3 experiments were conducted:

1) A rich initial game state and a trivially easy target game state
2) A minimal initial game state and a moderately large/ distant target game state
3) A minimal initial game stat and an extremely large/ distant target game state

Each A* implementation was run 5 times, and the real world calculation time was measured and averaged. Also, the quality of the provided path was noted in the form of "Game Turns", which represents the distance between the initial game state and the target game state. A smaller amount of turns taken is better.

While different implementations may produces slightly different results, there was never a trial run for a given implementation that produced a different answer. That is to say each algorithm was deterministic and produces the same output for the same input.

The Accordion A* algorithm is like the Memory Bounded A* in that it limits the number of nodes that can be put onto a priority queue. However, once the Accordion A* algorithm reaches its maximum memory bound the worst half of all the nodes currently in the queue are purged. The idea is that, on average the memory usage is comparable to a traditional Memory Bounded A*, but the Accordion A* allows for periods of more "risky" edge exploration. In this way, we hoped that the algorithm could find and hold onto branches that initially looked bad but turned out to be good. Unfortunately, these small experiments seem to suggest the Accordion strategy is slightly worse than a traditional Memory Bound.

Another major take away from these experiments is that there was no major performance difference between a "normal" A* algorithm and a memory bounded one. This could be for 2 reasons:

1) The heuristic used is very refined and considers several variables to determine the remaining distance
2) The "Game Space" is very open. The equivalent analogy for traditional pathfinding is that there are very few walls or dead ends

These two facts together make it easy for a default A* algorithm to move straight towards the goal on the fastest line, and therefore optimizations won't produce much performance gain.

More testing will be done over the next few days, including modifications to the heuristic as well as more A* modifications (like bidirectional search, Dynamic weighting, some variation of Jump Point Search) as development time allows. The complexity of the target game state will also be increased with the introduction of more types and complexity of buildings and resource types.

Experiment 1

Initial Game State	Stockpile	Per Turn Income
Gold	1000	0
Stone	1000	0
Wood	1000	0

Target Game Sate	Stockpile	Per Turn Income
Gold	1	1
Stone	1	1
Wood	1	1
Silver	1	1

Search Algorithm	Real Time (second)	In Game Solution (turns)
Normal A*	**0.0208**	64
Memory Bounded (200 nodes)	0.0215	64
Accordion (300 nodes)	0.0231	64
Iterative Deepening	0.0213	64

Experiment 2

Initial Game State	Stockpile	Per Turn Income
Gold	100	0
Stone	100	0
Wood	100	0

Target Game Sate	Stockpile	Per Turn Income
Gold	20000	450
Stone	20000	286
Wood	20000	199
Silver	20000	156

Search Algorithm	Real Time (second)	In Game Solution (turns)
Normal A*	3.283	6849
Memory Bounded (200 nodes)	3.256	6849
Memory Bounded (2000 nodes)	**3.227**	6849
Accordion (300 nodes)	3.765	6882
Accordion (900 nodes)	3.597	6865
Accordion (3000 nodes)	3.317	6849
Iterative Deepening	3.292	6849

Experiment 3 (note, the target game state income per turn is 5x larger that in Experiment 2)

Initial Game State	Stockpile	Per Turn Income
Gold	100	0
Stone	100	0
Wood	100	0

Target Game Sate	Stockpile	Per Turn Income
Gold	20000	2250
Stone	20000	1430
Wood	20000	995
Silver	20000	780

Search Algorithm	Real Time (second)	In Game Solution (turns)
Normal A*	25.000	20366
Memory Bounded (200 nodes)	24.239	20366
Memory Bounded (2000 nodes)	**23.563**	20366
Accordion (300 nodes)	23.721	20366
Iterative Deepening	24.678	20366