

Institiúid Teicneolaíochta Cheatharlach



INSTITUTE *of*  
TECHNOLOGY  
CARLOW

At the Heart of South Leinster

# Computer Games Development CW208

## Project Report

### Year IV

Przemysław Tomczyk

C00218004

3rd of May 2020

Institiúid Teicneolaíochta Cheatharlach



INSTITUTE of  
TECHNOLOGY  
CARLOW

At the Heart of South Leinster

## Faculty of Science

Open-Book and Remote Assessment Cover Page

**Student Name:** Przemyslaw Tomczyk

**Student Number:** C00218004

**Lecturer Name:** Philip Bourke

**Module:** Final Year Project

**Stage/Year:** 4th

**Date:** 03/05/20

### Declaration

**This examination/assessment will be submitted using Turnitin as the online submission tool. By submitting my examination/assessment to Turnitin, I am declaring that this examination/assessment is my own work. I understand that I may be required to orally defend any of my answers, to the lecturer, at a given time after the examination/assessment has been completed, as outlined in the student regulations.**

## **Table of Contents**

<b>Acknowledgements</b>	<b>4</b>
<b>Project Abstract</b>	<b>4</b>
<b>Project Introduction and Research Question</b>	<b>5</b>
<b>Background</b>	<b>6</b>
<b>Literature Review</b>	<b>6</b>
<b>Study</b>	<b>7</b>
<b>Project Description</b>	<b>8</b>
<b>Project Milestones</b>	<b>11</b>
<b>Results and Discussion</b>	<b>12</b>
<b>Project Review and Conclusions</b>	<b>13</b>
<b>References</b>	<b>15</b>

## **Acknowledgements**

I would like to thank Philip Bourke for supervising my Final Year Project. He has helped me quite significantly. He had valuable suggestions and feedback that made the final product much better. Phil was my lecturer in 3 out of 4 years while I was in college and he has always provided valuable feedback and knowledge that allowed me to get through these 4 years of college.

I would like to thank Ross Palmer for research suggestions that steered me towards this topic. He was a tremendous help when I was trying to figure out a project proposal. He is also an incredible source of knowledge which I was able to get a little of, in 2 years, that I had him as a lecturer.

I would also like to thank Zhang An for his Rectangle Expansion A\* paper and incredible pathfinding algorithm. I had an incredibly good and rough time figuring it out but nonetheless it was really fun and challenging to work on.

I would also like to express immeasurable gratitude to both of my parents, Mariusz and Hanna, for supporting me throughout my college time.

Quick thank you to my friends that have helped me out in figuring out bugs, caused by none other than myself. Otherwise, I would still be fixing them.

## **Project Abstract**

This project is about testing relatively new algorithm called Rectangle Expansion A\* (REA\*).

As the name of the algorithm indicates, it derives from A\*. Due to this, my idea was testing REA\* and comparing efficiency, speed and CPU cost of both algorithms with a large number of NPCs that are spread over several threads.

The idea is that this will show a much bigger improvement over regular A\* when multithreaded with a large number of NPCs.

Showing a result that is a small number indicating how long, in milliseconds, the algorithm ran for doesn't show the improvement over regular A\* as a modern PC will have no problem in handling several NPCs running on an optimised A\*.

My solution is to show how long it will take, with a large number of NPCs, which will put a lot of pressure on the PC.

## **Project Introduction and Research Question**

This project is aimed at games with huge maps, grids and search spaces. I chose this project as I have an interest in these sort of games and have experienced poorly optimized algorithms or simply just going with A\* due to its popularity and simplicity. This often resulted in NPC's poor pathfinding but also a delay when a large group was trying path find but also poor game performance even when the game wasn't a triple A title.

A\* has been the golden standard for a very long time, the moment there is any path finding to be done, most developers implement A\* but it's not necessarily the best choice for some games.

Picking the right tools for the job is very important. Not doing so can potentially cause harm to the game so it is important that you dive deeper rather than simply going with what is the easiest and fastest to implement. With this project I am to spread awareness of relatively new algorithms that are much better suited for these tasks instead. That being said, algorithms have their own use cases and new doesn't always mean better.

In this project I will attempt to implement REA\* from Zhang An's paper and compare it against A\* in 4 different search spaces. First layout will be 140 tiles, this layout was used in REA\*'s paper. I will use it to confirm both my A\* and REA\* are working similarly to Zhang An's. Second layout is going to be 10,000 tiles. This layout should start to show the differences between the two algorithms. Third layout is going to be the medium layout that will be 250,000 tiles. Lastly, I will try to push both algorithms in a search space of 1,000,000 tiles.

