Today:

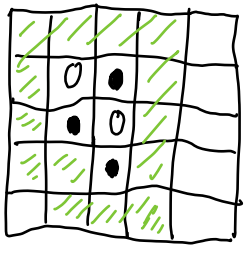
- Advanced intelligence
  - Efficiency
  - MCTS

} (Recorded)

- Discussion on projects (Not recorded)

How to make the implementation efficient.

- Calculate the possible moves
  - Extend the Igra class with a field meja (boundary) Set<Poteza> (HashSet<Poteza>)



  - Calculate the list of possible moves by iterating through the boundary set
  - Add the list of possible moves as a new field to the Igra class

    List<Poteza> poteze

    (LinkedList)

    which is calculated dynamically using the meja set.

Blocked positions

Detecting a blocked position:

- Position blocked if current player cannot move but opponent can.

Dealing with a blocked position

- Add a new state Stanje.BLOKIRANO

or • represent it using a choice of "dummy move" e.g. Poteza(-1, -1).

How to do position evaluation

I'm using

#my pieces − #opponent's pieces

(very crude!)

I maintain the score as a field in the Igra class and update when pieces are turned over.

Two issues with minimax / $\alpha\beta$

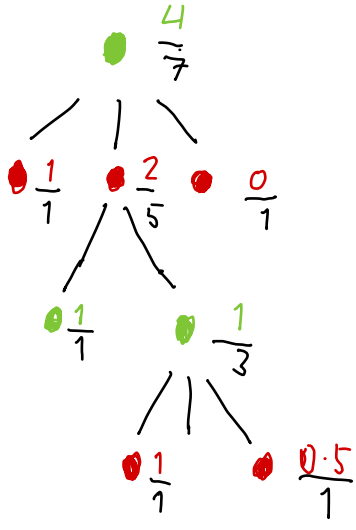
- Defining position evaluation
- Game tree grows exponentially with depth

Monte Carlo Tree Search (MCTS)

addresses both issues

- no evaluation function needed
- it carries out a random exploration focussing on parts of the tree that look promising

MCTS illustrated

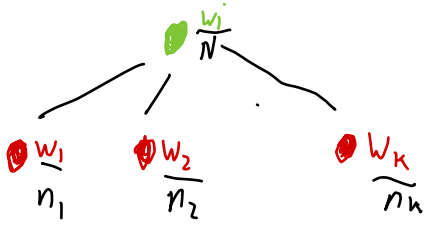


Run for a fixed number of iteration.

Each run of MCTS has 4 steps

1. Selection

From a fully expanded node (all children visited)



Pick child node i to maximise the value of

$$\frac{w_i}{n_i} + C \cdot \sqrt{\frac{\ln(N)}{n_i}}$$

where $N = n_1 + \cdots + n_k$

$w_i = n_i - w_i$

C: A chosen constant $\geq \sqrt{2}$

A larger C leads to a more balanced search

A smaller C leads to a more focused search.

Theory suggests: $C = \sqrt{2}$

(In practice tune C)

Selection determines path to a non-fully-expanded node

2. Expansion

From a non-fully expanded node randomly choose one child that has not been visited and add that node to tree

3. Simulation

From a leaf of the current tree play a random game and give the node the relevant value 0/1

4. Back propagation

Go back along path to root and update score

After a fixed number of runs choose child node i with maximum value of n_i

(In practice this maximises $\frac{w_i}{n_i}$)

This is the claim move.

To implement in Java

- HashMap < TreeIndex , TreeEntry >

TreeIndex

essentially List<Potez>

Care util hashCode

Make sure indices are immutable.

TreeEntry:

Fields	Iga	game position
	int	visits
	double	wins