General Game Playing (GGP)
Winter term 2013/2014

7. Real GGP I
<table>
<thead>
<tr>
<th>Date</th>
<th>What will we do?</th>
</tr>
</thead>
<tbody>
<tr>
<td>22.10.2013</td>
<td>Introduction, Repetition propositional logic and FOL</td>
</tr>
<tr>
<td>29.10.2013</td>
<td>Repetition FOL / Datalog and Prolog</td>
</tr>
<tr>
<td>05.11.2013</td>
<td>Game Description Language</td>
</tr>
<tr>
<td>12.11.2013</td>
<td>Design of GDL games</td>
</tr>
<tr>
<td>19.11.2013</td>
<td>Search Algorithms 1</td>
</tr>
<tr>
<td>26.11.2013</td>
<td>No lecture</td>
</tr>
<tr>
<td>03.12.2013</td>
<td>Search Algorithms 2</td>
</tr>
<tr>
<td>10.12.2013</td>
<td>Real GGP 1</td>
</tr>
<tr>
<td>17.12.2013</td>
<td><strong>Midterm competition</strong></td>
</tr>
<tr>
<td>14.01.2014</td>
<td>Real GGP 2</td>
</tr>
<tr>
<td>21.01.2014</td>
<td>Meta-Gaming</td>
</tr>
<tr>
<td>28.01.2014</td>
<td>Game Theory</td>
</tr>
<tr>
<td>04.02.2014</td>
<td><strong>Final Competition</strong></td>
</tr>
<tr>
<td>11.02.2014</td>
<td>Exam</td>
</tr>
</tbody>
</table>
Midterm Competition

• Two virtual machines
• Access via SSH
• Each machine has
  – 8 cores
  – 16 GB of RAM
  – 50-100 GB local memory
• Accessible starting from Thursday (I hope)
  – Details will be announced via Goya
• One more restriction for both competitions:
  – We will only play single-player and two-player games!
Midterm Competition

• General procedure:
  – Your GamePlayer(s) is started on the virtual machine
  – GameServer (from GGP-Galaxy project) is started
  – GameServer connects to your implementations and sends game description
    • See lecture on GDL for protocol
  – Game is run until termination

• Please check whether your client adheres to the GGP-protocol beforehand, not only at 17th December!
Midterm Competition

• How many groups/students will take part?
Oral exam

• Two dates:
  – 11\textsuperscript{th} February 2014
    • 9:00, 9:30, 10:00, 10:30, 11:00, 11:30 (RUD 26, 1.307)
    • 12:30, 13:00, 13:30, 14:00, 14:30, 15:00 (RUD 25, 4.406)
  – 25\textsuperscript{th} February 2014
    • 9:00, 9:30, 10:00, 10:30, 11:00, 11:30 (RUD 25, 4.406)
    • 12:30, 13:00, 13:30, 14:00, 14:30, 15:00 (RUD 25, 4.406)

• Registration
  – Starting from 13\textsuperscript{th} January 2014
  – Make an appointment by mail
  – First come, first served

• Exam duration
  – 30 minutes
Repetition: Minimax
Repetition: Minimax

What can be pruned?
Two-bit pattern databases

• There really is a technical glitch with searching two-bit pattern databases: if there exists different paths to the goal with the same length modulo 3, then the heuristic value is not admissible anymore.

• Two-bit pattern databases:

• The solution might be hidden somewhere here:
  – http://citeseerx.ist.psu.edu/viewdoc/download;jsessionid=0AAC69B31F0EBEDD0A8136AC2D31E694?doi=10.1.1.38.4710&rep=rep1&type=pdf

• If you are really interested, dig through the paper (not relevant for the exam)
Today

• Today we will look at the following GGP programs:
  – Winner 2005:
    • **Cluneplayer** by James Clune (UCLA)
  – 2nd place 2005:
    • **Ogre** by David M. Kaiser (Florida International University)
  – Winner 2007 and 2008:
    • **Cadiaplayer** by Yngvi Björnsson and Hilmar Finnsson (Reykjavik University)

• Next time:
  – Winner 2006:
    • **Fluxplayer** by Stephan Schiffel and Michael Thielscher (TU Dresden)
  – More recent winning programs
Cluneplayer
Cluneplayer

• Key idea:
  – Abstract the game to its core aspects
  – Compute the exact value of the simplified game (as with MiniMax)
  – Use exact values to guide search in real game

