TECHNISCHE UNIVERSITÄT DRESDEN

Fakultät Informatik
Institut Künstliche Intelligenz
General Game Playing

## Content

Program Design

Game Analysis

Visualizer

Singleplayer

Multiplayer

# laikLee

Christoph Möbius
André Viergutz
Robert Willner

TECHNISCHE UNIVERSITÄT DRESDEN

Fakultät Informatik
Institut Künstliche Intelligenz
General Game Playing

Program Design

Game Analysis

Visualizer

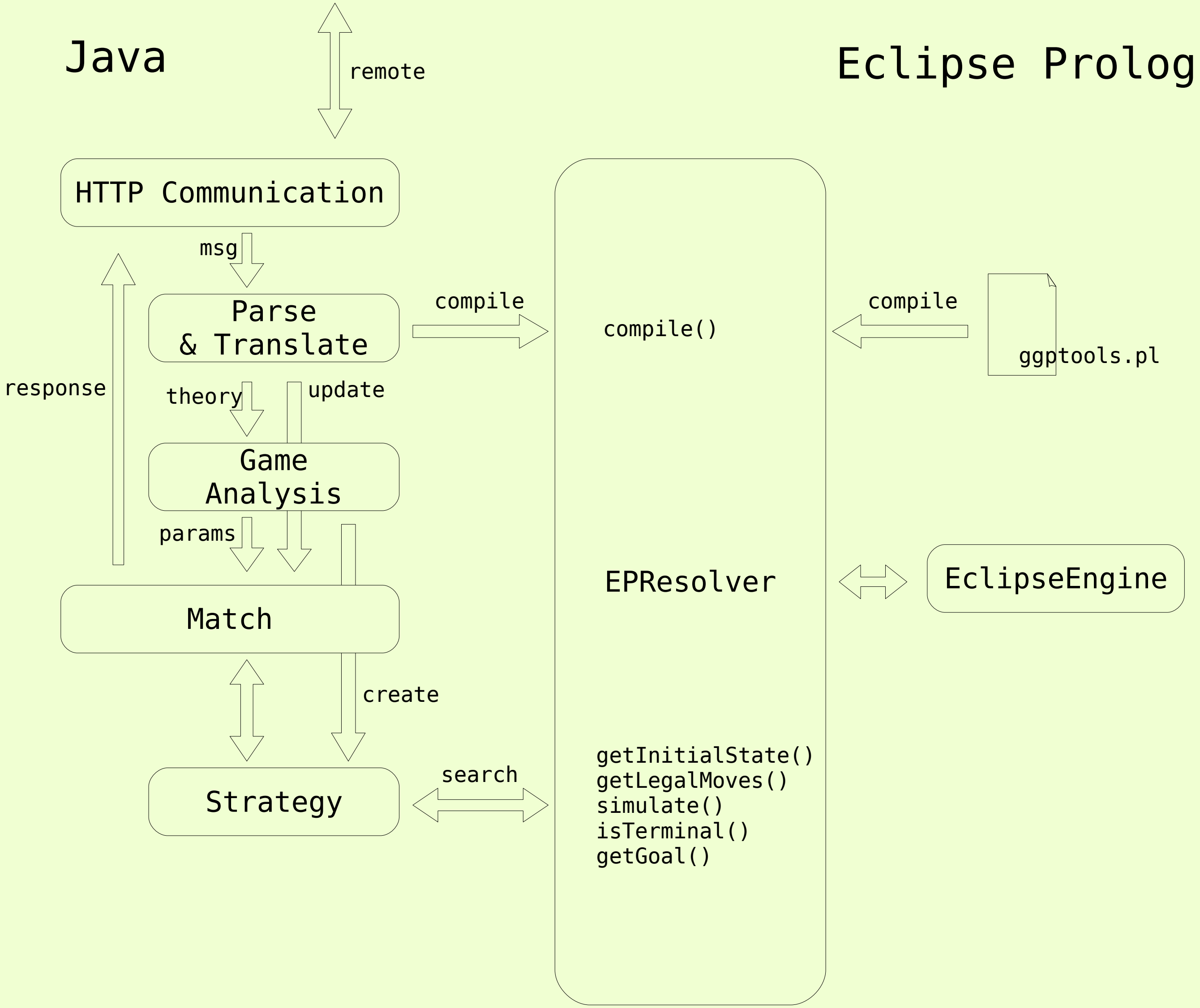
Singleplayer

Multiplayer

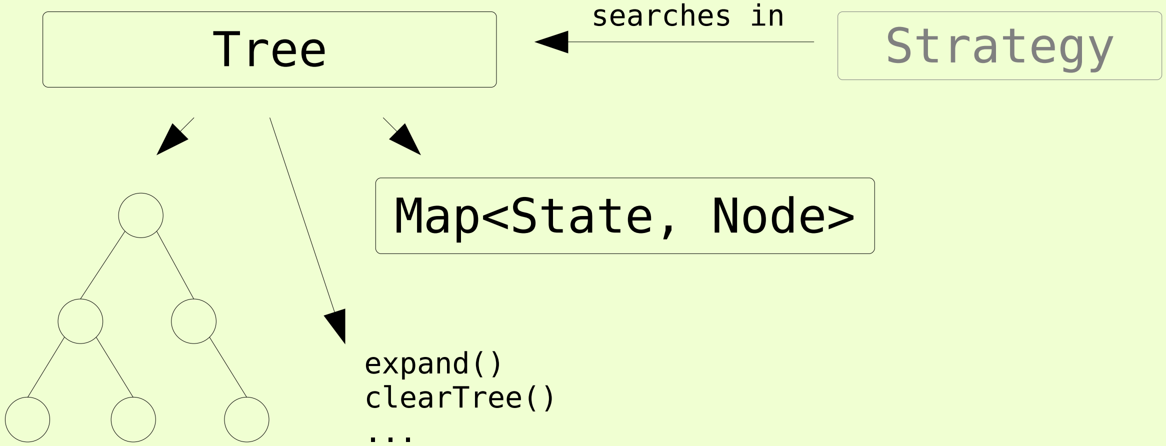
Heuristics

# laikLee

Christoph Möbius
André Viergutz
Robert Willner



Using data structure for tree to get fast access to nodes


Tree

searches in

Strategy

Map<State, Node>

