Patrick Merrill
May 13, 2011
CS730W

Snake: Artificial Intelligence Controller

Throughout the course we have discussed many different ways of representing and computing knowledge so that programs can be more effective and be applied to bigger and harder problems. Although out of all of the different methods and schemes to apply knowledge, none of them attempted to deal with time. If they did deal with time it was mainly based on assumptions or an explicit timing given for each action. However, time is an important and harsh factor that all agents must be able to handle if they are to bridge the gap from simulation to actual practice and function properly. Therefore I wanted to use this project opportunity to explore how the algorithms we discussed could hold up in a true real-time environment.

Problem Statement

The core focus of my project was to determine which algorithms would be more effective in a hard real-time environment. The domain in this case is the Snake Game, which will, in turn, attempt to identify an, or even the, algorithm that can not only play the game but compete with human players. The Snake Game is a classic arcade style game where it is a single-player game but the focus is to achieve the highest score possible thus competing with yourself and others.

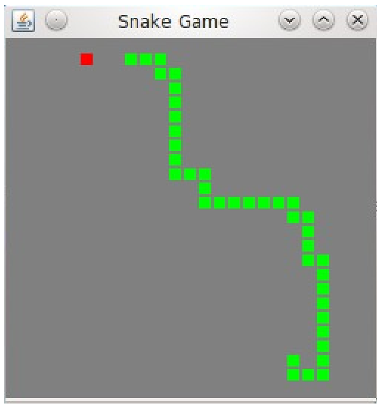


Illustration 1: the red square is the apple and the green line is the snake

To play the game one controls a snake by selecting one of the cardinal directions that the snake will move in. In order to score points you must direct the snake to an apple, there is only one apple in the game at time. The snake then eats the apple and increases in length by one square or unit. The game ends when the snake runs into either the boundaries of the play area or itself, the trailing snake body.

The domain provides a very interesting problem given that the snake always moves after a given timing delay and the snake continually grows in size. The delay is the feature that really makes the game difficult because if you do not react fast enough the snake will continue moving in the last direction given. This causes the player to try to act as quickly as possible before the snake runs into an obstacle. Also because the snake is constantly trailed by its tail(being the main obstacle) any move taken cannot be undone or immediately back tracked. So if you were to make a wrong turn into a dead end there is no way to reverse that move to back out of the loop.

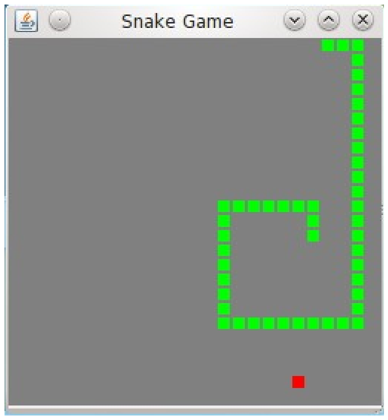


Illustration 2: In this situation the agent directed itself towards the apple but also a dangerous dead end.

Although with every move your tail moves thus creating a situation where a dead end or closed loop will no longer be a dead end after x moves. The apple also can be very tricky since you cannot know where the next apple position will be and this makes it difficult to go straight for the current apple without thinking of how to get into a good position for the next apple.

Possible Approaches

There are many possible approaches to this problem, given that the domain is well known and that the agent bases its decisions on a plan designed to guide the head of the snake to the apple. The baseline algorithms I used to judge my real-time algorithm implementations were A* and weighted A*. I chose A* because it has been proven to produce optimal solutions given an underestimating heuristic. Each of the algorithms I implemented used a heuristic to help guide them, which I left constant in order to have a less biased comparison. The heuristic was simply the manhattan distance of the snake head to the apple, which was only two integer comparisons, making it ideal for this problem because it is accurate and incredibly fast to compute. Other possible heuristics could involve the snake tail considering it is the main obstacle of the game. Such as weighting the movement decisions on how many tail lengths were in a given direction(i.e. would try to keep distance from its tail) or move towards larger open spaces. Those methods could decrease the chance of the snake getting caught in an inescapable closed loop, however, they would have also decreased the overall plan calculation time and all algorithms would suffer.