• Core aspects:
  – Expected payoff
  – Control (relative mobility)
  – Expected game termination (game longevity)
Cluneplayer: Identifying General Structures

• When a human first learns the game of chess, there are some obvious metrics that appear relevant to assessing states:
  – Size of the board
  – the relative cardinalities of regular pieces of each color and kings of each color.
• The more sophisticated player may consider other factors as well, such as
  – the number of openings in the back row,
  – degree of center control, or
  – Distance to promotion.
• Definition:
  – **feature** refers to a function from states to numbers that has some relevance to assessing game states.
• A feature is not necessarily a full evaluation function, but features are building-blocks from which evaluation functions can be constructed.
Cluneplayer: Identifying General Structures

• GDL game description, although isomorphic to the physical game, has none of these properties, so feature identification is not so simple.

• Approach:
  1. Extract candidate expressions
  2. Impose interpretations on expressions
  3. Identify relevant features based on stability
Cluneplayer: Identifying General Structures

- Foundation: a routine that analyzes variable replacements (i.e. constants) for each argument of each expression
- If a domain of a variable has a few constants only, then perform grounding
- Other expressions are ignored or grouped by trying to identify distinctness, e.g. border rows have no neighbors
Cluneplayer: Interpretation 1

• Solution cardinality
  – The idea is that we take the given expression, and find the number of distinct solutions to this expression there are in the given state. In the case of the expression (cell ?x ?y black queen), the number of solutions corresponds to the number of black queens on the board.
Cluneplayer: Interpretation 2

- **Symbol distance**
  - Binary relations among symbols in GDL are game specific relations with arity two. For example, the chess description utilizes a game-specific next rank relation to establish ordering of rows. Another example: artificial turn counter
  
  - Approach:
    1. Construct a graph of game-specific symbols appearing in the description, where each constant symbol is a vertex in the graph. Edges are placed between vertices that appear together in the context of a binary relation in the game description.
    2. Once this graph is constructed, the symbol distance between two symbols is the shortest path between the two symbols along this graph.
Cluneplayer: Interpretation 2

• Symbol distance
• Example
  – (cell ?x rank8 red_piece)
  – Constants: rank8
  – Replace with variable: (cell ?x ?y red_piece)
  – Suppose we have two solutions
    • (cell a rank2 red_piece) and (cell f rank5 red_piece)
  – Distance is 6 and 3
  – Overall abstract distance is 3
Cluneplayer: Interpretation 3

• Partial solution
  – This interpretation only applies to compound expressions involving multiple conjuncts or disjuncts. The partial solution interpretation of the conjunction results in a number that is proportional to the fraction of conjuncts satisfied

• Example:
  – In Connect-4, the rule for winning involves a conjunction of four pieces of the same color in a row
  – Evaluation for three in a row: 75%
Cluneplayer: combining interpretations

• For each candidate expression all interpretations are applied
• Finally, some additional features are generated by observing and exploiting symmetry in the game description (not covered here)
Cluneplayer: Feature Selection

- Which features are relevant to state evaluation?
- The intuition behind the criteria is that
  - quantities which **wildly oscillate** do not provide a basis for assessing the value of a state
  - quantities that **vary only incrementally** are much better.

=> Introducing of a measure called the **stability**.
- How to do that?
Cluneplayer: Feature Selection

• Computation:
  – Generate sample states by random exploration of the game tree and assess the value of each feature in each state. Compute variance over all states => total variance
  – Two pairs of states are adjacent if one is the immediate successor of the other in some path through the game tree.
  – adjacent variance is computed by summing the squares of the difference in feature values for adjacent sample states and dividing by the number of adjacent state pairs.
  – The stability quotient (stability) is the ratio of the total variance to the adjacent variance.
  – If the feature wildly oscillates from state to state, the stability will be low (roughly 1), whereas if the feature value changes only incrementally, the stability will be significantly greater than one.
Cluneplayer: Abstract model

- The abstract model reduces the game to five parameters:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>( P : \Omega \rightarrow [0, 100] )</td>
<td>Approximates payoff function.</td>
</tr>
<tr>
<td>( M : \Omega \rightarrow [-1, 1] )</td>
<td>Relative mobility.</td>
</tr>
<tr>
<td>( T : \Omega \rightarrow [0, 1] )</td>
<td>Proximity to termination.</td>
</tr>
<tr>
<td>( S_P : [0, 1] )</td>
<td>Relative stability of ( P ).</td>
</tr>
<tr>
<td>( S_M : [0, 1] )</td>
<td>Relative stability of ( M ).</td>
</tr>
</tbody>
</table>
Cluneplayer: P - Correlation between stable features and payoff

- Correlation types:
  - positive, negative, not correlated

- Create pseudo-states with $+L$ and $-L$ and compute their evaluation under the assumption that they are terminal states
  - If the payoff values are the same, the feature is considered uncorrelated with payoff.
  - If the payoff is higher on the state with the higher feature value, then the correlation is positive.
  - If the payoff is lower on the state with the higher feature value, then the correlation is negative.

