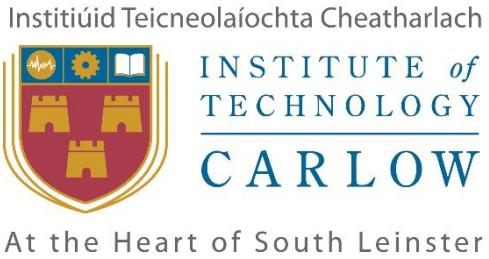


Computer Games Development Project Report Year IV

[Timo Bos]
[C00243739]
[Date of Submission]

[Declaration form to be attached]

Contents

Acknowledgements

I would like to thank the following people who assisted in completing this project including;

John Doe of ACME who kindly agreed to …

I would also like to thank ICME for use of ….

Project Abstract

Replace this text with an appropriate Project Abstract.

This section should introduce the problem domain and clearly identify, justify and explain the solution(s) chosen. Care should be taken to ensure that the summary clearly demonstrates the writer's expert understanding of the problem domain.

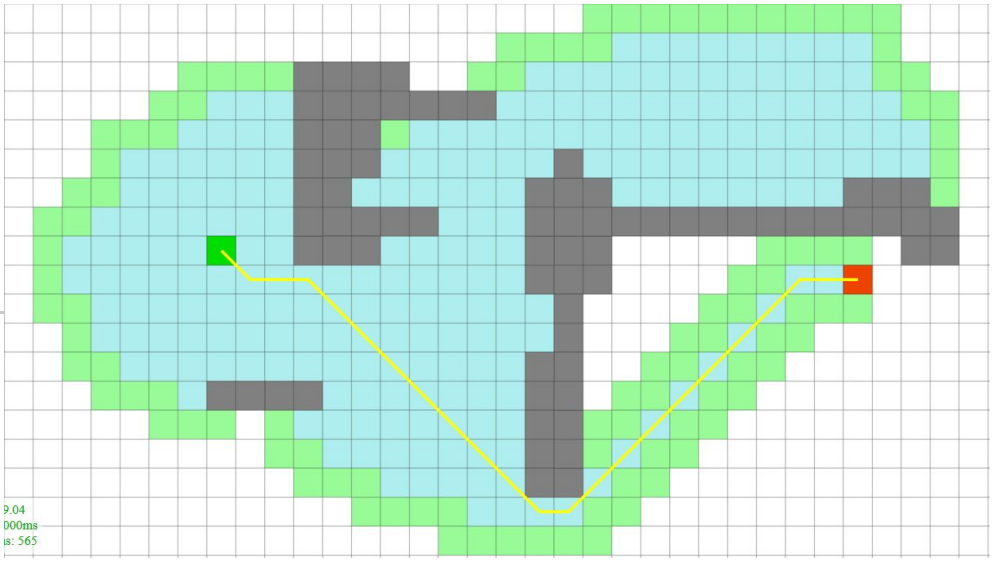
Project Introduction and Research Question

This research project was chosen because, i'm interested in AI. I knew about A* and that it was the "best" algorithm to use in game development, but A* didn't look realistic on itself, so without optimization. So this got me thinking is there another algorithm you can use which looks more realistic and is the same or "better" than A*. That's when i found Theta*. Theta* should be as good as A* but will look more realistic. That's what i'm trying to find out in this research project. Thus my question is "Is Theta* better than A*?". To calculate what better is, I will check different statistics. For example, runs the algorithm faster, the length of the path and how many nodes was visited.