The only problem I saw A* having was meeting the time deadline. This issue arose because A* cannot commit to its first move until the optimal path is found. Therefore if the play area is too big, where the apple is N moves away from the head of the snake(i.e. a solution depth is N) and N > than maximum depth reached by A* within the given time limit, or the snake tail obstruction to the apple is highly significant than A* will not finish calculating fast enough and run right into an obstacle. If A* is lucky, however, the snake may continue straight and get close enough to the apple to latch on like a magnet and continue the game.

With knowledge of this the possible long calculation time I also implemented Weighted A* to allow for less optimal solutions that could speed up the algorithm. I made the weighted A* algorithm with an anytime component, where once a solution is found it commits to a move then continues calculation in hopes of finding a more optimal solution and then commit to a different move(Likhachev et al., 2008). Also it is important to note that these as well as the other algorithms I will discuss are run every time the snake takes a move. That is incredibly important because every time the snake moves, the state of the world changes and if calculation does not complete in time the old plan is invalid given that the snake is no longer in that initial state.

My Approach

My approach to the problem became two different algorithms one with lookahead the other without, both of which I based on a paper on RTA* entitled "Real-Time Heuristic Search" by Richard E. Korf. The main aspect I wanted the algorithm to have was a way of committing to a move even if a solution had not been found, thus creating an true anytime algorithm that, when given only a thousandth of a second and no time to look for the apple, will still avoid an obstacle if possible. The parts of RTA* I used were minimin and alpha-pruning which are common approaches to single-player games given that there are no opponents but the overall benefits remain the same. Minimin is a method of lookahead where the algorithm searches through the state space and for every milestone or new depth the best looking child is chosen. Then the algorithm commits to a move that will lead to that child and continue searching. This allows for committing to a first move even without having a solution.

Alpha-pruning was just used to speed up the search by pruning all nodes that have a higher f value than the current best child. This is based on the concept that, as the heuristic is underestimating and will not cause the f-values to decrease along a path, then a node with a higher f-value cannot lead to a better solution. RTA* is also meant to be used with an underlying algorithm that the lookahead and pruning will benefit, in this case I used A* again to keep the comparison as unbiased as possible. Also if the optimal solution can be found in time, then the algorithm will benefit from the optimization A* offers.

The aspects of RTA* I did not use were backtracking and learning only given knowledge of the domain. Backtracking is used in RTA* because it is understandable that if a move is made before an optimal solution is found that it is possible that the last state had a better solution path that is worth the cost of returning to that state. In this domain you cannot backtrack immediately, otherwise you would crash and the only other way to return to a state would take more moves to get the snake head and tail in the right place that it is highly unreasonable and unlikely. Learning of the kind that LRTA* and LSS-LRTA* perform is not useful in this case, given the simplicity of achieving one apple at any given time(Koenig and Sun, 2009). Since any learning made will be obsolete after an apple is eaten, even if the apple were to end up in an old location, the snake has grown since then and will remove the possibility of returning to states that were expanded or generated the last time the snake was in that location.

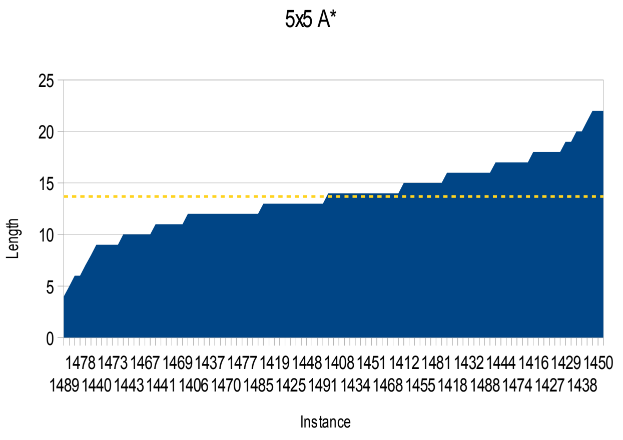
The other algorithm that resulted from my application of RTA* was a greedy algorithm, where it does not do any lookahead beyond the initial state's immediate children. I implemented this algorithm just as another comparison, though it actually had some interesting aspects given that the first "instinct" of the algorithm might be better than what lookahead could imply. Also it can function under the harshest of time limits and still have the same results under the same random seeding of apples. Thus allowing this algorithm to handle the biggest and fastest of games, though my version of RTA* could potentially do just as well.

