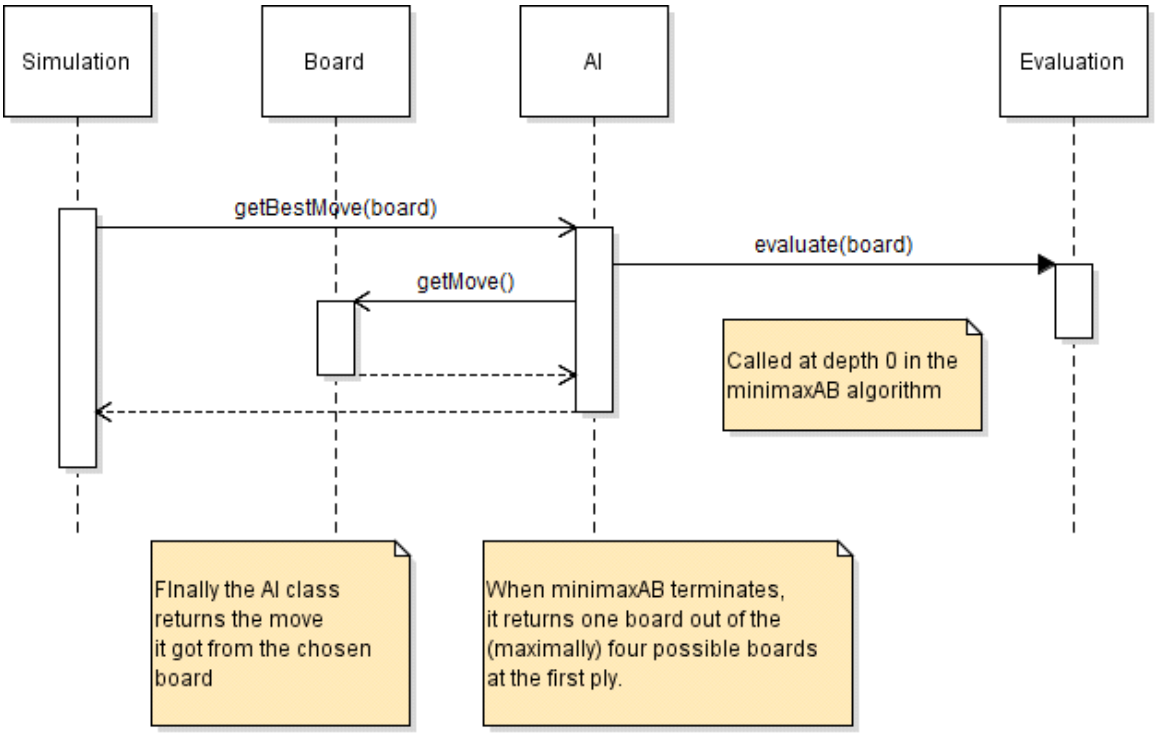
2048 Report - Johannes Moskvil

22. oktober 2015 14:17

Structure and key components

For this assignment I chose to use the Minimax algorithm with alpha-beta-pruning. The structure is quite simple. I have a simulation class that calls alphaBetaPruning which returns the best calculated move, that the simulation class in turn executes. This process is repeated until there is no more possible moves. Almost no changes had to be made to the standard out of the textbook implementation of minimax-with-alpha-beta-pruning. The only challenge and modification, is that the standard implementation returns the *value* of the selected leaf node, **not** the *move* that got it there. To solve this, I added a conditional in the max part of the algorithm to store the returned values at the first branch, of which there is four. When the algorithm terminates, instead of returning the value, it returns the board whose evaluation corresponds to the max value at the first branching. Each board object has an attribute "move" that says which direction was used to get there. This is the value returned by the minimax-a-b algorithm.

Below is a diagram that demonstrates the key components and overall structure of the system.



Heuristic function

Heuristics related to actual tile values are calculated in $\log_2$ space, the reason for this is that for example 512 is not actually 256 better than a 256 tile. It only requires one merge, which is what the result of $\log_2 512 - \log_2 256$ is, and hence motivates calculating board differences in $\log_2$ space.

My heuristic function is composed of 5 individual heuristics that are given different weights that influence how "important" they are. The evaluation value returned will be the sum of the following heuristics that have been multiplied by their associated weight. Naturally, there is also a high

punishment for moves that results in game over, so it will never choose to do such a move unless there is no other possibility.