expand()
clearTree()
...

Complicated clearing routines needed due to growing heap. At the time only possible to clear whole tree.

Program Design

Game Analysis

Visualizer

Singleplayer

Multiplayer

Heuristics

Able to determine the domain included minimum and maximum element

Finding successor relation

Finding step counter(s) at present but not implemented yet to deal with

Program Design

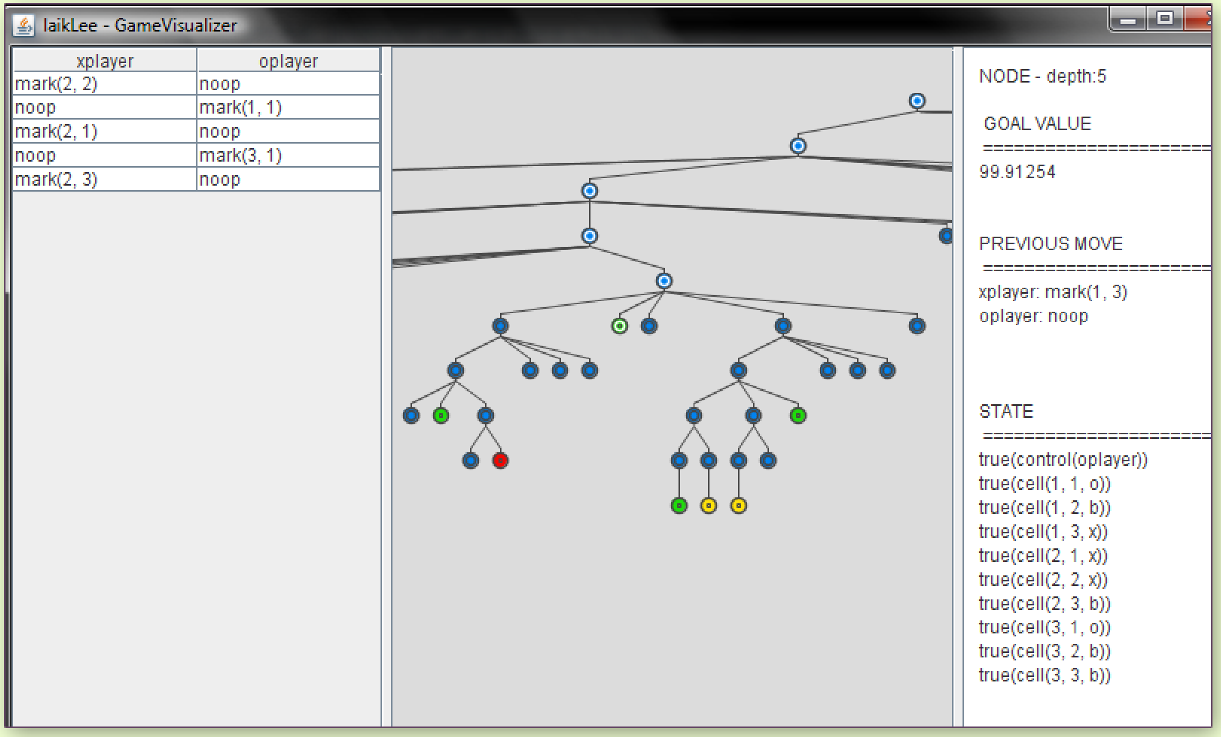
Game Analysis

Visualizer

Singleplayer

Multiplayer

Heuristics



laikLee - GameVisualizer

| xplayer | oplayer |
|---|---|
| mark(2, 2) | noop |
| noop | mark(1, 1) |
| mark(2, 1) | noop |
| noop | mark(3, 1) |
| mark(2, 3) | noop |