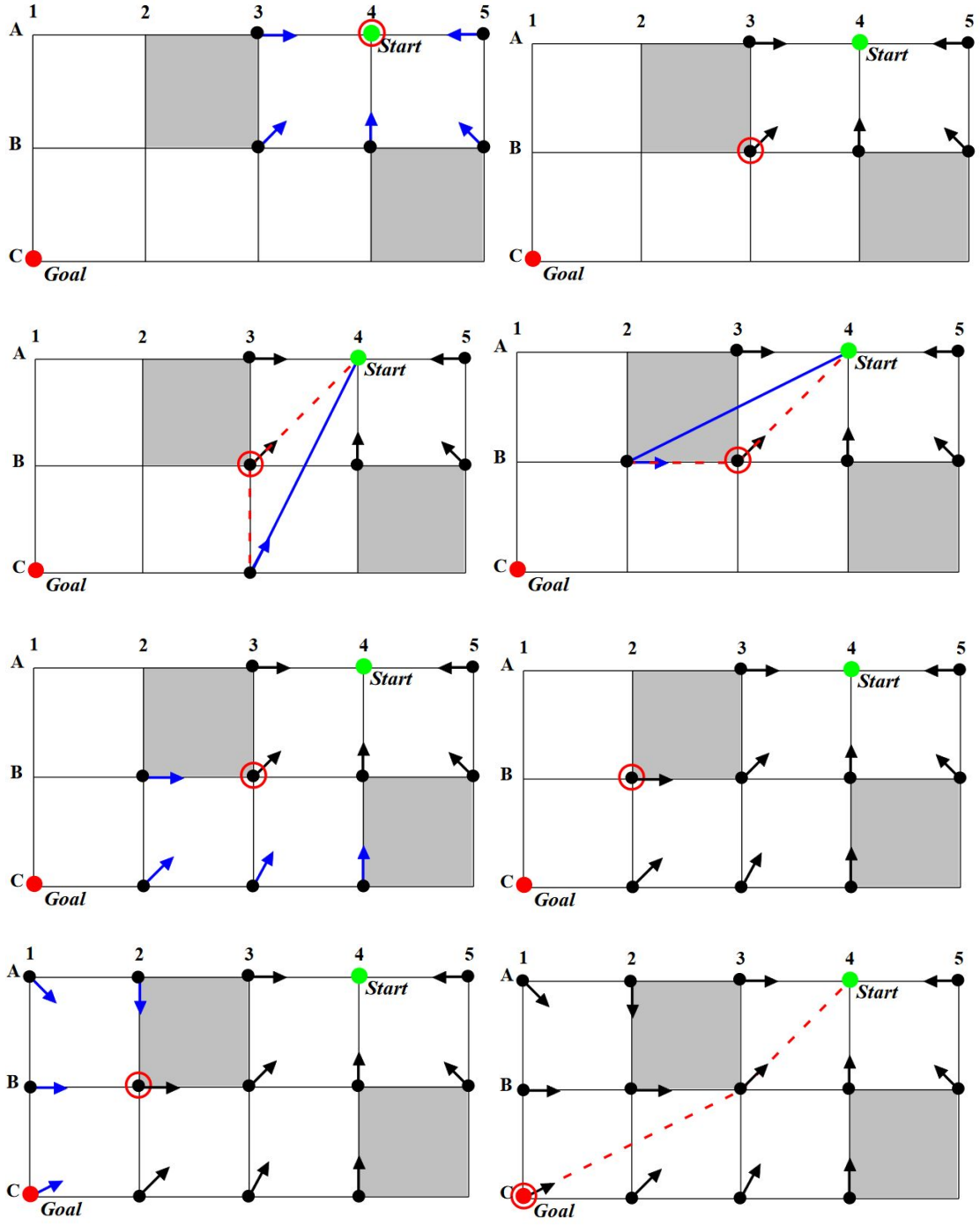
Literature Review

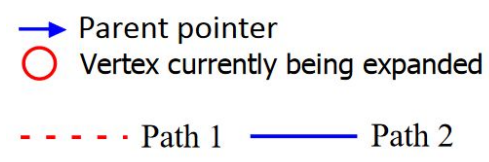
In grid based games, A* is a well-known 8-directional pathfinding algorithm [1]. Theta * is variation of A* that performs line of sight checks to find paths that don't strictly follow a grid [2]. Grid is that the playing field is divided into small tiles of a set size.

A* is a pathfinding algorithm which uses heuristics to find the shortest path in a short amount of time. A* will check all eight neighbours, and checks which one is closest to the goal. These neighbours will also do this check, and so on, until one of the neighbours is the goal. Then it will backtrack the path, because A* uses an extra evaluation, it won't check everything on the grid. The evaluation is an extra calculation that checks if that tile should be checked, shown as f(n) = h(n) + g(n). f(n) is the total cost of that path, lower the cost shorter the path is. g(n) is the cost from the goal to the current tile. h(n) is the heuristic, it's an estimated value of the cost from the current tileto the goal. The most common one for 8-directional pathfinding is Diagonal Distance[4]. If the algorithm thinks that one of these tiles won't be in the shortest path, it won't check that tile's neighbours. This will generate a short path, but this will look unrealistic, because the algorithm only goes in 8 directions; left, right, up, down and diagonally. What usually is done is optimize that path, to make it look more realistic.



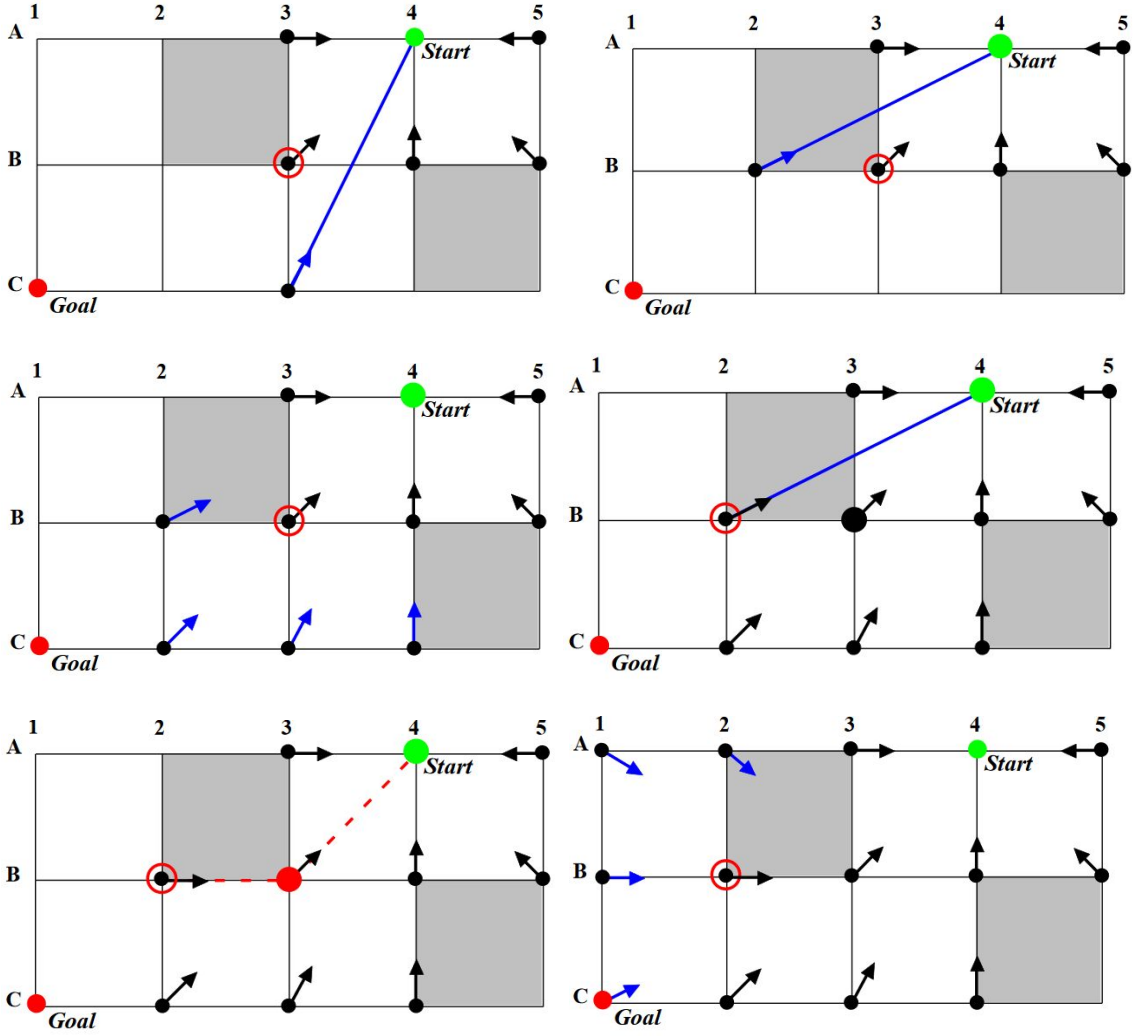
Theta* is almost the same as A*, in that way that it looks for the shortest path. Theta* makes two paths. It checks it's neighbours the same as A* and create a path between the neighbours, that's the first path. After that it checks if you can make a straight line between those neighbors, that's path 2.