- Only keep features with non-zero correlation
- Removing some subsuming features
Cluneplayer: P - Correlation between stable features and payoff

• When there are multiple features that are correlated with payoff and not subsumed by other features, the features are weighted according to their stability, with the coefficient being positive for positively correlated features and negative for negatively correlated features.

• Finally, the overall coefficient and offset are set such that the values of the resulting payoff function for all the sample states fall in the range of [0, 100]
Cluneplayer: M – Mobility function

- The mobility function is intended to quantify differences in the number of moves available to each role.

\[
\frac{m_{\text{red}}(\omega) - m_{\text{black}}(\omega)}{\max_{\omega' \in \Omega}(m_{\text{red}}(\omega') + m_{\text{black}}(\omega'))}
\]

- Positive numbers indicate red has more moves, negative numbers indicate black has more moves. In games with more than two roles, \( m_{\text{black}} \) is replaced with the sum of the number of moves of the adversaries.

- The denominator cannot be measured directly because the state space is too large, so it is approximated by taking the maximum quantity over the sample states.
Cluneplayer: M – Computation of mobility function

• Start with stable features
• Eliminate features that do not influence legal actions
• Remove absolute features subsumed by relative features.
• Example: in cylinder checkers two features left:
  – the number of red pieces minus the number of black pieces and
  – The number of red kings minus the number of black kings.
• To quantify the relative contribution of these features, a collection of sample states is generated by simulating game play with random moves. Least squares regression is performed with the control as the dependent variable and the features as the independent variables to find the best fit for control values in terms of the features.
Cluneplayer: T - Termination

- Determine the relative importance of the payoff and control functions, the intuition being that in the opening it may make sense to attempt to gain more control of the game’s trajectory, but that in the endgame focusing on the payoff is paramount.
- Treating termination as probabilistic is perhaps counter-intuitive, given that the termination of each state is in fact deterministic.
- The probabilistic treatment enables to abstract over a region of states with similarly valued stable features. The termination function is computed statistically by least squares regression, with the target values 1 for terminal states and 0 for non-terminal states.
Cluneplayer: Heuristic evaluation function

- How to combine P, M, and T?
- Consider the game as compound lottery
  - With probability T, the game terminates and red is awarded payoff P.
  - With probability 1 − T, the game continues and has an outcome determined by a second lottery.
  - The second lottery has two possible outcomes.
    - The first outcome, with probability SM is a payoff to red of 100 * M.
    - The second outcome, with probability SP = 1 − SM is a payoff to red of P. The values SP and SC are the payoff stability and mobility stability, respectively.

\[ v = T(\omega) \times P(\omega) + (1 - T(\omega))(50 + 50M(\omega)) \times S_M + S_P \times P(\omega) \]
Cluneplayer: Heuristic evaluation function

\[ v = T(\omega) \cdot P(\omega) + (1 - T(\omega))(50 + 50M(\omega)) \cdot S_M + S_P \cdot P(\omega) \]
Cluneplayer: Racetrack Corridor

http://130.208.241.192/ggpserver/public/view_game.jsp;jsessionid=EC4EBE80B7418EE359D7A8A4D52843F1?name=racetrackcorridor
Cluneplayer: Racetrack Corridor

Payoff = 50 + 7.5 (goal distance: black - white)

Mobility =

0.040 (# walls on left side of white lane
- # walls on left side of black lane)
+0.039 (# walls on right side of white lane
- # walls on right side of black lane)

Terminal = 0.016 (current step #) - 0.102

Payoff Stability = 0.732, Mobility Stability = 0.268
Cluneplayer: Chess

• One hour of analysis
• No payoff function found

\[
\text{Mobility} = \\
0.060 \ (\# \ white \ queens - \ # \ black \ queens) \\
+0.035 \ (\# \ white \ rooks - \ # \ black \ rooks) \\
+0.027 \ (\# \ white \ bishops - \ # \ black \ bishops) \\
+0.017 \ (\# \ white \ knights - \ # \ black \ knights) \\
+0.0031 \ (\# \ white \ pawns - \ # \ black \ pawns)
\]