I will also conduct tests on the 10,000 tiles layout, with several NPCs running in a thread queue, to show a more realistic comparison to what it would run like in a game.

## Background

I have researched 2 available methods of multithreading. There are features in C++11 (and higher) that could be used. There is also the Boost library with a lot of useful features that I may require throughout this project.

C++11 introduced multithreading to the language which allowed for quite a lot of functionality that Boost provides. This allows for multithreading features without needing to implement this huge library.

C++20 has a lot of features planned that will further help with introducing multithreading features that are native to C++. These features will increase native functionality of multithreading in C++ without needing libraries.

Boost is a humongous library with a variety of features. This library used to be a must have when working with multithreading. When a language has native features that are more robust and don't require an external library. I decided to go with Boost as it is more commonly used while providing more features.

## Literature Review

### Rectangle Expansion A\*

<sup>1</sup>*“Search speed, quality of resulting paths and the cost of pre-processing are the principle evaluation metrics of a pathfinding algorithm”*

This paper talks about different algorithms, how they apply post-processing steps to achieve better than grid-optimal pathing while avoiding constraining paths to grid edges.

It introduces “a new optimal algorithm” called Rectangle Expansion A\* (REA\*). It gives a detailed description of the algorithm for 8-connected grid maps, optimality and analysis of REA\*, a possible trade-off to further enhance it and some results and comparisons with different A\* variants.

### A\* Algorithm for Multicore Graphics Processors

Since this algorithm is relatively new, there hasn't been any work done with it but there has been work done on running algorithms in parallel on the GPU.

<sup>2</sup>*“We have implemented A\* in CUDA, NVIDIA's programming platform for graphics processors. Our graphics processor A\* implementation is faster and more efficient than a similar CPU version of the same algorithm.”*

In this paper, the researcher talks about how GPUs are “very powerful and highly parallel” as GPUs have hundreds of processor cores and thousands of threads.

<sup>3</sup>This can be seen in real life as GPUs are preferred for Deep Learning. We can see the results of using GPUs for Deep Learning with NVidia's recent release of RTX Voice.

While some focus on “Performance evaluation of parallel multithreaded A\*” to solve the 15

puzzle problem as it “outperforms the sequential approach in terms of time complexity and speedup”.

### Block A\*: Database-Driven Search with Applications in Any-angle Path-Planning

Peter Yap talks about how <sup>4</sup>“a RTS game (like Starcraft) must handle the pathfinding requests of up to eight competing human players, each capable of controlling an army of up to 200 units: a worst case total of 1600 pathfinding calls which must be handled in real-time over a network.” Games Industry keeps pushing hardware to its limits every time a new Triple A title comes out. To put it in perspective, Starcraft 2 was released in 2010. This really calls for people to come up with newer and better algorithms and optimisation techniques. His algorithm, Block A\* delivered at the time. In his paper he showed that his algorithm “Block A\* is fastest, over 560 times faster than Theta\*”. Theta\* was one of the better algorithms at the time.

When REA\*'s paper was released, it was much faster than Block A\*. I'm really impressed that REA\* was able to beat an algorithm that was already 560 times faster than other ones.

### Study

In my research, I will take 2 path finding algorithms, REA\* and A\*, and compare how they perform on different search spaces, from small to large, several amounts of NPCs running on a work queue. NPCs will be placed on a Work Queue, to utilise all CPU threads, and measure how long it takes for each algorithm to find a path to the goal.