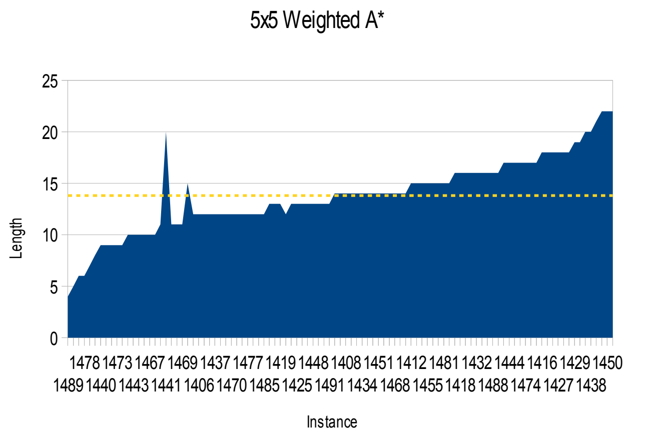
Evaluation

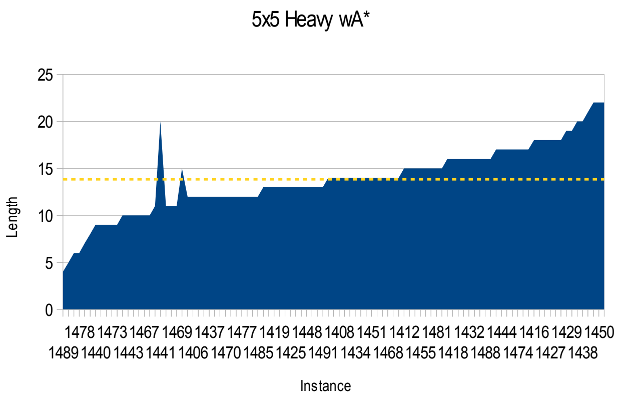
In order to evaluate all of the algorithms I used 101 seeds starting from a randomly selected number, in this case 1400(going to 1500), in order to achieve an average performance of each. Using the same seeds also allows for individual comparisons of each of the algorithms on each specific game. I then ran each of the algorithms across those seed values while also using varying domain arrangements such as speed or delay of the snake and the size of the game area to determine how the algorithms react to each combination. The speeds I used were a snake movement every 50 milliseconds and one movement every 10 milliseconds, thus giving the algorithms computation time of a difference of 5 times. The game area sizes were a small 5 by 5, 25 by 25, and 50 by 50 square space.

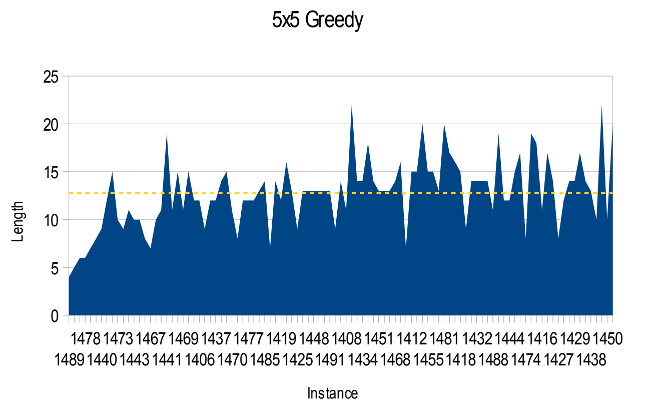
The results of the tests are in the graphs as follows. Note that the graphs measure the length of the snake for each instance in a given situation, where length is synonymous to apples eaten and score.

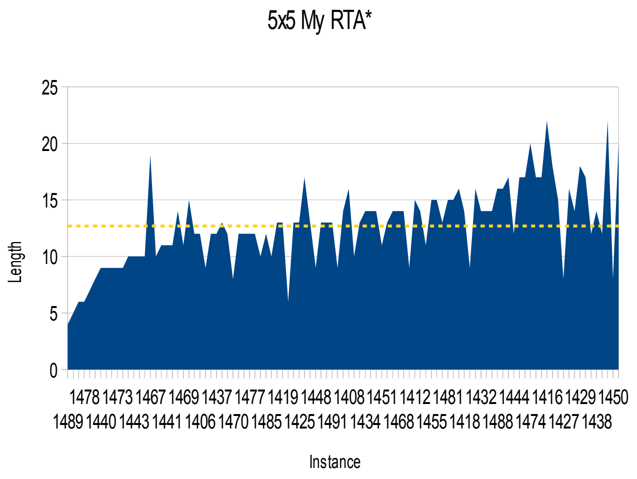
Also weighted A* was ran twice for two different weights, 1.5 and 5, heavy wA* being the latter:



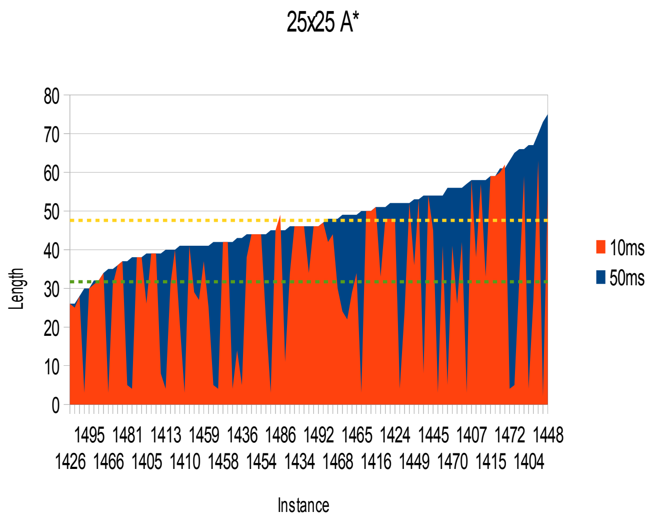


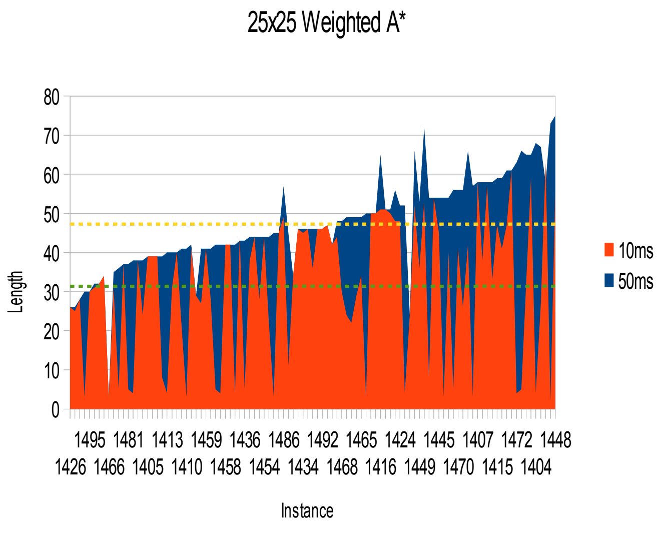


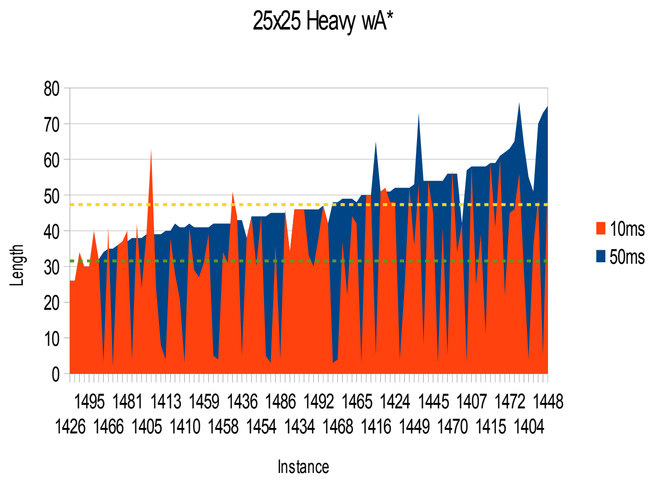


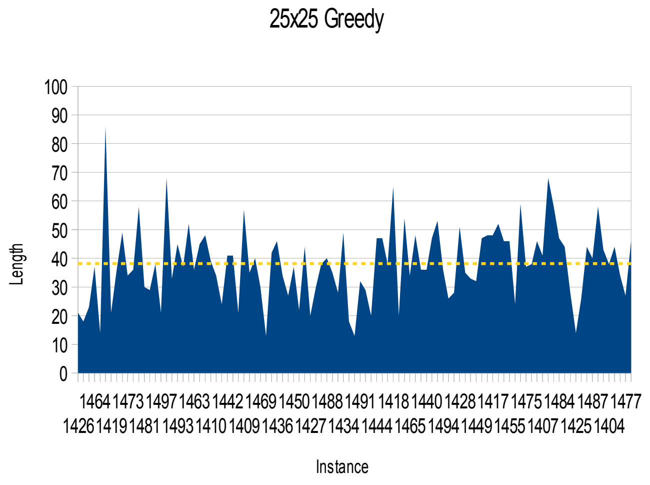


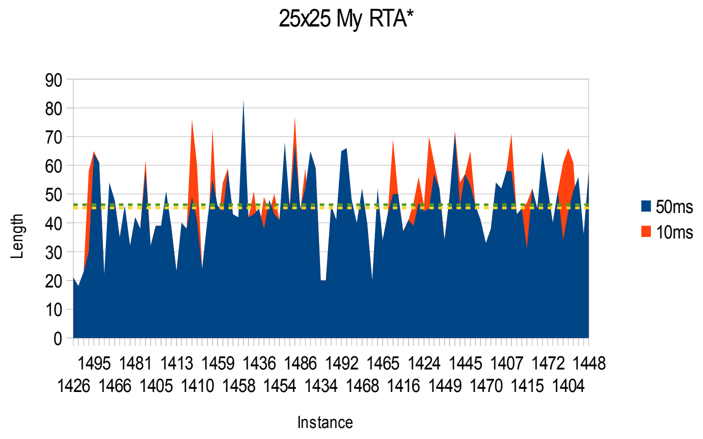
The instances in the graphs above are sorted in increasing order of difficulty for A*. Also there is only one set of data for each one because the result was identical for both 50ms and 10 ms delay speeds. The dashed yellow line in each graph represents the mean value of that agent across all instances. You can clearly notice by comparing these graphs that the algorithms that use A*(all but greedy) follow the same overall trend. However, weighted A* and My RTA* which are less focused on optimality fluctuate away from A*'s results and achieve both better and worse scores in some cases.