• Usual rating of chess figures:
  – 9 queens, 5 rooks, 3 bishops, 3 knights, 1 pawns
Cluneplayer: Anytime sampling algorithm

- N=25
- LOOP
  - Compute the model for N sample states
  - N=N*2
- UNTIL no more analysis time
- Return results for best computed model
Cluneplayer: Overall search algorithm

• Iterative-deepening MiniMax search with (alpha-beta pruning, transposition tables, and aspiration windows)
• When evaluating nodes at the frontier, terminal nodes are valued according to their actual payoff values as determined by the game description.
• To evaluate non-terminal nodes, we first compute values for P,T,M, SP, SM, by evaluating the functions constructed in the analysis phase. Compute v. These values are propagated according to the minimax algorithm and are used as the basis for move selection.
Cluneplayer: tweaks

- **Transposition tables**
  - States can be reached by several ways
  - Avoid evaluating same position several times
  - Use hash tables for storing evaluations
    - Before a position is considered for evaluation, the table is consulted

- **Aspiration windows**
  - Heuristic for Minimax
  - [http://en.wikipedia.org/wiki/Alpha%E2%80%93beta_pruning#Heuristic_improvements](http://en.wikipedia.org/wiki/Alpha%E2%80%93beta_pruning#Heuristic_improvements)
Cluneplayer

• Multi-player strategy: “paranoid assumption”
  – Other players form a coalition against me
  – They know my move beforehand at simultaneous-move games
  – Seek to minimize my payoff, instead of maximizing theirs
Cluneplayer: Single Player Games
Cluneplayer: Single Player Games

- Only payoff function is constructed (no mobility and no termination)
- Random walks down the game tree; only the best plan is recorded (in case no heuristic can be found)
- Only 33% of the time is used for evaluation function construction, the rest used for searching
- Informed and uniformed search in parallel
- During gameplay:
  - Minimum lookahead search
  - Depth-first search
Ogre
OGRE

• OGRE attempts to generate an efficient evaluation function by examining the syntactic structure of the game definition as well as dynamic features that appear in the game during a self play stage.

• Features recognized solely from the game definition include the dependency graph, static predicates, successor functions, and turn counters.

• Features discovered through self play include pieces and board position.

• Overall search algorithm: again MiniMax with alpha-beta-pruning
OGRE: Speed up reasoning by identification of static predicates

- Static predicates only need to be resolved one time, because they don’t rely on `true` or `does`
- More inferences in a given amount of time
OGRE: Feature extraction – Turn counters

- GDL games are guaranteed to end in a finite number of turns. Many GDL games achieve this by using a turn counter. These turn counters are particularly vexing because game states that might otherwise be identical appear unique when there is a turn counter.
  - (true (puzzle 1 b 2 3 4 5 6 7 8)) (true (turn 14))
  - (true (puzzle 1 b 2 3 4 5 6 7 8)) (true (turn 19))

- Equality of the two states?
- Also important for transposition table
- Pattern:
  
  (--- (NEXT (<turn> <varY>))
  
  (TRUE (<turn> <varX>))
  
  (<successor> <varX> <varY>))
OGRE: Feature extraction – Turn counters

- GDL games are guaranteed to end in a finite number of turns. Many GDL games achieve this by using a turn counter.
  - (true (puzzle 1 b 2 3 4 5 6 7 8)) (true (turn 14))
  - (true (puzzle 1 b 2 3 4 5 6 7 8)) (true (turn 19))
- Why is the (explicit) identification of turn counters important?
OGRE: Feature extraction by variance