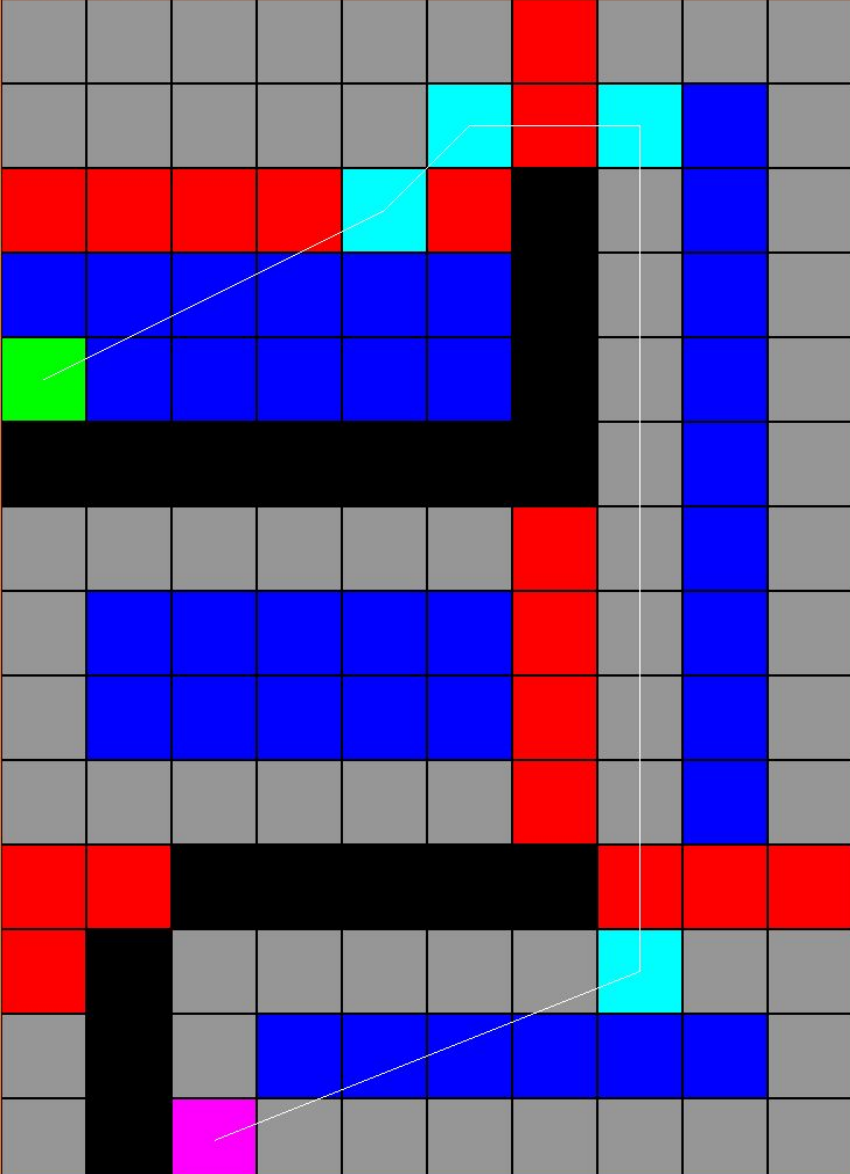
I will measure time taken and path length for each algorithm to determine whether or not the time spent implementing, debugging and using a relatively new algorithm is worth it, rather than using the old and trusted A\*.

I will take results from each algorithm, slot them into a table where they can be easily and quickly compared. In the future, this would allow us to generate a graph to show at what point it starts to pay off by using newer algorithms.

## Project Description

Implemented feature set:

1. Dynamic grid with obstacle placement and deleting
2. Quick grid size changing using F1 to F4 keys (smallest to largest)
3. Toggle for “Debug” mode
  - a. A\* auto steps through
  - b. REA\* step through when N pressed
4. Quick Switching between A\* and REA\*
5. Time and Distance displaying
6. Test Layout - 140 tiles total
  - a. Used to compare with REA\*’s paper
  - b. Managed to get shorter path than REA\*’s author



**Mouse controls:**

- Press LMB to place Goal
- Press RMB to place Start Tiles
- Press/hold MMB to place Obstacles

**Keyboard controls:**

- Press ENTER to start search  
\*\*START and GOAL required\*\*
- Press SPACE to toggle between placing obstacles and resetting tiles using MMB
- Press TAB to switch algorithms  
Using **REA\***
- Press F1 to load Test Layout
- Press F2 to load Sandbox Layout
- Press F5 to turn debug ON/OFF  
**DEBUG OFF**
- Press R reset the grid

