T-622-ARTI: Gervigreind
Project 1: Knight through

Ingólfur Ari Jóhannsson

February 8th 2023

1 intro

Knight-Through is a modified version of chess where all pieces are horses, but they move and kill like pawns and cannot move backward. It is an adversarial search problem, where an agent must search the game tree to find the optimal move given the current board state. In our case the state-space is very big, so we need to implement an iterative deepening search where we improve our estimation with each depth step.

2 State space

The state space model of the game is board state, which is represented as a two-dimensional array where each element represents a cell on the board. Each cell can be either empty or contain a horse piece of either color white or black. The bare-bones state space will then be $3^{w*h} * 2$, where there are 3 options for each cell, w = width, h = height and either player has the current turn. But this can be simplified and made smaller by excluding impossible scenarios, for example you cant have 2 black knights on the first rank, the game would already be over. We also have constraints on how many pieces there are allowed on the board, which are 2*width for each player, this would also have to be taken into account for the true state-space.

The legal moves for each piece are calculated based on the current position and position of opponents. The state update function is used to apply the chosen move to the current board state, and the termination test is used to check for winning, losing, or tie conditions. The pieces can only kill like pawns in regular chess, and can only move forward like a knight, this makes the game much shorter and simpler.

3 Search

The search algorithm used in the agent is a combination of mini max with alpha-beta pruning and iterative deepening. Mini max is implemented correctly, and alpha-beta pruning is done correctly, including pruning at the all levels of the tree. Iterative deepening is also done correctly, with timing handled appropriately to ensure that the agent does not run over the time limit. The move returned by the agent is the best move from the last iteration of the iterative deepening that finished.

3.1 heuristic

To improve the agent's performance, a heuristic function is implemented. The heuristic incentivizes pieces to move to the middle of the board to gain more control over the game. Winning terminal states are heavily incentivized, while losing states are heavily dis-incentivized. Ties are also given a negative score, although not as severe as a loss. Many generations and attempts were made to improve upon the original heuristic.

4 Time saving

One noteworthy feature of the agent is the use of a hash map to learn from previous games. Although a transposition table was not implemented, the hash map is used to store previous board states and their corresponding values. This allows the agent to make more informed decisions based on past experience, which can lead to a much better performance in future games. If you run the 5x3 map over and over (in a non-deterministic environment) for example it will quickly run entirely on the hash map as it builds on past visited spaces.

5 summary

Overall, the agent performed well in the Knight-Through game. With the appropriate combination of search algorithm, heuristic function, and hash map learning, the agent was able to make semi-optimal moves and win against opponents (I can't beat it anymore). The implementation of a hash map to learn from past games is a good feature as long as you don't terminate the program, but we could easily implement a save function. The hash map could also be further improved with the implementation of a transposition table.

6 experiments

do some experiments, put them in a readable format, graphs, statistics etc. Do this the same time as we implement along the steps

Experiments were conducted with different board sizes and time constraints to evaluate the agent's performance versus the original heuristic. The results showed that the agent performed better with larger board sizes and longer time constraints. The heuristic function improved the agent's performance compared to the default heuristic provided in the instructions, but not in all cases (it did better with longer time). Also there is a clear bias towards white for smaller boards, the advantage of beginning is enough to win in all cases.

Here is the data: x is a win, - is a loss.

Tafla 1: We play black 3x5

time	1	5	10
white	x	x	x
black	-	-	-

Tafla 2: We play white 3x5

time	1	5	10
white	x	x	x
black	-	-	-

Tafla 3: We play black 5x5

time	1	5	10
white	x	x	x
black	-	-	-

Tafla 4: We play white 5x5

time	1	5	10
white	x	x	x
black	-	-	-

Tafla 5: We play black 6x6

time	1	5	10
white	x	-	-
black	-	x	x

Tafla 6: We play white 6x6

time	1	5	10
white	x	x	x
black	-	-	-

Here you can see my heuristic breaking through and starting to win against the advantage of beginning with the white role.

Tafla 7: We play black 7x7

time	1	5	10
white	-	x	-
black	x	-	x

Now after this setup, my heuristic wins in all cases, I can safely assume my strategy is better than the original.

Tafla 8: We play white 7x7

time	1	5	10
white	x	x	x
black	-	-	-

Tafla 9: We play black 8x8

time	1	5	10
white	-	-	-
black	x	x	x

Tafla 10: We play white 8x8

time	1	5	10
white	x	x	x
black	-	-	-

Tafla 11: We play black 9x9

time	1	5	10
white	-	-	-
black	x	x	x

Tafla 12: We play white 9x9

time	1	5	10
white	x	x	x
black	-	-	-

Tafla 13: We play black 10x10

time	1	5	10
white	-	-	-
black	x	x	x

Tafla 14: We play white 10x10

time	1	5	10
white	x	x	x
black	-	-	-

7 Conclusion

In conclusion, I found this assignment to be quite challenging and required a significant amount of research. While working on this project, I faced difficulties with my group and made the decision to proceed independently. Initially, I attempted to create a Tree/node model to traverse the tree recursively, but this approach proved to be inefficient and ineffective. With guidance from Magnús, I realized that using the call stack as a tree could achieve the same algorithmic structure without the need for objects. This approach improved the code in multiple ways.

During the implementation process, I also made several heuristic attempts, which in hindsight, were not the most effective. I learned that it is essential to consider heuristics as a useful endpoint, and to always prioritize strategies that have been proven to be beneficial. For instance, controlling the middle position, striving for wins, and avoiding draws can enhance the overall outcome.

Overall, this project was challenging, but I learned a great deal and improved my problem-solving and decision-making skills.