* Simulated (non-rollout) games are not reliably finding captures for all roles for all piece types – can we fix this easily?
* Should rollouts avoid unforced losses even with greedy rollouts disabled?
* Measurement of greedy rollout effectiveness should include suppression of unforced losses
* Can/should different piece sets be given different weights (e.g. – kings vs pawns in checkers, or more significantly kings vs pawns in escort latch breach)
* Try piece count heuristic to weight playout move selection
* Try non-linear exploitation functions for expansion selection
* Can we use piece-heuristic-like generalized state mask heuristics – e.g. – the corners in Reversi?
* Experiment with move action histories now we know how to blend heuristic signals
* Move action histories to weight playouts too?
* Add game isomorphism detection
* Learn parameters such as:
  + Negatively correlated heuristics (relative to decision maker) separately from positively correlated (negative ones are currently not acted upon)
  + Exploration bias
  + Heuristic weights
  + Use of 1-level minimax
  + Playout heuristic weights (somewhat?) independent of UCT heuristic weights?
* Consider rejoining the split propnet with propagation coloring to control what part is propagated through
* Can we compile a propnet so that base props (and legal and goal) are simply directly executed logical functions of the base + move + base props? Java Assist. Would also need dependency maps so that given a base prop or move prop change we know which need recalculating
* Opponent modelling in especially MP games – can we detect random/poor players and exploit them?
* Analytic solution for iterated games (Nash mixed strat)
* ~~Simultaneous moves – weight average over children by selection strength??~~
* ~~Continue to think between turns!!~~
* ~~Fix selectLeastLikely() – needs separate trimming counter~~
* ~~Trim init off the propnet after retrieving intial state~~
* ~~Trim goals off main propnet and use full only after we know the state is terminal (test in Reversi!)~~
* ~~Do ANDs feed ANDS or OR feed ORs – if so probably best to collapse into large gates even if multiple input stages are involved (test)~~
* Can we trim some of the terminality logic based on lack of available moves – should work for some games (Checkers etc.)
* ~~Try dynamically varying the exploration bias as it thinks – either by striping a range or by gradual reduction through the turn play time~~
* Can we eliminate step counters from the stored state and police them on playouts directly (at least for state comparison purposes to get better transposition handling). Try this in 8-puzzle first. WORKS GREAT - need to automate
* Can we exploit symmetries, at least in puzzles, to have states considered equivalent for exploration purposes – especially thinking peg jumping with A\*
* If puzzle ever sees rollout score of 100 just store the path in the same way A\* does
* Figure out why knights tour (large) fails to build propnet
* ~~Nodes with a subset of children that are complete at mid-range scores can still complete the parent if the other nodes + their current exploration UCT cannot match their values (check in select())~~
* ~~Consider adding a cousin value to UTC exploitation even for non-simultaneous games (will need to extend to back to first opponent choice ancestor). Should provide crude move action history effect. C5 probably a good test game~~ [note – subsumed by action history work]
* ~~When a node is a decisive completion consider doing something to bump the selection probability of that moves in uncles~~
* Add a commit/rollback to the statemachine, using an 2-element output array with a committal index switching between the two. Should reduce the cost of making multiple calculations (successor states typically) from a single parent sequentially since the cost to back out the move will be removed
* Use move weights (diluted?) in playouts
* Make sample size dynamic based on rollout queue saturation
* Find a way to easily feed in a canned test-state (e.g. – from a point it was observed to make a bad move from Tiltyard) conveniently (rather than the current approach of creating a modified set of game rules containing that state as init manually, which is painful)
* Test cases – how about some means to feed specific canned states to a player with hand-crafted acceptable moves – such cases could be developed from mistakes it has been observed to make, and would make for very fast tests since we’re just looking for a single move calculation and checking it is one of some acceptable set
* Should add more possible heuristics to test for correlation in meta-gaming and hence potentially use - at least a mobility (num available move choices) heuristic
* Consider processing the goal net to extract a ‘partial completeness’ metric for use as a heuristic. For example if the ‘win’ goal (with unclear scoring may need a weighted average for each goal prop) is an OR of things that are themselves ANDs (considered shallowly this will always be the case at the end of any goal network) then evaluate as some sort of weighted average (perhaps just max, but more likely max + 0.5\*<second best> + … or something) of the number of and AND’s inputs that are true. To make this really effective however we’d need to spot latches so that and AND which has a 0-latched input always contributed 0 (e.g. – 3 reds in a line in conect4 when the 4th spot is already occupied by black)