Danmarks Tekniske Universitet

DTU

02180
Introduction to Artificial Intelligence

Assignment 1

Authors:
Connor Wall - s217170
Peter Revsbech - s183760
Kasper Solhøj Jørgensen - s194579
Tobias van Deurs Lundsgaard - s194616

Date: 21.03.2022

1 Game rules

The game we have chosen is Kalaha. The overall goal of the game is to collect more seeds in your store than your opponent collects in their store. In a players turn, the player takes all the seeds in one of their houses, and moves in the direction of play, dropping them one by one in the houses encountered and in their own store.

In figure 1, the basic terminology of the game, that we will use throughout the report, is illustrated. Each player has 1 store and 6 houses, and the game has a total of 48 seeds.

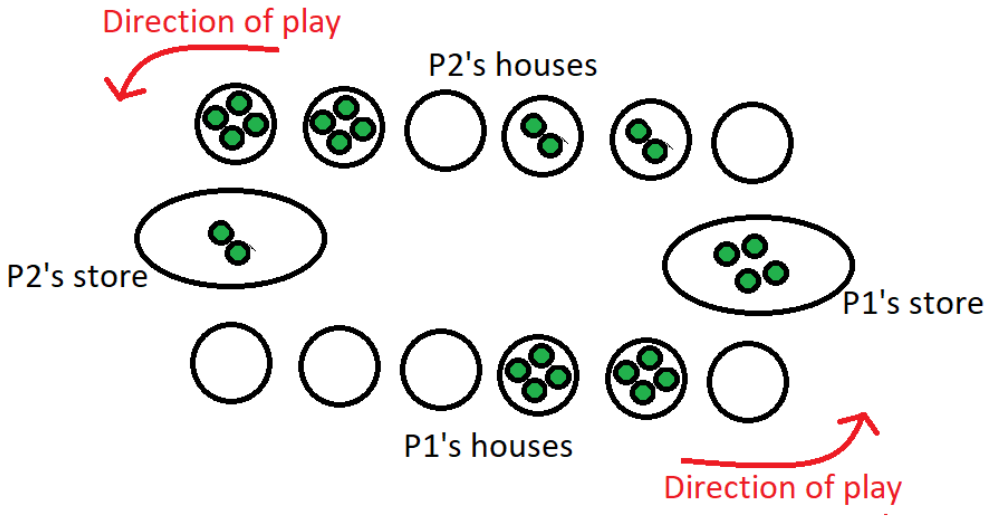

Figure 1: The Kalaha board and the terminology used. The green balls are called seeds. Note that the number of seeds in the actual game is 48, but this is a simplified model.

All the rules we have implemented are not universal for every game of Kalaha, so the more technical rules are listed below:

- There are 4 seeds in each house upon starting the game.
- You may only pick from a house on your side of the board.
- If your last seed drops in your store, you may pick one of your own houses again and continue your turn.
- If your last seed drops in a house, your turn is over.
- When the player who's turn it is has no seeds in any of their houses (no valid moves), the game is over. The remaining seeds of the game are counted as the other players points.
- The player with the most points at the end of the game wins. If the players have the same number of points, it is a tie.

2 What kind of game is it

Below is a list of attributes which we think describe our game of kalaha quite well.

- Our game is a multi-player game, made in a way so two humans, one human and one AI or two AI can play each other.
- This game is competitive as the player with the most seeds in there store wins.
- Since there can only be one winner of the game, the game is a zero-sum game.
- The game, kalaha, is turn-based so our implementation of it is also turn-based, but a player can have multiple turns in a row.
- The player has full observably over the game.
- Since each series of possible moves will give the same end state our game is deterministic.

As the game is a zero sum turn-based two-player game, we need to use an algorithm which takes this into account. The algorithm does not need to take randomness in to account as the game is deterministic.

3 State space

We will now try to estimate the size of the state-space of our Kalaha game. We have 12 houses, 2 stores and initially 4 seeds in each house. We consider that there are a total of 48 seeds, and each of these can be placed in 14 different places (either a store or a house). This totals $14^{48} \simeq 1.0 \times 10^{55}$ states which is astronomically large. The issue is that this way, we consider the location of each individual seed, although the seeds are indistinguishable. Instead, we should consider the number of seeds in each place in order to tighten the bound.

A better upper bound can be found with the following argument: There are 48 seeds in total in the game. In each house and each store, we can have no more than 49 configurations (between 0 and 48 seeds). So in total, this would give us an upper bound of $49^{14} \simeq 3.4 \times 10^{22}$ states. We know that this is still a very loose upper bound, since we consider configurations where the total sum of seeds i larger than 48.