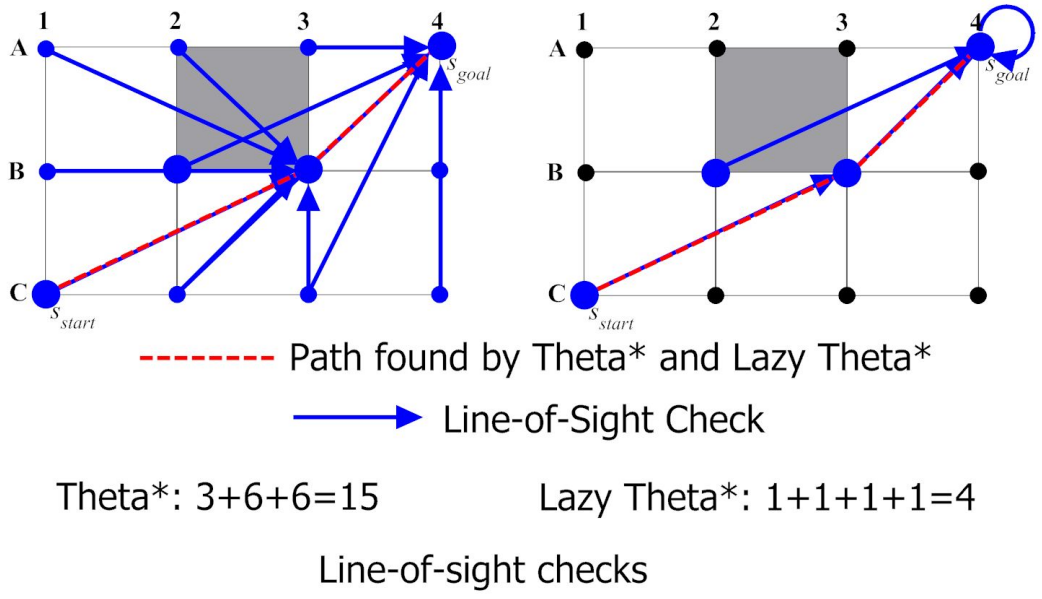
[6]

The figures above explain how Theta* finds a path. First the neighbours of the start node(A4) are checked which is closest to the goal(C1) thus it chooses B3. Next the neighbours are checked which are closest, so it makes two path for each neighbour, the dotted line is the first and the straight line is the second, it also does the line-of-sight checks for those neighbours with the start. It will choose B2, and now the neighbours of B2 are checked and it will find the goal, so it will choose the goal. This will get a short and realistic looking path. That's because Theta* uses line of sight so, it will go in any direction instead of just 8 [1]. So, Theta* looks more realistic, but it also needs to do more calculation for that. There's a variant of Theta* that fixes that problem called, Lazy Theta*. Lazy Theta* is almost the same as regular Theta* but with less line-of-sight checks. Lazy Theta* will only do a line-of-sight checks when it's absolutely necessary.



The figures above show the steps of Lazy Theta*, the first two steps aren't included because they're the same as Theta*. As you can see there is already what looks like a tiny difference, but it is actually pretty big. There is no dotted red line in the first two figures. That's because Lazy Theta* doesn't check it's neighbours the same as A* and Theta*, but it will automatically assumes that the neighbours of the expanded node has a line-of-sight the parent of the expanded nod. In this case Lazy Theta* assumes that the neighbours of B3, B2 and C3,

has a line-of-sight with the start node. And with that information it will pick the best node, which you see in the next step. B2 is picked, and now it will do a line-of-sight check between B2 and the start node. It will find out that it's not possible, so it will update B2 so it's parent is set back to B3. And this will continue until it finds the path.