**REA\* Stats**  
19 micro-seconds  
23.271513

**PLACE MODE**



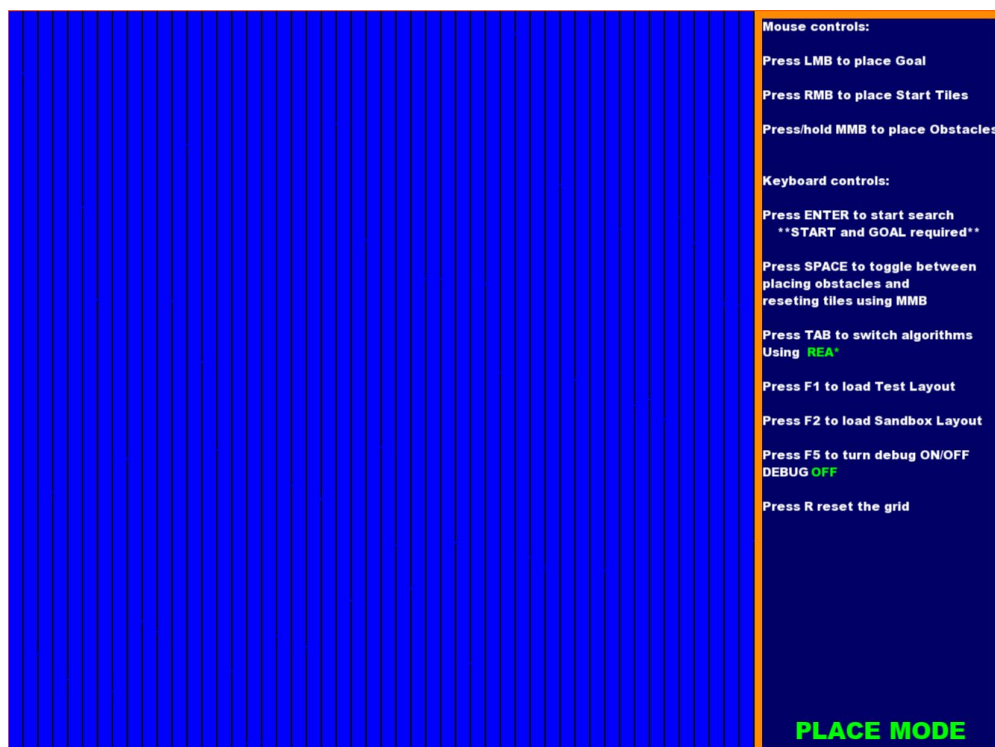
7. Small layout - 10,000 tiles total

- a. Segmented into squared areas
- b. Used to imitate a larger search space with multiple rooms



8. Medium layout - 250,000 tiles total

- a. Segmented into vertical rectangles



9. Large layout - 1,000,000 tiles total
  - a. Divide into 4 vertical rectangles



I started this project expecting a REA\* vs A\* comparison with several NPCs running on a threaded Work Queue. I sadly was not able to get multithreading implemented.

As I was working on it, I was rendering on a separate thread. Once I started to make bigger layouts and paths were much longer, it started to cause problems. Since I wasn't focusing on rendering speeds, I took it to focus on the algorithm work.

Multithreaded work queue. Due to me severely underestimating how low it will take me to figure out REA\*'s pseudo code, actual implementation, fixing bugs and problems and finally optimising it to a reasonable standard, I failed to implement multiple NPCs using a work queue. This isn't all of it though. There were some time management issues on my side. This included the ARGO Project in February, where as a team we fully focused purely on that. In the end it paid off, we won 2 great awards but at the expense of not being able to fully compare the algorithms as I set out to. As I kept pushing the Work Queue later and later, the deadline kept getting closer and closer. This ultimately led me with not enough time to implement it due to how I designed my Grid Manager etc.

In my opinion, this doesn't completely ruin the comparisons as the execution times can get quite lengthy with certain grid layouts. This would've led me to being unable to show off the project live but ended up trading off more accurate comparison.

This is far from taking away from the comparison. They are still providing some interesting results, including my way of implementing REA\*, giving even shorter paths than the author's.

## **Project Milestones**

I started off with a grid and A\* implementation. A rough implementation was done between October and November, with REA\*'s first implementation at the end of the Christmas Holidays.

At that point I was relatively happy with the project schedule and progress made up to that point.

It got worse from that point as we got overloaded with assignments and some unreasonable deadlines for some projects. There was time here and there where I could've chipped away at it but I was too burnt out. At that point Project ARGO came along and our team decided to commit 100% of February's time to it.

After ARGO we had some deadlines yet again. I was starting to get back into the swing of things when COVID-19 sent Europe into chaos. I got smacked off course due to having to move back home. This wouldn't have been an issue if I hadn't fallen behind earlier.

In April I finally got into a work schedule and started making tremendous progress. My REA\* implementation was no longer as problematic and buggy.

At this point I was extremely behind schedule and was 80% sure Work Queue will not make it into the final build, which ended up happening.

## Results and Discussion

	Path Length		Time Taken	
	REA*	A*	REA*	A*
<b>Test 10x14</b>	23.27	24.07	18 $\mu$ s	15 $\mu$ s
<b>Small 100x100</b>	990.50	998.61	770 $\mu$ s <1ms	930 $\mu$ s <1ms
<b>Medium 500x500</b>	7192.21	7319.17	9863 $\mu$ s 9.8ms	33856 $\mu$ s 33.8ms
<b>Large 1000x1000</b>	3700.91	3930.07	25581 $\mu$ s 25.5ms	172074 $\mu$ s 172ms

Results of running both algorithms with the same Start and End points.

Here we can see a table of results returned by both algorithms from the 4 different layouts and sizes.