To tighten the bound, we might consider, that actually there are only 14 configurations, where a place (house or store) has 48 seeds. There are only 14×13 configurations, where one place has 47 seeds and another has 1 and so on. This way of thinking quickly gets extremely complicated as well, since the number of ways we can distribute these 48 seeds is exponential in the number of places, we can place them, which is 14.

We might also want to find a lower bound of the size of the state space. Here, we can try and consider the number of configurations, when there can be at most 4 seeds in one place, which would give us $4^{14} = 2.6 \times 10^{8}$. The problem is, that we cannot prove that all these configurations are reachable states in the game.

To summarize this analysis, we can only conclude that the size of the state space is no larger than 3.4×10^{22}.

4 Game elements

It will now be explained, how the functions RESULT, ACTIONS, UTILITY and so on are implemented in the project. Below is a table of what their names turned into in our code:

- PLAYER(s) -> isP1Turn()
- ACTIONS(s) -> actions(s)
- RESULT(s,a) -> performMove(s,a)
- TERMINAL-TEST(s) -> goalTest(s)
- UTILITY(s,p) -> evaluateFinishedGame(s)

We chose these names as they were easier for us to understand while coding. Some parameters changes that fit our program. Namely isP1Turn does not take a parameter as it is located in the State class and thus know the state. evaluateFinishedGame only takes a state as it is already known what players turn it is when the method is called.

Now we move on to the methods. isP1Turn returns true if it is player 1's turn. actions return an array containing the legal houses the player must choose from. performMove returns a new state that is obtained after picking the seeds in house a. goalTest checks if the state is a terminal state. It is terminal if the player who's turn it is has no legal moves. evaluateFinishedGame evaluates the point-value of a finished game. It returns the max-value for a win, 0 for a tie and the min-value for a loss. Since it is a zero-sum game, it holds that $max\_value = -min\_value$

5 Representation

The initial state s0 and all other states are stored in an array of length 14. Rules of the array:

- Index 0 is the store of player 2.
- Index 7 is the store of player 1.
- Index 1-6 are the houses of player 1.
- Index 8-13 are the houses of player 2.

In the initial state, all houses have 4 seeds and the stores have 0. So s0 would be an array with the following data:

$$s0 := \{0, 4, 4, 4, 4, 4, 4, 0, 4, 4, 4, 4, 4, 4\}$$

The actions method returns an array containing the indices of the houses from which the player can draw in their turn.

To make the AI for this game it was enough to represent a game state as an array indicating the number of seeds in each store and house on the board. It was not necessary to have belief states as there are no elements of randomness or any other kinds of uncertainty in outcome when doing any move.

6 AI and illustration

To make an AI that can play Kalaha, we need to make sure the AI attempts to win against its opponent. The minimax algorithm is ideal in this case. Algorithms like And-Or-Graph-Search could be used to traverse a possible set of game states if we assumed the opponent chose their actions randomly. However, it would be more beneficial for the AI to assume its opponent is rational. This allows for better pruning of the game-state-tree.

For our game we have used the minimax algorithm described to us in the course. We added alpha-beta pruning to the algorithm, which is also as we have been taught in the course. Examples of the minimax algorithm can be seen in the book. Our AI also uses iterative deepening with a time limit. This means that we repeat our search, searching one layer deeper each iteration. We cease starting new searches if we surpass the time limit. Searches already in progress get to finish, even if the limit is surpassed.

Now the way we use the minimax algorithm is a bit different to the examples in the book. Due to Kalaha being a game in which a player may or may not make multiple moves in their turn, we needed to redefine how we look at states. When using minimax on a state, we split its child states in two separate groups. The two groups are defined by whether or not the player turn has changed in the new state. We call states that have the same player turn internal children, and states that have a different player turn external children. Whenever we expand a node to get its children, we expand all internal children to the fullest so that we get all possible external children reachable from the initial node.

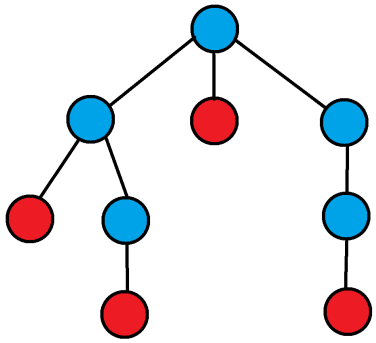

Figure 2: A single call to max may look like this. It should be noted that in a real scenario in the game, the tree would likely be a lot bigger.

