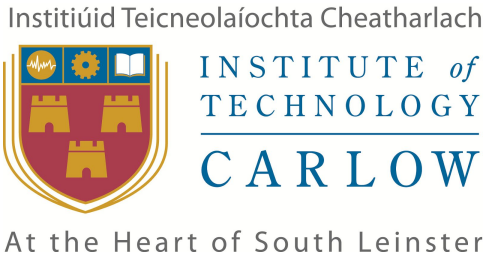


Computer Games Development CW208
SRS
Year IV

Przemyslaw 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

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.

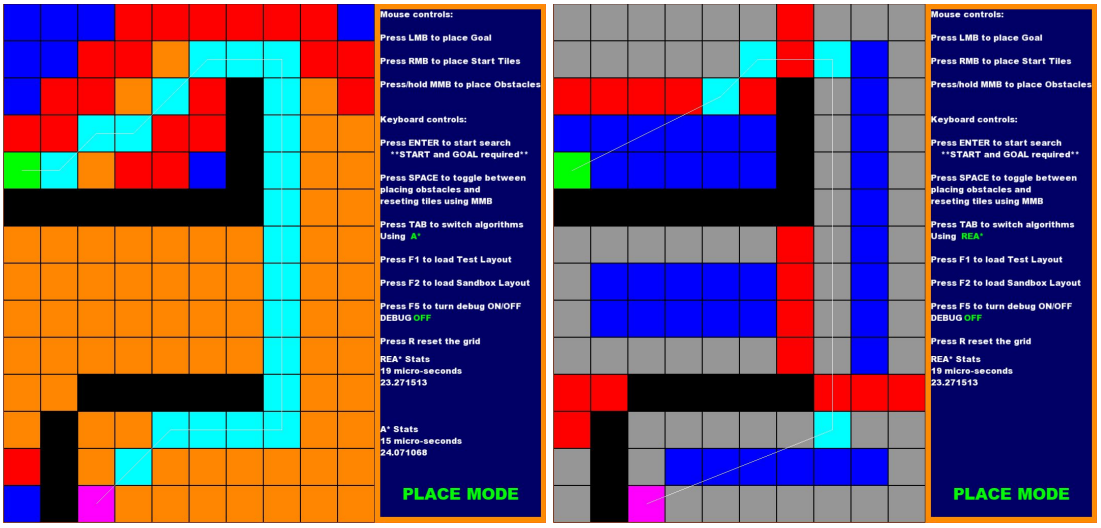
C++11 introduced multithreading to the language which allowed for quite a lot of functionality that Boost provides. This allows for multithreading features without needing to implement this huge library.
C++20 has a lot of features planned that will further help with introducing multithreading features 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.

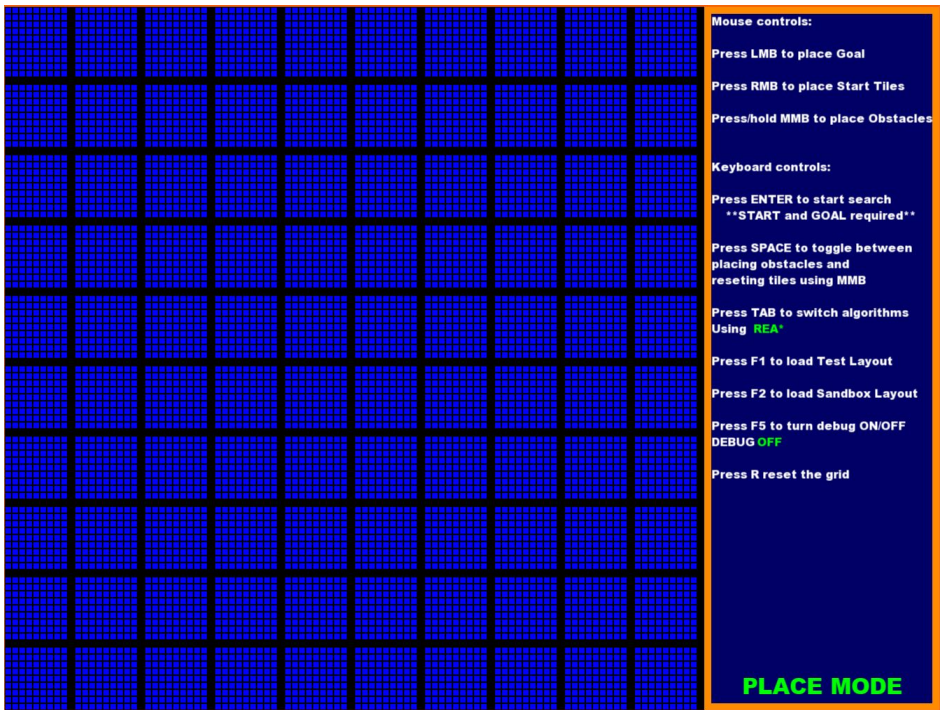
Project Description

Implemented feature set:

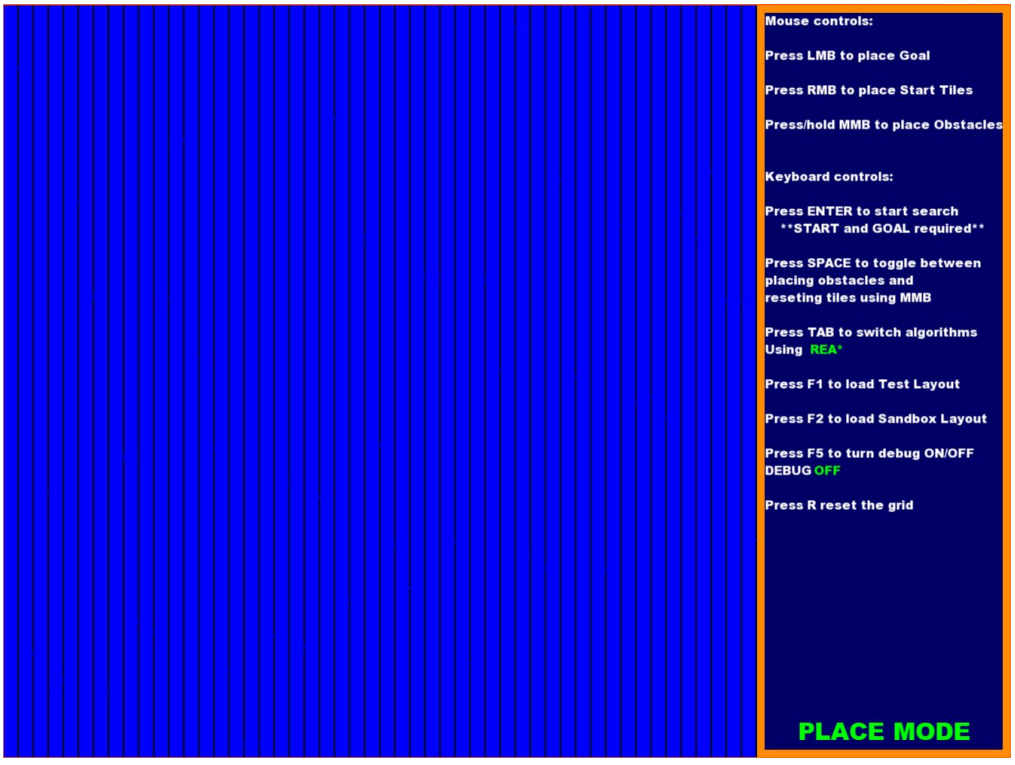
1. Dynamic grid with obstacle placement and deleting
2. Quick grid size changing using F1 to F4 keys (smallest to largest)
3. Toggle for “Debug” mode
    a. A* auto steps through
    b. REA* step through when N pressed
4. Quick Switching between A* and REA*
5. Time and Distance displaying
6. Test Layout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



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 3 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.

Define the Application

What is the application supposed to be?

My application is supposed to be used as a comparison between a relatively new algorithm REA* and the golden standard A*. These two were meant to be compared using several NPCs, with each NPC's pathfinding put in a threaded Work Queue.

What the application is.

The application allows to change between these two algorithms and compare them without the multithreaded work queue. Both algorithms are compared on the basis of time taken and path length on 4 different search spaces. Each search space has a different layout and total tile amount. Tile amount ranges from 140 to 1 million.

What is the application supposed to do

This application is meant show that each algorithm has its use cases and that A* shouldn't be blindly used for all games. Newer algorithms can be better suited for certain types of games. These games can range from extremely large search spaces, where A* doesn't perform well, to games where even the slightest difference in path length matters.

Metrics

My application gives usable and comparable metrics.

First metric is time taken in microseconds. I opted for microseconds as these give more detailed time without being too much. This allows to compare how long the algorithms took to find a path, if there is one, to the goal.

Second metric is the distance of the path. My application shows the distance of the path that the algorithms returned. This is used to determine which algorithms returned a shorter path and if the time difference justifies the longer/shorter path.

Project Milestones

I started off with a grid and A* implementation. A rough implementation was done between October and November. At that point I was on track with time and delivery.
REA*'s first iteration was implemented at the end of the Christmas Holidays. At that point I felt my pace had significantly slowed down. I was just slightly short of being behind on deliverables.

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.
At this point I was behind on deliverables. I should've had a second iteration, or fixes for the first one, at least started at this point.

After ARGO we had some deadlines 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.

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.