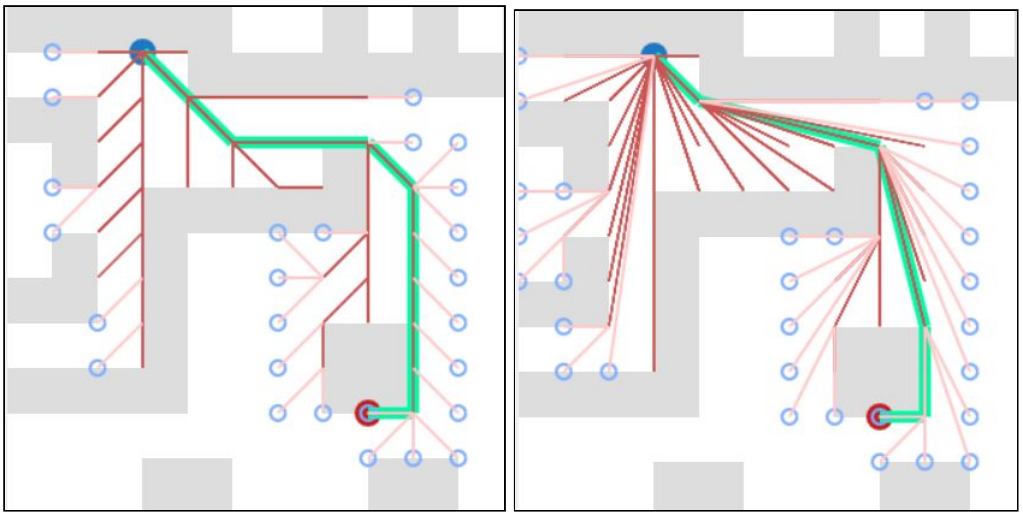


Line-of-sight checks

[2]

This figure shows how many line-of-sight checks each algorithm has done. The start and the goal are the wrong way in this figure. A4 needs to be the start and C1 the goal.

Lazy Theta* is practically the same as normal Theta*, but with less calculations. It turns out that Theta* performs more line-of-sight checks than it must. If Theta* performs a line-of-sight check between two tiles, and it's not further used, then it is wasted computation. Given that that is the case, Theta* might be a little too eager to perform line-sight checks. Because it performs a line-of-sight check for each visible neighbour even though many of those tiles may never be used [2].



(Visualization of A*and Theta*, http://idm-lab.org/project-o.html , accessed 10/10/2018)

Here is a comparison between A*(left) and Theta*(right). Blue dot is the start and red dot is the goal. The green line is the path the algorithm uses. The red line is a possible path and thus expanded. The orange line is checked but not used because it's to far and leaves a blue circle.

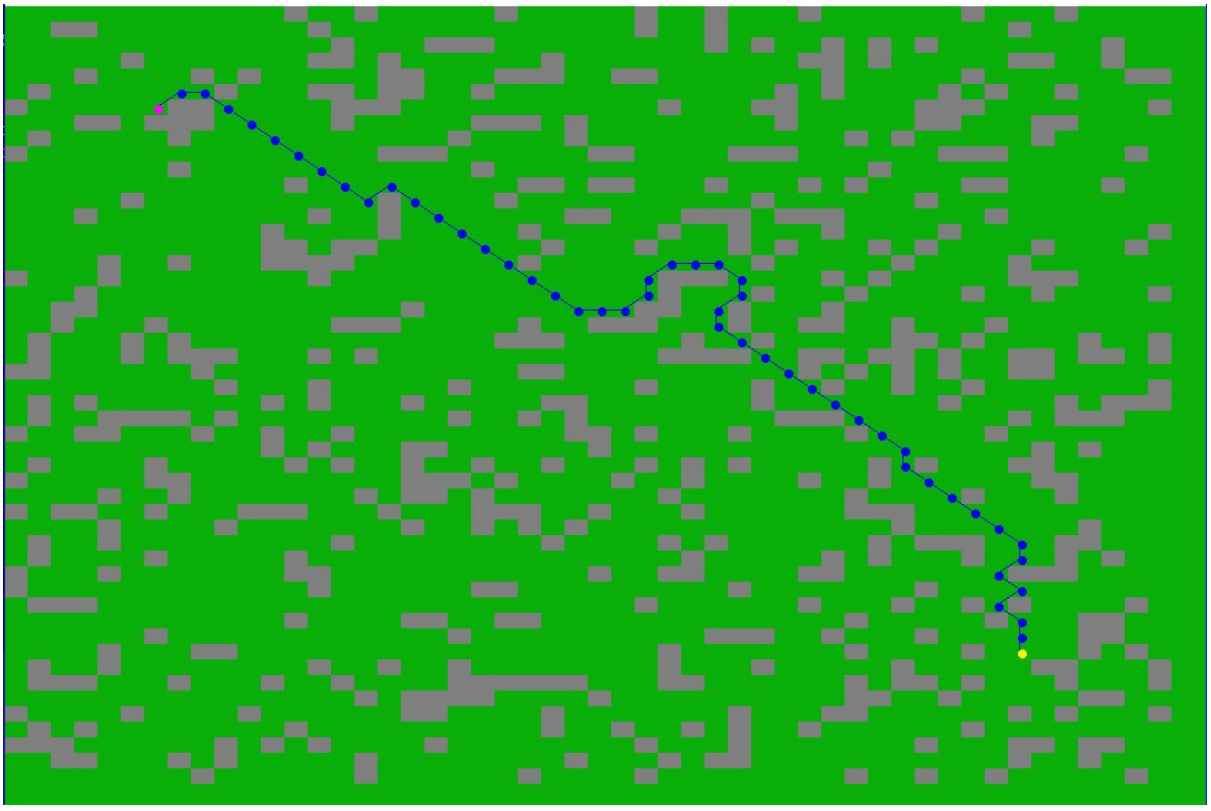
A* did 42 checks on different tiles, and the length of the path is 13.24 tiles, but as you can see the path it took doesn't look realistic. Theta* on the other end had 47 checks, but the length of the path is 12.66 and it looks realistic. So A* has less checks thus it needs less computer power, but Theta* gets a shorter and more realistic path.