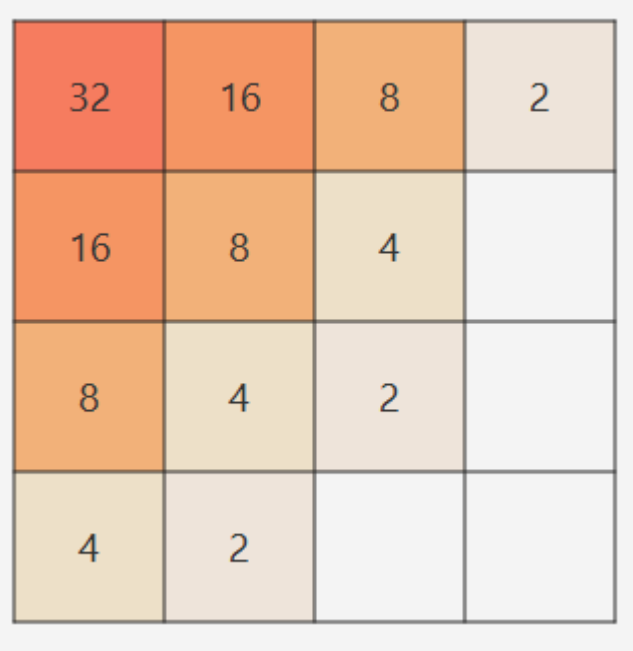
These properties are generally recognized as representing a good standing.

- Monotonicity
- Smoothness
- Number of empty tiles
- Highest tile achieved
- Highest tile in a corner

With these heuristics I try to mimic how most humans play the game, by trying to have the largest tile in a corner, and build it there. I'll go through each of these heuristic and explain how they are calculated as a combination of pseudocode and mathematical expressions.

MONOTONICITY

The idea is that you want to have a board that diagonally descends in value. This is an example of a perfectly monotonic board.



Which is given the score of 0. Boards that violate this constraint is punished by receiving a negative score.
Monotonicity is calculated as this.

1. Initialize score to 0.
2. For each row on the board do:
   $score = score\ -\ unitMonotonicity(row)$
3. For each column on the board do:
   Score = $score\ -\ unitMonotonicity(column)$
4. Skip to 1 and repeat for each of the four corners.
5. Return the highest of the four scores.

Where unitMonotonicity is defined as: $\sum_{i=0}^{2} |\log_2 b_{i+1}\ - \log_2 b_i|\,, if\ b_{i+1} > b_i, otherwise\ skip\ adding.$
And b is the row or column to calculate score for.

Running the monotonicity heuristic will yield the following scores for each corner.

Upper left	0
Upper right	-10
Lower right	-23
Lower left	-13

Hence it evaluates the upper left corner as the best and returns this value, 0.

SMOOTHNESS

Smoothness measures how easily it will be to merge tiles together, having neighbours of the same value (disregarding empty spaces) improves the smoothness score. By having a high smoothness, you will be in a good position to merge tiles together on subsequent plies.

Smootness is calculated like this.

1. Initialize score to 0.
2. For each row on the board do:
   $score = score\ - unitSmoothness(row)$
3. For each column on the board do:
   Score = $score\ - unitSmoothness(column)$

Where unitSmoothness is defined as:

1. Set score to 0.
2. Filter out all 0s from the row or column to be evaluated.
3. Score = score + $\sum_{i=0}^{2} |(\log_2 b_{i+1} - \log_2 b_i)|$

NUMBER OF EMPTY TILES

Having few empty tiles generally depicts that you are close to losing the game. This heuristic is simply meant to favor boards with a lot of empty space.

$$EmptyScore = \sum_{x=0}^{3}\sum_{y=0}^{3} 1\,, if\ b_{xy} == 0$$

HIGHEST TILE ACHIEVED

This heuristics gives a bonus score for the highest tile achieved. Generally this gives a small advantage to boards with a higher max tile. But the real bonus comes from giving a signifantly higher bonus to tiles beyond 2048. For example, I give boards containing the 2048 tile an additional bonus of 10 000 to ensure that the AI will always prefer this move, even if it gives a worse structure.

$$MaxTileScore = \max(b_{xy})\,, for\ x\ in\ range\ 0{,}3\ and\ y\ in\ range\ 0{,}3, where\ b\ is\ the\ board\ matrix$$

HIGHEST TILE IN A CORNER

Part of the normal strategy involves getting the highest tile in a corner, which is the reason I give an additional boost for achieving this. It only gets credit for the corner with the highest tile, if that tile is indeed the highest achieved anywhere on the board.

$$HighestCornerTile = \max(b_{00}, b_{03}, b_{30}, b_{33})\,.$$

If the HighestCornerTile equals MaxTile then that value is returned, otherwise it just returns 0.