a heuristic function (score-difference) that would evaluate the board state. After a couple of runs on the server, we realized that the heuristic function was not enough to beat the better agents (MinMax-6 and higher). Finally, by implementing a cache which stores the score value for each move for a given board state from the previous depths, we were able to beat all the bot agents.

7.6. (262449)

Idea

Our general approach was to implement the alpha-beta-search algorithm by setting the beta and alpha values and choosing the best move depending on whose turn it is.

Implementation

We began writing our agent by using the Python template which implements the MinMax algorithm as a reference.

7.7. Kalah Knight (262452)

Idea

Our idea was to implement aplha beta pruning search with evaluation function and ordering strategy that works best in our experiments.

0. Evaluation Function

- Basic -> Difference of score in our and opponents pits
- Basic with Capture -> we added bonus if there was a capture possible
- Basic with Capture v2 -> along with our bonus capture, we would penalize if our move would let other user capture our seeds
- Basic with Capture v2 and total pit score -> same as above but this time we add difference of total seeds on our side vs opponents side
- Basic with Capture v2, total pit score, potential move -> we count how many pits are non zero (leading to possibility of move) on both sides and take a difference

1. Alpha Beta Prune ordering

- with no ordering
- with ordering based on evaluation function, highest value first
- with random ordering

Alpha-Beta Pruning with basic evaluation function and random ordering performed best.

Implementation

We began with trying out the already exisisting code and found that it was performing very poor. So we changed the agent code -> to incorporate better yeilds with depth of 2 to `kgp.size` and tweaked the evaluation function to consider difference between the number of seeds between our and opponents pit. Surprisingly, that worked well and was winning most of the matches it played except for full MiniMax agents like `MiniMax-4`. Our orignal idea was to implement alpha-beta pruning with better(probably complex) evaluation function to beat other agents.

Althouth we use alpha-beta pruning, we found out that our agent was not exploring very deep depths. During the midgame, it looks up to 8 moves ahead, and in the endgame, due to the fact that there are less possible moves, it can look up to 18 moves ahead. To try to improve this, we tried increasing the recursion limit, which in python is 1000 by default. We performed some tests with recursion limit of 4000 and even 8000, but at the end after performing some tests, we realized it didn't make a big difference. To test this and to test the different evaluation functions, we created multiple agents and let them play against each other. Namely:

Agent Name	Evaluation Function	Recursion Limit	Additional Info
Kalah Knight	Difference between scores	1000	This is the base agent
Kalah Knight 1000	Difference between scores	1000	This agent outperformed the others
Kalah Knight 4000	Difference between scores	4000	This agent was eventually outperformed
MinMax De	Complex evaluation function	1000	Performs well, but was outperformed

Additionally, we tried ordering the child nodes using the evaluation function, to achieve better pruning by alpha-beta. However, this didn't work as well as we expected, as the same depths were being explored, and there was not gain in performance or efficiency at all. Quite the opposite, we found out that better results were obtained using random ordering.

7.8. MNI (262499)

Implementation

The agent is written in Python.

7.9. watchthepartydie (262606)

Idea

Our general approach was to use a minimax algorithm with alpha-beta pruning and move ordering.

Implementation

The agent is written in Python using the provided template. We began writing our agent by developing a standard minimax algorithm. After that, we added alpha-beta pruning, move ordering, memoisation, and experimented with different evaluation functions.

7.10. Agent1337 (262622)

Idea

Our general approach was to write a Minimax algorithm with Alpha-Beta pruning that priotizes moves based on potential extra turns and capture opportunities.

Implementation

The agent is written in Python using the provided template. We began writing our agent by expanding the given Minimax algorithm.

7.11. (258406)

Idea

Father forgive me for I have sinned. Instead of writing the agent in `Haskell` like the lord intended, I decided to use `C++` to get more comfortable with the language. The algorithm is minmax with iterative deepening search, alpha-beta pruning and move ordering.

Implementation

Honestly, I did not realize that there are prebuilt libraries for the game logic, so I implemented it from scratch. It was a good exercise, but I really struggled to even get linked lists working (damned pointers!!11!). I used `kgpc` for communication. The rest was straightforward:

1. Implement minmax
2. Add alpha-beta pruning
3. Add move ordering
4. Wonder why agent still loses to MinMax-2
5. Lose sanity

...

6. Regain sanity
7. Realize that the minmax algorithm has been flawed from the beginning, evaluations were used completely wrong.
8. Fix Minmax
9. Realize that the evaluation function is also garbage and you should revert to the basics.
10. For most moves the agent reaches a depth of around 9 => happy.

The evaluation function is a linear function that only considers seeds in the stores and on the field. For move ordering we use a different evaluation function, that evaluates moves in pretty much the same way, except that captures are prioritized.

8. Wall of Shame

This section contains a list of teams and their sins (in no particular order and with wildly different severity).

Team	Infraction
262622	We did not install `numpy` in our docker container
262499	We did not submit in the required format
262499	We did not submit an `ABOUT` file
262622	Our agent does not connect correctly
257864	We did not check the boxes in the `ABOUT` file
261360	We did not submit our source code initially
261360	The source code we submitted did not compile initially
261360	We used an outdated template
pkal	I do not understand `POSIX` shell expansion