- Initial fact groups: cell/3, control/1, step/1

<table>
<thead>
<tr>
<th>Args</th>
<th>N</th>
<th>Mean</th>
<th>Col. Var.</th>
<th>Sort Sequence</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cell/3</td>
<td>arg1</td>
<td>592</td>
<td>8</td>
<td>0</td>
</tr>
<tr>
<td>Cell/3</td>
<td>arg2</td>
<td>592</td>
<td>8</td>
<td>0</td>
</tr>
<tr>
<td>Cell/3</td>
<td>arg3</td>
<td>592</td>
<td>5.14</td>
<td>92.50</td>
</tr>
<tr>
<td>Control/1</td>
<td>arg1</td>
<td>74</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>Step/1</td>
<td>arg1</td>
<td>74</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>Pawn/1</td>
<td>arg1</td>
<td>6</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>Pawn/1</td>
<td>arg2</td>
<td>6</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>Moved/3</td>
<td>arg1</td>
<td>103</td>
<td>1.09</td>
<td>0.08</td>
</tr>
<tr>
<td>Moved/3</td>
<td>arg2</td>
<td>103</td>
<td>1.64</td>
<td>0.22</td>
</tr>
<tr>
<td>Moved/3</td>
<td>arg3</td>
<td>103</td>
<td>1.23</td>
<td>0.17</td>
</tr>
<tr>
<td>Check/4</td>
<td>arg1</td>
<td>2</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>Check/4</td>
<td>arg2</td>
<td>2</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>Check/4</td>
<td>arg3</td>
<td>2</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>Check/4</td>
<td>arg4</td>
<td>2</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>
OGRE: Motion Detection

- Try to identify moving symbols from sorted fact groups
- The motion detection algorithm performs a comparison operation on two sequential game states. Symbols that appear in the same location in both states are ignored. Symbols that change or “move” are identified and retained.

```
(step 1)    vs (step 2)    => (. 2)
(control white) vs (control black) => (. black)
(cell a 1 wr) vs (cell a 1 wr) => (....)
(cell a 2 wp) vs (cell a 2 b) => (.... b)
(cell a 3 b) vs (cell a 3 wp) => (.... wp)
/* all others resolve to (....) */
```
OGRE: Piece identification

Play $m$ Random Games, store game history $H$
Identify Fact Groups $G$
For every fact group $g$ in $G$
    For every argument $a$ in $g$
        For every symbol $x$ in $a$
            Calculate variance $v$ of $x$ in $a$
        End For
    End For
End For
Determine sort order for $g$
End For
For each game state $s$ in game history $H$
    For each fact group $g_i$ in $s$
        Sort $g_i$
        /* Detect motion */
        Compare $g_i$ with $g_{i+1}$
    End For
End For
OGRE: Piece identification

<table>
<thead>
<tr>
<th>symid</th>
<th>symbol</th>
<th>argument</th>
<th>symbolvariance</th>
<th>symbolmean</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>wr</td>
<td>3</td>
<td>0.134054</td>
<td>2.162162</td>
</tr>
<tr>
<td>12</td>
<td>wp</td>
<td>3</td>
<td>3.127927</td>
<td>6.270270</td>
</tr>
<tr>
<td>19</td>
<td>bp</td>
<td>3</td>
<td>1.042162</td>
<td>6.567565</td>
</tr>
<tr>
<td>21</td>
<td>br</td>
<td>3</td>
<td>0.0</td>
<td>2.0</td>
</tr>
<tr>
<td>22</td>
<td>wn</td>
<td>3</td>
<td>0.243783</td>
<td>1.445945</td>
</tr>
<tr>
<td>23</td>
<td>bn</td>
<td>3</td>
<td>0.0</td>
<td>2.0</td>
</tr>
<tr>
<td>25</td>
<td>wb</td>
<td>3</td>
<td>0.211351</td>
<td>1.689189</td>
</tr>
<tr>
<td>26</td>
<td>bb</td>
<td>3</td>
<td>0.038378</td>
<td>2.040540</td>
</tr>
<tr>
<td>28</td>
<td>wq</td>
<td>3</td>
<td>0.0</td>
<td>1.0</td>
</tr>
<tr>
<td>29</td>
<td>bq</td>
<td>3</td>
<td>0.0</td>
<td>1.0</td>
</tr>
<tr>
<td>31</td>
<td>wk</td>
<td>3</td>
<td>0.142882</td>
<td>1.175675</td>
</tr>
<tr>
<td>32</td>
<td>bk</td>
<td>3</td>
<td>0.0</td>
<td>1.0</td>
</tr>
</tbody>
</table>
OGRE: Evaluation functions

• Game Structure evaluators
  – Distance-Initial (Run-Away)
  – Distance-ToTarget
  – Count-Pieces
  – Occupied-Columns
OGRE: Evaluation functions

- Game Definition evaluators (apply to a broad set of games)
  - Count-moves
  - Depth
  - Pattern
  - Purse
OGRE: Evaluation functions