Starting off with the Test layout. This layout was shown in REA\*'s paper so I decided to use it more as a comparison to my implementation than REA\* vs A\*.

This is a tiny layout and both algorithms fly through it without any problems. A\* found the goal marginally quicker, 3 microseconds faster, but this was at the expense of path length, which was 0.8 units longer than REA\*'s.

Once we move to small, larger in comparison to the previous layout, we can see that REA\* is now returning faster, 160 microseconds, than A\* while also giving us a shorter path by 8.11 units.

Medium layout is when we start to see a significant improvement over A\*. REA\* returns a path 126.96 units shorter while also being 7 microseconds short off of 24 milliseconds faster. This is the first result where we see both algorithms going over 1ms but REA\* does it while being more than 3 times faster.

This comes to no surprise that the largest layout of 1 million tiles is dominated by REA\*.

The distance returned by REA\* is nearly 230 units shorter than A\*'s.

The time taken by REA\* is much less than A\*. It is 146ms less than A\*.

This means that REA\* not only returned a shorter distance but could've done it 5 more times, very close to 6, before A\* even finished.\*

\*I realised after the fact, that A\* was doing some preprocessing. Fixing that only shaved off about a tenth of its time, meaning it was still 5 times faster.

## Project Review and Conclusions

Let's start with what went right.

1. REA\* is working well and performing even better.

As we can see from the results, my implementation was quite successful in performing better than A\*. It is by no means perfect, there are plenty of things that can be further improved upon. These issues/fixes are:

- (a) Preventing REA\* entering previously created rectangles. This can lead to some nasty issues where the algorithm's time drastically increases but the path still manages to be shorter.
- (b) Further implementation of optimisation techniques from the original author. This includes creating starting rectangles at Start and End point to see which one covers more area.
- (c) Another method the creator uses, albeit it trades off path optimality for time taken, is to identify and handle semi-closed areas differently. The path will still be shorter than A\*.
- (d) I feel like the algorithm has areas where it could easily utilise multithreading. One of the main functions in REA\*, called Expand, takes 3 borders of the newly expanded rectangle and sets them up with heuristics and parent pointers. This could be completed parallel as there would be very little to no synchronisation required to have thread safety.

2. Managed to improve original author's algorithm to an extent

As mentioned earlier, my implementation managed to return a distance slightly shorter than original creator's. I see where and how it manages to improve over it but I am unsure whether it was to keep the optimality of the algorithm or due to some oversight.

What went wrong.

1. My first implementation was severely flawed, and wasn't following the author's pseudo code as closely. This ultimately required a full rework down the line, taking up a lot of time. Again.
2. Overall time management. Due to this, the project ended up missing some key features. Ultimately this didn't take away from the project too much.

Outstanding/missing features.

1. Work Queue

This is the work I ended up dropping. This would be an incredible addition to the project, showing results closer to RTS games where players can control individual units.

Advice for similar projects.

Set up monthly deadlines depending on known free time. Update and adjust it as things come up. A big project comes up, then reduce the amount of available hours for that month.

Looking back, if I kept track of available time and amount of hours to get done per month I should get done, I would've been in a much better place than I was in the end.

If a student was to undertake a project in a similar area, I would recommend importing some of the maps used to benchmark pathfinding algorithms. They provide a much better and replicable search results and comparison to other algorithms. When browsing other papers, I noticed that they usually used those.

## References

Referenced Publication	Citation	Reference
Research paper <sup>1</sup> .	Zhang An 2016	<a href="https://www.sciencedirect.com/science/article/pii/S1000936116301182">https://www.sciencedirect.com/science/article/pii/S1000936116301182</a> Rectangle expansion A* pathfinding for grid maps
Research paper <sup>2</sup> .	Rafia Inam 2009	<a href="http://publications.lib.chalmers.se/records/fulltext/129175.pdf">http://publications.lib.chalmers.se/records/fulltext/129175.pdf</a> A* Algorithm for Multicore Graphics Processors
Website <sup>3</sup> .		<a href="https://blogs.nvidia.com/blog/2018/06/20/nvidia-ceo-springs-special-titan-v-gpus-on-elite-ai-researchers-cvpr/">https://blogs.nvidia.com/blog/2018/06/20/nvidia-ceo-springs-special-titan-v-gpus-on-elite-ai-researchers-cvpr/</a>
Research paper <sup>4</sup> .	Peter Yap	<a href="https://webdocs.cs.ualberta.ca/~holte/Publications/aaai11PeterYapFinal.pdf">https://webdocs.cs.ualberta.ca/~holte/Publications/aaai11PeterYapFinal.pdf</a>