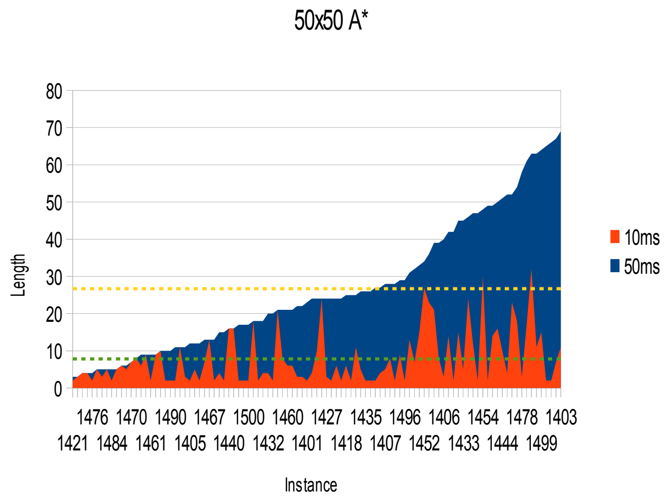


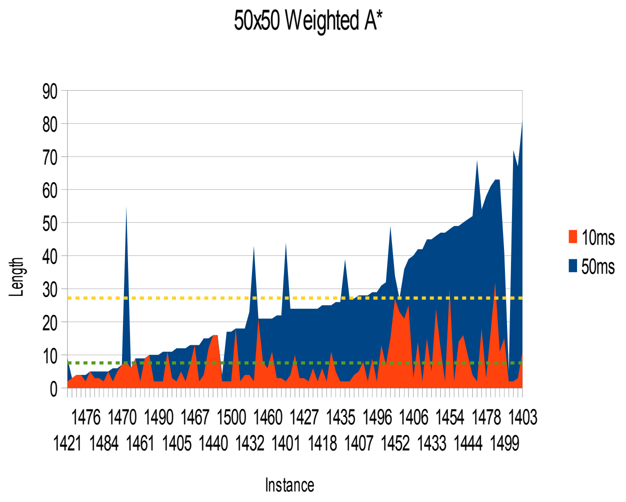


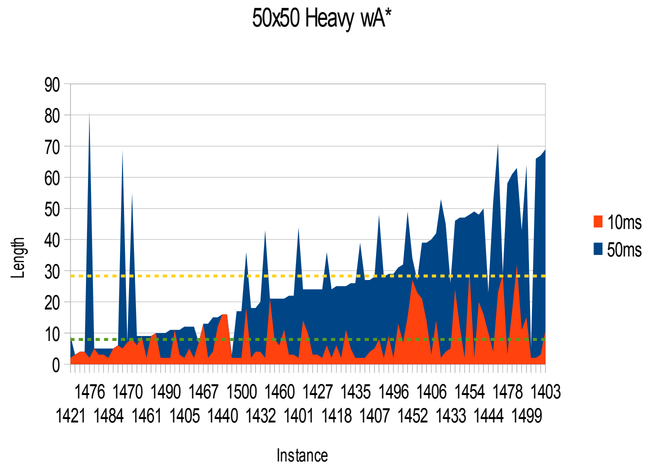


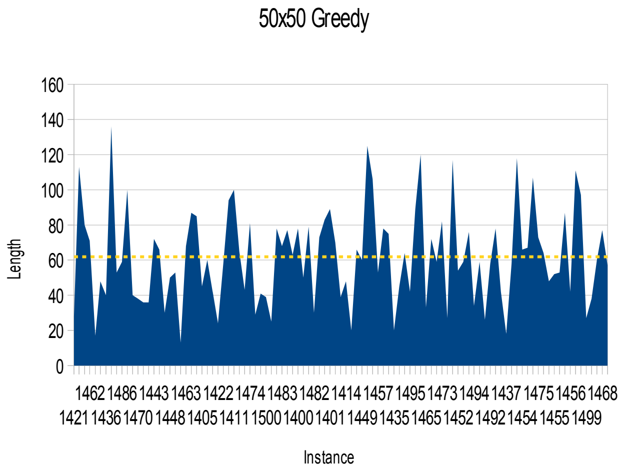


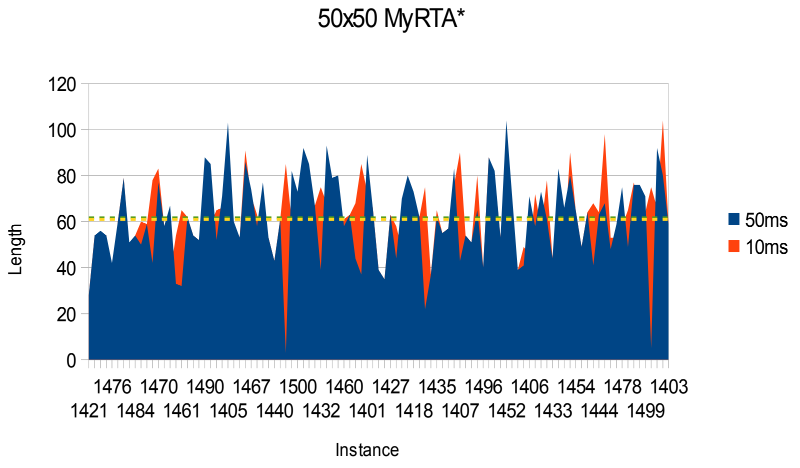
This set of graphs is sorted by increasing order of difficulty for A* at the slower 50ms delay speed. Greedy only has one set of data because both speeds gave the same results. The yellow line is again the mean value for the 50ms delay speed and the green line is the mean value for the 10ms delay speed in each graph. In these graphs you can clearly see A* and weighted A* beginning to breakdown based on decreasing the allotted time for calculations. Yet My RTA* seems unaffected and is actually performing better at the faster speed.











These graphs show the algorithms performing under the largest sized play area and under both speeds. At this point it is clear that A* cannot hold its own in a true time critical dynamic environment and breaks down quickly at the increase in speed, as well as not comparing to Greedy or My RTA*. Greedy managed the highest scores in this set, although still had a high variance of scores from 20 to 135. My RTA* again remained consistent at both speeds, still performing better at the faster speed, and had a smaller overall variance of scores from 40 to 90.