• Combining evaluators into a single evaluation function
• First stage (50%):
  – identify all evaluation functions
• Second stage (50%):
  – find weights for evaluation functions
OGRE: Combining evaluators

```
SelectEvaluators
    FOR each evaluator E
        FOR each piece P
            Create instance of E (En) using P.
            Play one game using En weighted +10.
            IF win THEN
                add En to list L
            ELSE
                Play game using En weighted -10.
                IF win THEN
                    add En to list L
            ENDFOR (piece)
        ENDFOR (piece)
    ENDFOR (evaluator)
RETURN list of evaluators L
```
Cadiaplayer
Cadiaplayer: Representation of game tree
Cadiaplayer: Hashing

• To every atomic symbol a 64bit hash value is assigned
• Compound symbols are hashed by combination of atomic hashes for each argument
• Problem?
Cadiaplayer: Hashing

• To every atomic symbol a 64bit hash value is assigned
• Compound symbols are hashed by combination of atomic hashes for each argument
• Problem?
  – (cell 1 2 b) … (cell 2 1 b)
• => obtain the same hash value, but are clearly distinct
• Argument-based hashing:

```
Algorithm 1 getZKey(Compound C)
1: key ← keymap(C.symbol())
2: shift ← 7
3: for all \( c_i \in \text{args}(C) \) do
4:   key ← key \( \oplus \) rotate(getZKey(c_i), shift)
5:   shift \( \leftarrow \) (shift + 7) mod 64
6: end for
7: return key
```
Cadiaplayer: Action Buffer

- Map actions to identifiers
- Use identifiers in game tree
- Two advantages:
  - Saving space
  - Because the model now knows the available actions for any state that has been added to it, there is no need to query the Game Logic Interface for available actions, as all data needed to make a transition is stored in the Action Buffer.
Cadiaplayer: Single-player

• Memory Enhanced IDA*
• The search starts immediately during the start-clock. If successful in finding at least a partial solution (i.e. a goal with a higher than 0 point reward) it continues to use the algorithm on the play-clock, looking for improved solutions.
• If unsuccessful, the engine falls back on using the multi-player (UCT) algorithm on the play-clock.
• Note that Cadiaplayer is really multi-player GGP
Cadiaplayer: Multi-player

- Monte-Carlo simulation
- Upper confidence bounds applied to trees
Cadiaplayer: Monte Carlo methods

- Reinforcement learning without prior knowledge
- “Just” simulate a lot of games
- Computation of a MC-value function $\text{States} \times \text{Actions} \rightarrow \text{Score}$
- Each time a simulated episode reaches a terminal state the rewards are backed up to all state-action pairs included in the episode and are averaged into the estimated rewards.
Cadiaplayer: Monte Carlo methods

• Main problem:
  – Which actions are chosen? Exploitation vs. Exploration

• Solution:
  – UCT algorithm
Cadiaplayer: Basic UCT algorithm

- UCT selects the next action by:

\[ a_t = \arg\max_{a \in A_t} \left\{ Q(s, a) + C_p \sqrt{\frac{\ln N(s)}{N(s, a)}} \right\} \]

- The N function returns the number of visits to a state or the number of times a certain action has been sampled in a certain state, depending on the parameters.
- Cp is set to 40 in Cadiaplayer
Cadiaplayer: UCT algorithm
Cadiaplayer: problem

- Can you see the problem applying UCT if it is blacks turn?
Cadiaplayer: problem

• Can you see the problem applying UCT if it is blacks turn?
  – Black is unlikely to foresee the capture event, if the opponent is modeled with “random” sampling

• Solution:
  – History heuristic: Actions are assumed good, if they were good in other states
  – Storing $Q_h(a)$ in addition to $Q_h(s,a)$
Summary

• General game players are not about designing a single heuristic!
• Many heuristics are necessary, you “only” need to know which one to trust at run time
Preview

- 17\textsuperscript{th} December: Mid-term Competition
- 7\textsuperscript{th} January 2014: Real GGP 2
  - Fluxplayer, Turbo Turtle,…
Acknowledgements

• James Clune, “Heuristic Evaluation Functions for General Game Playing”
  – https://sites.google.com/site/jimcluneresearch/clune_dissertation.pdf?attredirects=0

• David Michael Kaiser, “The structure of games”
  – http://digitalcommons.fiu.edu/cgi/viewcontent.cgi?article=1002&context=etd

• Hilmar Finnsson, “CADIA-Player: A General Game Playing Agent”