NODE - depth:5

GOAL VALUE
======================
99.91254

PREVIOUS MOVE
======================
xplayer: mark(1, 3)
oplayer: noop

STATE
======================
true(control(oplayer))
true(cell(1, 1, o))
true(cell(1, 2, b))
true(cell(1, 3, x))
true(cell(2, 1, x))
true(cell(2, 2, x))
true(cell(2, 3, b))
true(cell(3, 1, o))
true(cell(3, 2, b))
true(cell(3, 3, b))

- can display the complete structure of the game tree
- also displays information stored in our search tree

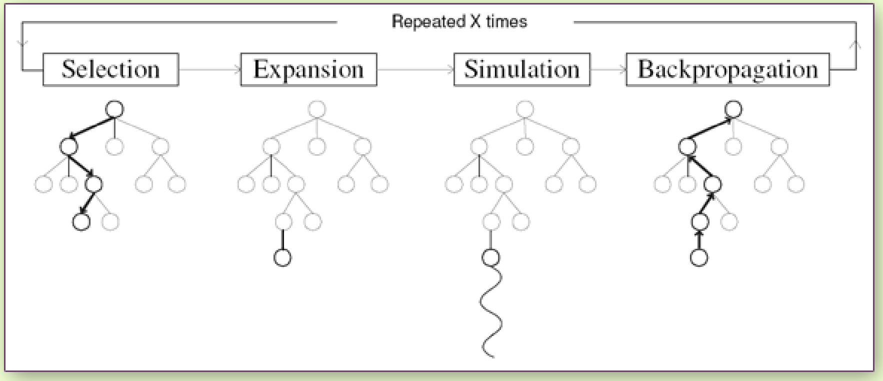
- very useful for debugging the tree-structure

Iterative deepening search

Saving information about best move and maximum evaluation of children per state

Very space consuming, but so far only solution, because of missing evaluation

- UCB1 (Upper Confidence Bounds) for rollout-based Monte-Carlo planning
- builds its lookahead tree by repeatedly sampling games from the current state



- the selection function is applied until a leaf node is reached
- one node is created
- play one simulated game
- the result of this game is backpropagated in the tree

- selection function controlls balance between exploitation and exploration

Program Design

Game Analysis

Visualizer

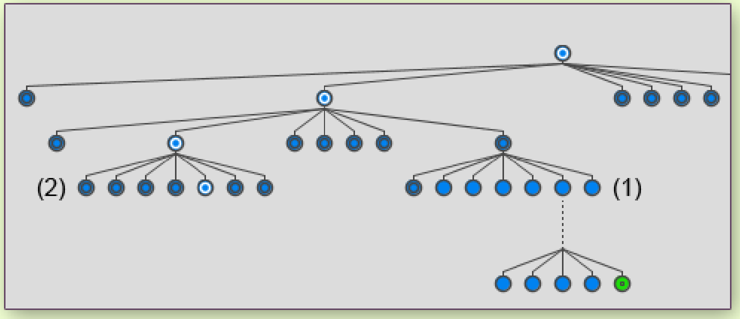
Singleplayer

Multiplayer

Heuristics

(1) selection function prefers unexplored nodes

(2) if all children nodes are explored select a node with the highest UCT value



$$\text{UctValue} = \text{node value} + \text{CONST} * \sqrt{\frac{\ln(2 * \text{parentAttendCount})}{\text{nodeAttendCount}}}$$

Example

$$100 + 50 * \sqrt{\frac{\ln(2 * 500)}{5000}} = 101{,}85846$$

$$0 + 50 * \sqrt{\frac{\ln(2 * 500)}{1}} = 131{,}41304$$

...May the force be with us

Program Design

Game Analysis

Visualizer

Singleplayer

Multiplayer

Heuristics

# Thank you!

- Kocsis, Szepesvari: Bandit Based Monte-Carlo-Planning, http://zaphod.aml.sztaki.hu/papers/ecml06.pdf
- Schiffel, Thielscher: Fluxplayer, http://www.fluxagent.org/download.php
    ?file=07-SchiffelThielscher-AAAI.pdf