Study

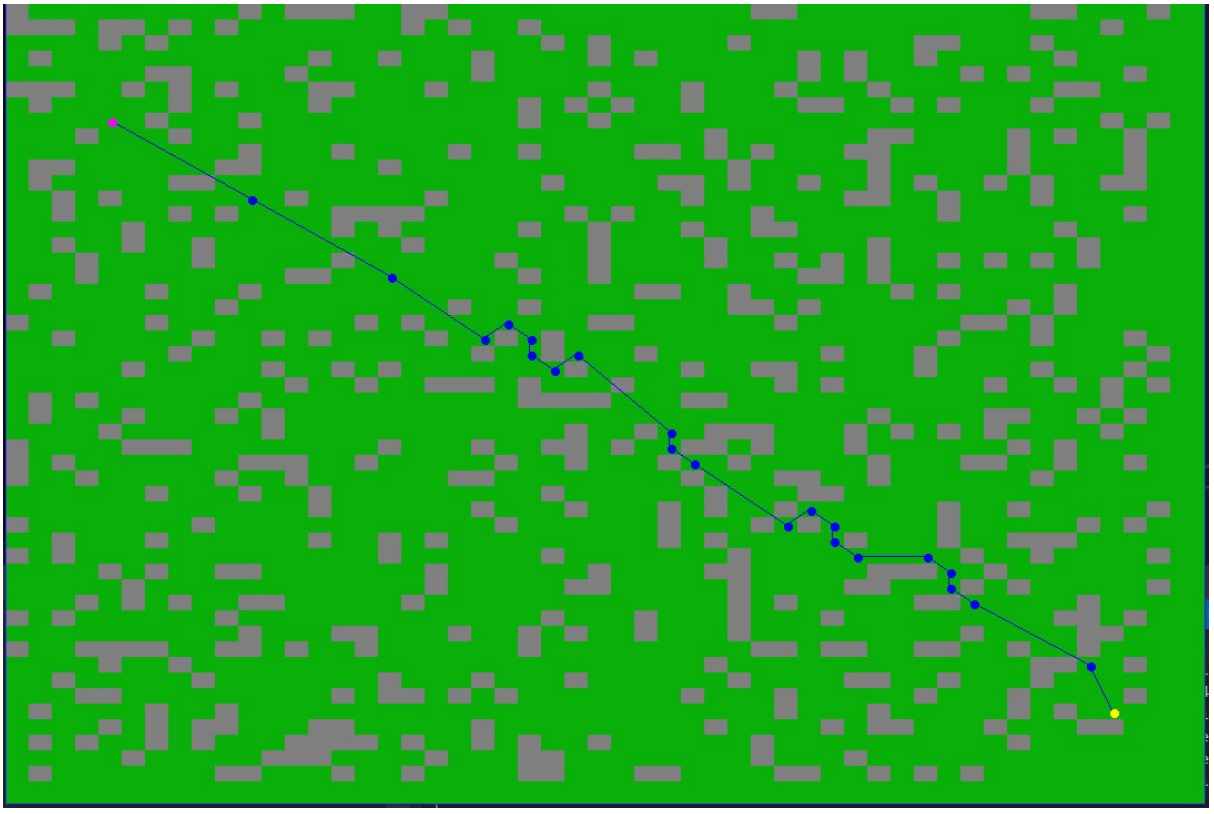
To compare these algorithms several test will be executed. That means that a playing field will be made, and then the algorithms are trying to find the shortest path. This will be done several times. Data will be recorded, the data is as follows: Whether a path was successfully found. Number of nodes visited and examined, this includes repeat visited. The time to walk the path in seconds and game loops. Number of nodes in the path and as last the length of the path in game world units. This test is based on an already made report[5].

Project Description

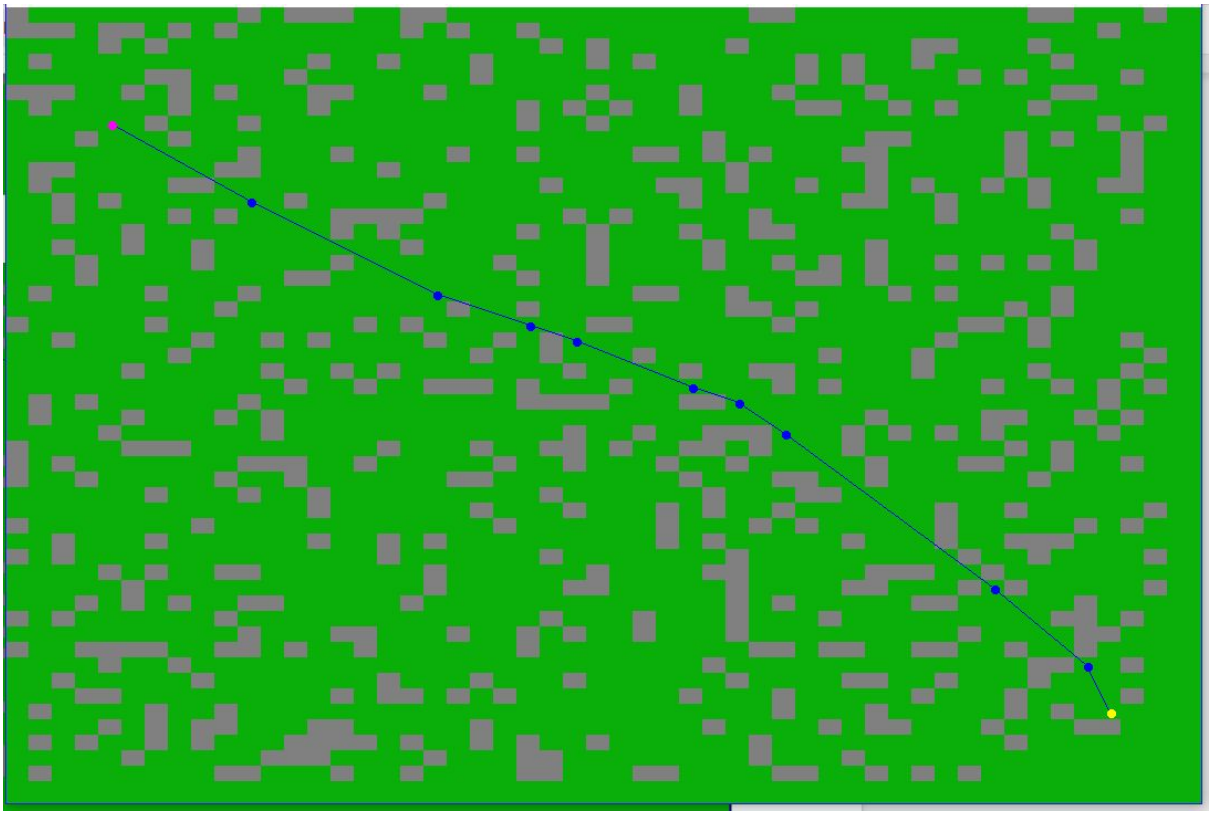
I made a visualization of the three algorithms above in c++ SFML. If you press a button it will find a path and draw that path. 'X' for A*, 'Z' For Theta* and 'C' for Lazy Theta*. And Data is saved afterwards to make a comparison.



