SGPlayer

Ideas, Concepts and Implementation Details

(Georg Püschel and Stefan Schlott)

Basic Guidelines

- quick, stable and responsive (→ every single task in an own concurrent thread)
- expensive computations should be done only once (reuse where possible)
- only trust actual goal values instead of values based on easy heuristics, which can be unstable for different games
- concentration on multiplayer games (impact on final results may be bigger)

Used Environment

- PrologProver (provided by Stephan Schiffel)
- EclipseProlog (Java interface from Palamedes)
- JAVA with own objects for graph infrastructure, scores, …
- communication with GameController done using given Palamedes classes

Performance Issues / Problems

- use of HashMap/HashSet where possible instead of Lists
- try to keep reasoning to a minimum (profiler shows that reasoning takes about 80% of runtime)
- cut of expanded tree only if nothing else helps to clear memory
- cope with mostly unpredictable Full-GC (pauses all other VM-Threads → timeouts can happen)

Core of SGPlayer

- class SGPlayer – observes time, controls messaging and retrieves results from graph search (extends Player from Palamedes)
- classes Cache, SanitationDepartment (background thread) – manage all existing caches and controls player state according to memory consumption (mostly if too high)
- class MainControl – contains static variables (e.g. for internal player state, rolecount, ...)

SinglePlayer

- central cache of already expanded nodes (reuse saves memory and increases speed)
- SingleBestMovePredictor manages IDS-Threads and returns next moves to SGPlayer
- nodes with higher inferred goal value preferred (or random node returned, since no heuristics)
- IDS in own thread (could be used for later parallelism)
- IDS increases lookahead depth after each run

Multiplayer (1)

- central cache of expanded nodes and of computed legal moves (high amount of reuse makes MonteCarlo competitive)
- MultiBestMovePredictor manager IDS- and MonteCarlo-Threads and return next moves
- multiplayer search is done in 3 stages (Breadth-First, UCT-MonteCarlo, IDS)
- ability to return legal moves comes always first

Multiplayer (2)

- Phase 1:
  - Breadth-First Search with depth 1
- Phase 2:
  - MonteCarlo Search controlled by UCT -values
- Phase 3:
  - IDS based on backpropagated (discounted) goal values found during MonteCarlo phase
  - IDS prefers nodes with best and worst predicted outcome