The actual mean values across all of the runs are as follows:

Size	Speed(ms)	A*	My RTA*	Greedy	wA*	h-wA*
5x5	50	13.69	12.7	12.78	13.81	13.82
	10	13.69	12.7	12.78	13.81	13.82
25x25	50	47.56	45.17	38.13	47.29	47.35
	10	31.7	46.33	38.13	31.36	31.55
50x50	50	26.71	61.07	61.87	27.24	28.34
	10	7.82	61.63	61.87	7.62	7.92

As you can see above at any speed in a 5x5 space A* and weighted A* ate the most apples and achieved the largest snake on average. The same result was true of the 25x25 space at the slower speed, however when the speed was increased A* and the weighted A*'s began to breakdown, leaving My RTA* algorithm to lead. At the 50x50 space A* and weighted A* continued to breakdown to the point of no longer being competitors, whereas greedy and My RTA* continued to be consistent with high scores on both speeds. This shows how real time domains require some type of early commitment especially when there is no guarantee on how long computation to a solution might take. Though that commitment may backfire, at least it allows for My RTA* and Greedy algorithms to fight as best they can against crashing. In some cases, like this one, survivability means more than finding the next way to score.

It is interesting how well greedy performed compared to My RTA*. Greedy itself may have been implementing a different high performance strategy to the game by restricting lookahead. It appeared that through the trials since greedy's movement choices were based on manhattan distance on a grid that it moved in a very box-like and rectangular fashion. Also when it found itself in a closed loop it actually filled the loop completely and usually resulted in the tail moving far enough to reopen the loop. Thus by taking a less than optimal path it improved the play of the algorithm. Yet without any look ahead at all it still made many poor decisions.

Perhaps for an extension to this project, it would be interesting to try to incorporate the rectangular movements and non-optimal path choices that greedy made into a smarter algorithm.

Another extension would be to make the game more efficient and try even larger game spaces to see if the greedy algorithm can actually overtake My RTA* which it might just be capable of. There is also the possibility of D* being the answer since its complexity was a bit much for a project of this length it would be a good addition(Likhachev et al., 2008). Finally, the only other attempt at improving the controller would be to try another heuristic. Even though none would compute faster than manhattan distance, My RTA*'s ability to commit to moves before a solution is found could still highly benefit from a more knowledgeable heuristic.

If future students were going to take on a project such as this one I would suggest to make sure all output can readily be transferred to a format where it can be analyzed. For instance, moving snake lengths or time played values into an excel spreadsheet, a comma or semicolon delimited output could really be useful to speed things up. Also make sure to account for how long it takes the program to run when doing data collection, so as to avoid trying to process numerous executions each spanning from 5 seconds to 5 minutes. Those small times add up fast, and as said in near every computer science class at UNH and probably every class is Start Early and don't be stuck throwing it all together during finals week.

Conclusion

In a real-time environment where the domain is dynamic and decisions are time critical the need for optimal solutions is less desirable. The true need in a real-time environment is a decision, and the more informed the decision the better. As shown by the results of my evaluation algorithms that produce optimal or suboptimal solutions, such as A* and weighted A*, perform well when there is enough computation time for a fully informed decision but break down quickly as that time requirement goes unmet. The algorithms that perform well, or at least consistently, in most any real-time variation of a problem are ones that can commit to a decision even before the solution has been realized. Greedy showed that committing to a decision can help tremendously in hard or long computation environments, though the commitments were not guaranteed to yield high scores in every scenario. My RTA*, using an anytime strategy, made the most informed decision it could with any amount of time it was given and produced consistent and non-sporadic scores in the Snake Game on any size area and over a variety of speeds. In the domain of the Snake Game My RTA* is the most competitive out of all of the algorithms I implemented, coming closely behind A* even when it was not the best. However, more work and improvements are needed to make My RTA* achieve higher scores and to be competitive with the highest ranked human players of the Snake Game.

References

Koenig, Sven and Xiaoxun Sun. “Comparing Real-Time and Incremental Heuristic Search for Real-Time Situated Agents”. *Autonomous Agents and Multi-Agent Systems.* Volume 18 Issue 3, 2009.

Korf, Richard E. “Real-Time Heuristic Search”. *Artificial Intelligence*. 1990:190-211

Likhachev, Maxim, et al. “Anytime Search in Dynamic Graphs”. *Artificial Intelligence*. Volume 172 Issue 14, 2008.

Russell, Stuart and Peter Norvig. Artificial Intelligence: A Modern Approach. New Jersey: Pearson Education, 2010.