A*



Theta*



Lazy Theta*

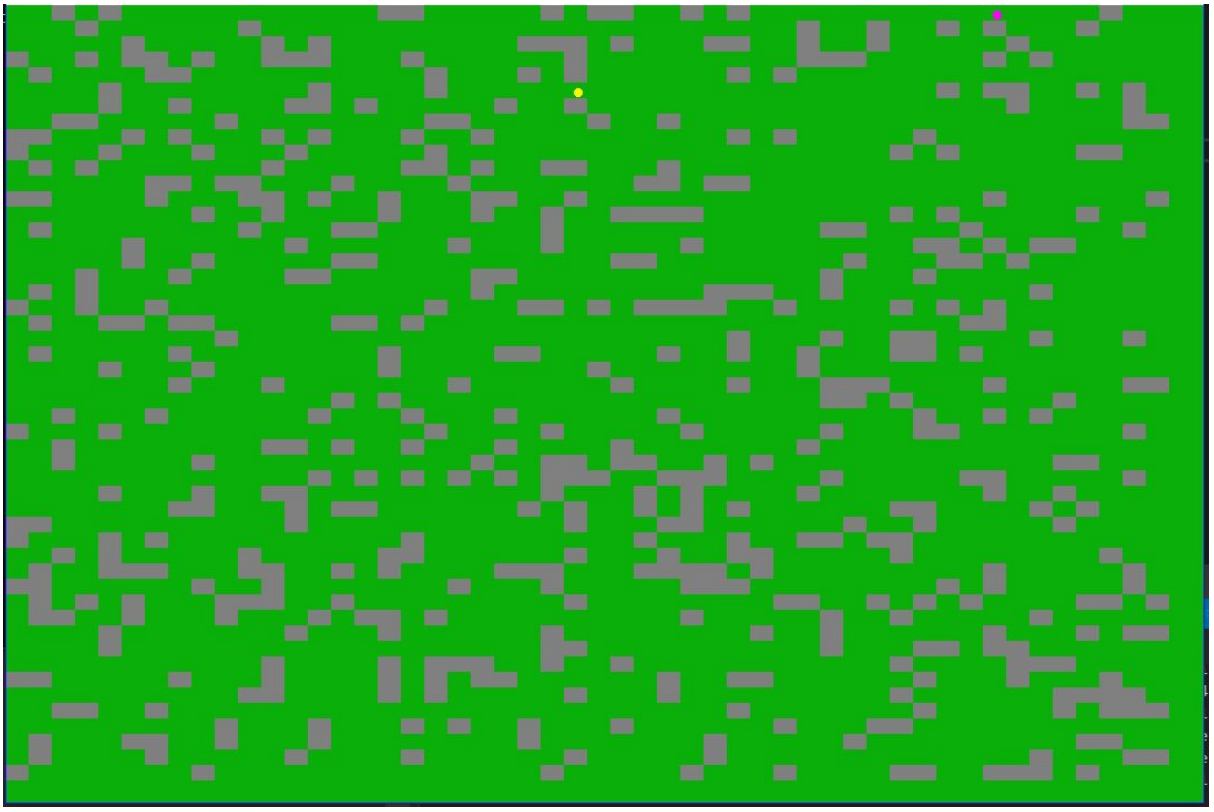
Results and Discussion

The Analysis uses data gathered from 20 test runs in two different environments using each of the three algorithms. that's in total 120 test. The data gathered from each test include:

1. The time it took to complete the algorithm(in milliseconds).
2. The length of the path in game world units.
3. The amount of nodes expanded
4. The amount of neighbours visited
5. The amount of nodes in the path
6. The amount of Line-Of-Sight checks(only relevant for Theta* and Lazy Theta*).

The tables will show the average of each of these data.

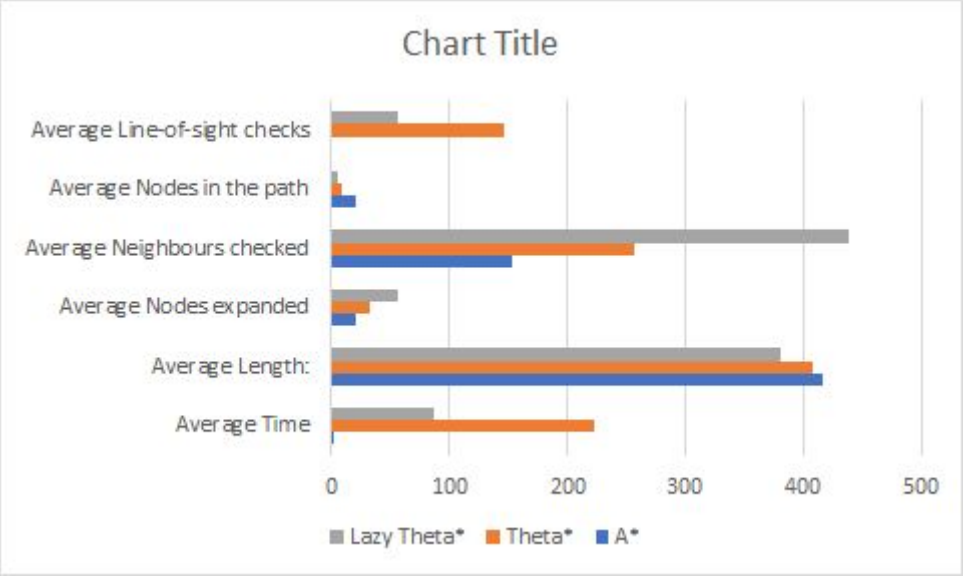
Environment 1:



Environment 1 is completely randomly generated.

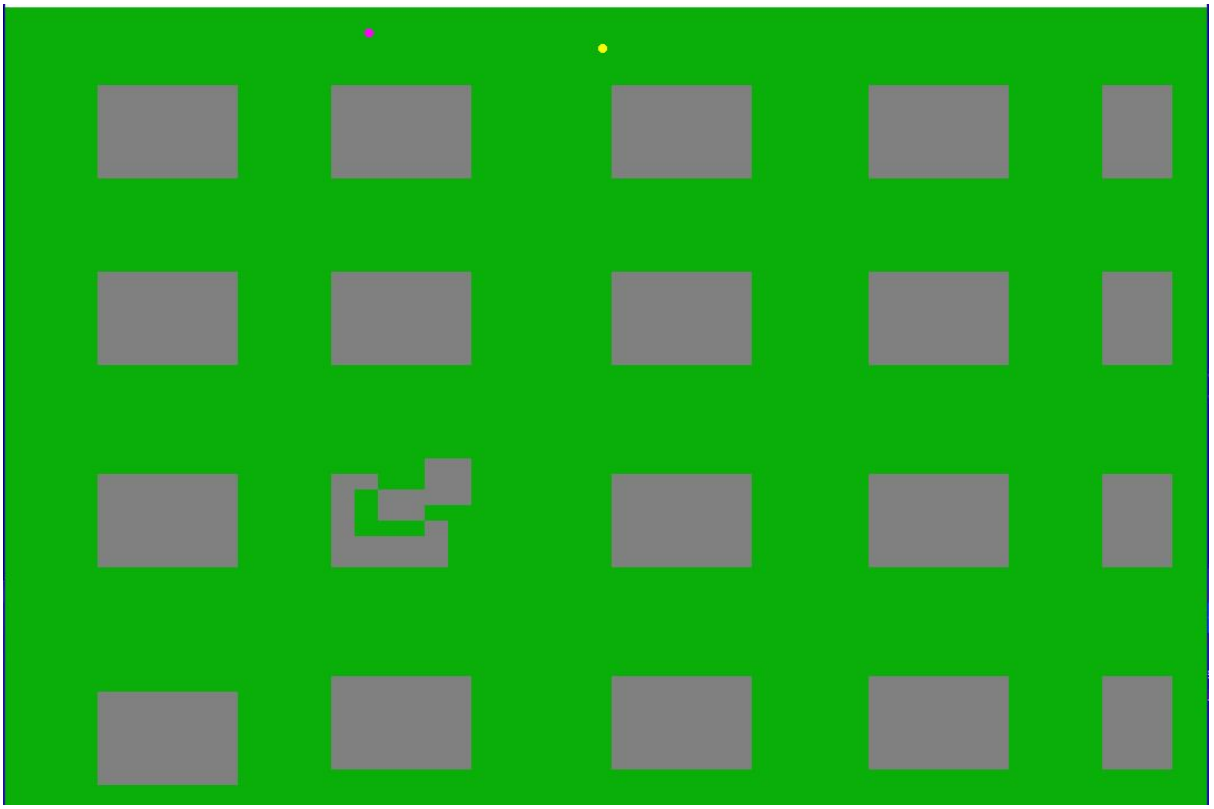
Algorithm	**A***	**Theta***	**Lazy Theta***
Average Time(in milliseconds)	0.44695	222.9919	86.6214
Average Length:	416.9373	407.3272	380.81161
Average Nodes expanded	20.45	33.4	56.55
Average Neighbours checked	153.95	256.05	438.6
Average Nodes in the path	20.1	8.75	5.2
Average Line-of-sight checks	0	146.65	56.55

All Algorithms found a path during these test.



The time is very different in all the algorithm. A* is the fastest by a long shot, that's because A* doesn't do any line-of-sight checks and A* needs to check less nodes and neighbours. The difference between Theta* and Lazy Theta* is still huge, that's where you see that the line-of-sight checks take awhile. Lazy Theta* does ⅓ of the line-of-sight checks as Theta* and the time of Lazy Theta* is also ⅓ of Theta*. However Lazy Theta* checks almost double the nodes and neighbours as Theta*, so that doesn't affect the time that much. There is also a pretty big difference between the length of the algorithms. Lazy Theta* generates a shorter path with least amount of nodes, and A* the longest path with the most nodes. It was to expected, A* is the fastest but generates a bigger path, and Lazy Theta* is slower but generates a shorter path.