In the image above, blue nodes (states) are internal children and include the initial node as well as all descendants where the turn did not change. The red nodes are external children where the turn changed. Assuming it was a max call that was made by the minimax algorithm, all blue nodes are considered the same max call. This way we consider all possible outcomes of the moves a player can perform before the turn is passed to the other player.

The motivation to logically separate the internal and external children is not that the min-max algorithm won't work without this abstraction. Technically, a maximizer-node could just take the max value of it's children even though it had both maximizer and minimizer children. Instead, the

motivation for this is to emphasize, that we can treat the internal and external children differently. In our algorithm, we only count the depth of a node by the number of external nodes from the root, e.g. the number of times the turn has switched between P1 and P2. This way, we always explore all possible outcomes of a move-sequence before the turn is handed over.

7 Evaluation function

In Kalaha, you are more likely to win the greater the difference between your store and your opponents store is. So for our game, the evaluation function aptly named evaluateState simply finds this difference with the formla: $P1\_Store - P2\_Store$. When a game is finished, we use a different function, evaluateFinishedState. This simply evaluates the point difference when a game is finished. It does the same as evaluateState, but also counts the remaining seeds as points for the player, who still has seeds on their side of their board.

8 Adjustable parameters

There are two adjustable parameters in the project. One is the time limit indicating how long an agent has to search for a solution. This parameter has the biggest impact on how well the AI preforms in the game. The time limit could for example be set to 50 ms, meaning that the agent cannot start searching on a new depth after 50 ms. Generally, the larger the search depth, the better the agent will play.

We have also made it possible to adjust the number of seeds in each house when the game is initialized. In the rules described, there are 4 seeds in each house initially, but this number can easily be increased to e.g. 10 making the game trees a lot larger and more difficult to search.

A test has been added to illustrate the difference it makes when the initial number of seeds is adjusted. The test plays a number of games between two agents given a specific number of seeds in each house initially and search time limit. It then evaluates how many wins each player gets (counting a tie as 1/2 win for both players). When experimenting with this test, it is clear that with only 4 seeds pr house initially and 50 ms search time, P1 has a huge advantage over P2, winning almost all of the games[1]. For a higher number of seeds initially, this advantage becomes much smaller[2]. This is probably because the game with 4 seeds per house initially is relatively simple. The AI can find a more or less perfect strategy for P1 to win every time, since P1 has the first turn. For 8 seeds per house initially, this effect seems less important. The tests can be repeated when the program is run, by following the instructions given in the console.

[1] The first run of the test with 4 seeds pr house initially, 50 ms search time and 20 games gave 20 wins for P1 and 0 for P2

[2] Another test with 8 seeds per house initially, 50 ms search time and 20 games gave 8.5 wins for P1 and 11.5 wins for P2.

9 Improvable features and reflection

While our solution incorporates minimax with alpha-beta pruning and iterative deepening, it is not perfect. If we were to enforce a strict time limit on the agent, meaning that it should not be allowed to search for more than e.g. 5 seconds, we would need to add an interrupt system to the iterative deepening. Since this was not a requirement, we just implemented a soft time limit instead.

We are very satisfied with our choice of algorithms, as it is virtually impossible for us to beat the AI when playing against it. With 4 seeds per house initially and a search time limit of 50 ms, our algorithm has been capable of doing up to 9 layer deep searches[3]. This is thanks to the alpha-beta pruning, which turned out well.

If we were to improve the agents performance even more, the most important aspect to look into would be move ordering. Right now, the moves in the min-max algorithm are not ordered according to any heuristic. This hugely restricts how large a cutoff we can get from alpha-beta pruning, since we do not make any effort to find the best moves first. Good move ordering functions can often improve the cutoff substantially, and it would not be that difficult to create one for Kalaha. We could for example choose to prioritize moves that end up in our own store, since this will give a point and an additional turn.

Another possibility for improvement could be to create a transposition table, e.g. a table containing the results of states that have already been searched. This could improve search times, since we could avoid doing redundant work of searching the same states multiple times.

The data structure of the states is a bit nasty to work with, as it is just one array with designated indexes for certain information. We would probably have changed it so that it had its own data structure with stores and houses separate. This was not done because working on the actual algorithm was prioritized.

[3]Remember that we count the depth of the search as by the "external" nodes, e.g. the times the turn switches.