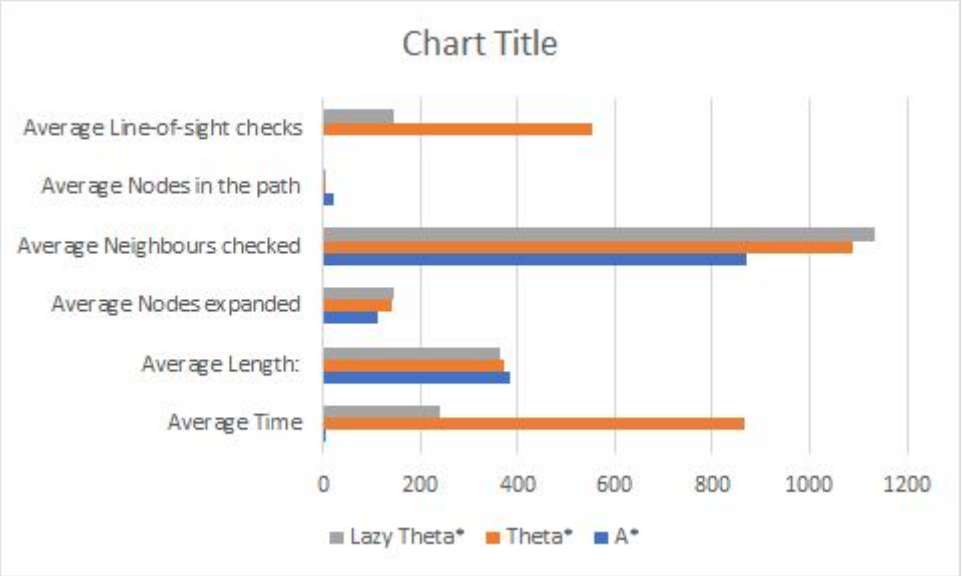
Environment 2:



For environment 2 I wanted to go for a self made map with the obstacles in a grid formation. I messed up one of the obstacles.

Algorithm	**A***	**Theta***	**Lazy Theta***
Average Time(in milliseconds)	0.738	867.2704	241.6396
Average Length:	385.9331	370.514	365.6984
Average Nodes expanded	114.1	141.2	147.4
Average Neighbours checked	871.75	1086.45	1135.45
Average Nodes in the path	21.6	3.95	3.05
Average Line-of-sight checks	0	553.25	147.4

All algorithms found a path.



There is a lot different between the two environments. a lot more nodes are checked and thus it took longer. 3 times longer for Theta* and Lazy Theta* and two times longer for A*. The path is shorter for all algorithms, and the difference in lengths between the algorithms are shorter. And for Theta* and Lazy Theta* there are less nodes in the path, but for A* there are more nodes in the path.

Project Review and Conclusions

With this data you can calculate which algorithm is the most efficient. so the algorithm which finds the shortest path with the least amount of nodes visited, is the most efficient. A basic efficiency score can be calculated. S = k/(V*L), Where S is the efficiency score, V is the average nodes visited, L is the average length of the paths, and k is a constant of 100.000 for a better understanding. A larger efficiency score indicates a more efficient algorithm. [5]

	Environment 1	Environment 2
A*	11.728	2.271
Theta*	7.350	1.912
Lazy Theta*	4.644	1.855

Of the three algorithm Lazy Theta* is the least efficient, but as shown in the previous tables it ran faster than Theta* and generates a more realistic path than A*. A* is the most efficient, that's was to be expected, it ran the fastest and checks the least amount of nodes, but the path looks unrealistic. it's also interesting to see that all three algorithms were more efficient on the first environment which was completely randomly generated.

I'm really satisfied with the outcome of this project. At the beginning of this project I was still unexperient to c++ and it was my first time working with SFML. I got a better understanding of c++ now because of this project. I also handled it different than is usually do, usually I look up tutorials of other people doing it, but this time i tried to do it all myself with the help of pseudocode. I already had a previous understanding of A* that was helpful. Theta* and Lazy Theat* were completely new to me and those algorithms are relative new so there wasn't a lot of research on it, luckily there was one paper that explained everything so I learned alot from that paper. I find AI pathfinding interesting so it's nice to know new pathfinding algorithms.

The things I want to improve if had more times are: My code of the algorithms could use some improvement to make it more efficient. Instead of using regular A*, using A* with post process smoothening. There is another Theta* variant which looks interesting named Strict Theta*, it would be nice to also implement that. And as last to run more test in different environments, it took me quite awhile to gather all that data and put in a graph.

References

Use the Harvard style of referencing to cite your information sources. See this document entitled 'Credit where Credit is due' for further information.

Web-site

[1] Nash, Alex. (2010, September 8). *Theta*: Any-Angle Path Planning for Smoother Trajectories in Continuous Environments*. [Online] (URL http://aigamedev.com/open/tutorial/theta-star-any-angle-paths/). (Accessed 7 October 2018)

[2] Nash, Alex. (2013, July 16). *Lazy Theta*: Faster Any-Angle Path Planning*. [Online] (URL http://aigamedev.com/open/tutorial/lazy-theta-star/). (Accessed 7 October 2018)

[3] Mendonca, Pramod. (2016, February 1). *C – THETA* : PATH PLANNING ON GRIDS.* [Online] (URL https://scholar.uwindsor.ca/cgi/viewcontent.cgi?article=6653&context=etd). (Accessed 7 October 2018)

[4] Patel, Amit. (2018, September 25). *Heuristics.* [Online] (http://theory.stanford.edu/~amitp/GameProgramming/Heuristics.html). (Accessed 15 October 2018)

[5] Krishnaswamy, Nikhil. (2009, June 1). *Comparison of efficiency in pathfinding algorithms in game development.* [Online] (https://via.library.depaul.edu/cgi/viewcontent.cgi?referer=https://www.google.com/&httpsredir=1&article=1010&context=tr). (Accessed 21 October 2018).

[6] Nash, Alex and Koenig, Sven. (2017, October 7). *Any-Angle Search Case Study: Theta**. [Online](http://idm-lab.org/slides/anyangle-tutorial.pdf). (Accessed 4 November 2018).

Appendices

Replace this text with Appendices.

This might include ethics application and other relevant material e.g. copy of any questionnaires used.