Research Report

Acknowledgments

I would like to thank Ross Palmer, my supervisor for providing me with a lot of reference materials and ideas.

Project Abstract

Pathfinding over the last few decades and resulted in a lot of different approaches while many solutions are more applicable depending on the computer application my particular deep dive into it will be finding the most efficient of these to use in a game and by extension compare a hybrid to some solutions individually

Project Introduction and Research Question

To investigate context-sensitive steering approaches and develop a hybrid AI which utilizes A* and context-sensitive steering in unison and compare it to other steering solutions. The entity will swap between the two systems and use both to navigate to a point somewhere in the simulation.

Background

[9]"Steering is a challenging task, required by nearly all agents in virtual worlds. There is a large and growing number of approaches for steering, and it is becoming increasingly important to ask a fundamental question: how can we objectively compare steering algorithms?". In the back of my mind I was constantly thinking of how to objectively measure how to accurately measure the differences between the different solutions and I ended up using my own version of a formula from this "*benchmark suite*"[9]. The original formula derived from the paper was "Si = wc,i(αC) + wt,i(βT) + we,i(γE) = C' + T' + E'"[9]. S simply represents "score" and essentially equates to C(collisions) + T(time) + E(effort efficiency). α , β and γ are used to normalise the values but haven't been standardised as of yet, *"Determining appropriate values of α, β, and γ is a challenging problem that we leave for future work."*[9] and w is an arbitrary value that refers to

the weightings of each value. For my project I have only added weight to collision. My formula was S = wC + T + E + P, with P referring to the distance travelled measured in pixels.

Literature Review

Pathfinding:

Pathfinding is computing the shortest route between two points in a computer application using some kind of algorithm or software. There are several different approaches to pathfinding and there are multiple variants of those approaches.There are also situations where the program knows where it's supposed to go and calculates the shortest path there. All pathfinding algorithms follow some fundamentals and may refer to them by a different name but will often serve the same purpose. To begin with an algorithm needs to have a starting point to search from some will use this starting point to compare the vertices(the cost associated with moving to that point)to the next point and others will simply use it as the starting point and calculate outwards checking every single point until they find what they're looking for and/or calculate the entire graph/map. I mentioned graph in my last point. Not all algorithms will use a graph but the likes of A* which I'll talk about later would where as simpler algorithms such as "seek" or "flee" don't require this as they simply need a point that they are seeking or fleeing from in order to calculate their velocity. Knowing where a program is supposed to go also makes it possible to compute bi-directionally(calculating the route from the end and starting point simultaneously and moving towards each other) as shown in Fig.1[2] This was conceptualized by a computer scientist Edsger W. Dijkstra [1] in 1959 and his algorithm appears unfocused and checks every vertex, working outwards until it finds its objective(Fig.2), which can be very time consuming.

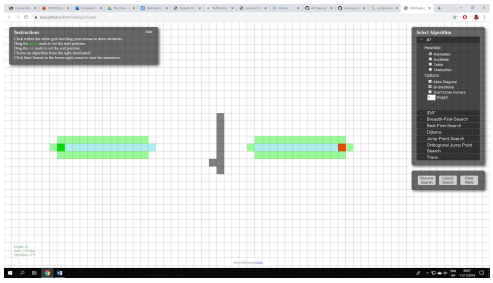


Fig.1

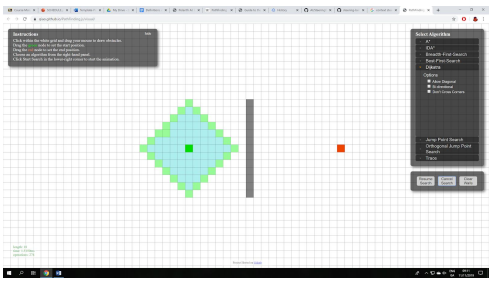


Fig.2

A*:

A* shown in Fig.1(example is bidirectional) is a graph traversal and path search algorithm. It works by storing nodes and calculating the routes between them all(heuristic), resulting in a "weighted-graph". This "weighted graph" or heuristic "represents a minimum possible distance between that node and the end. This allows it

to eliminate longer paths once an initial path is found."[5] It can be seen as an extension of Dijkstra's algorithm[3] shown in Fig.2[4]. "Dijkstra's algorithm works by visiting vertices in the graph starting with the objects starting point. It then repeatedly examines the closest not-yet-examined vertex, adding its vertices to the set of vertices to be examined. It expands outwards from the starting point until it reaches its goal"[4]. Greedy best-search first improved upon this by weighing which nodes were closer to the objective(Fig.3)[4]. Both of these solutions only work when there are no obstacles and are only accurate "when the map has no obstacles, and the shortest path really is a straight line"[4]. If we take a look at Fig.4[4]and Fig.5[4] we see that this is how Djikstra's and greedy best-search-first behave when objects are introduced.

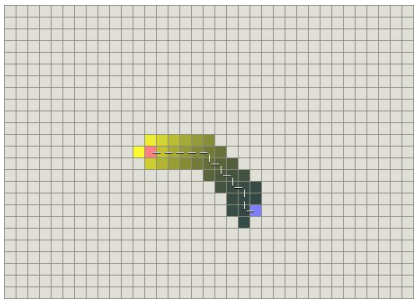


Fig.3

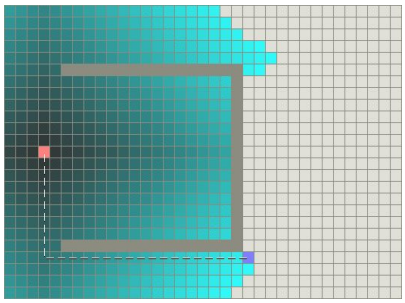


Fig.4

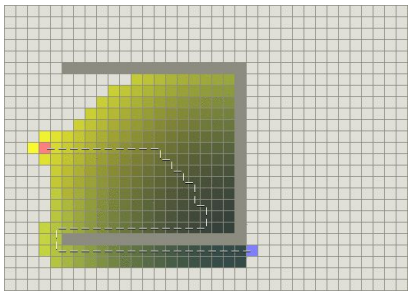
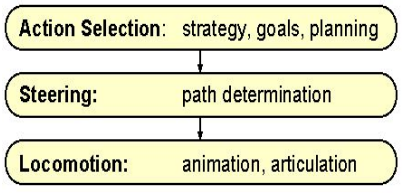


Fig.5

Context-free steering:

Context-free steering is specific to each game and by design does not take "context" into account when making steering decisions but instead performs calculations when a steering decision must be made. This can be very expensive the more objects you have in the game that needs to make these decisions and can provide results at the expense of smoothness and realism. Craig Reynolds[6] broke these decisions down into 3 parts: "Action Selection", "Steering" and "Locomotion". Action selection refers to the purpose or what needs to be done. An example of this would be seek which I mentioned earlier it's "Action Selection" would be to seek and move towards its intended target."Steering" is the "path determination" or pathfinding and "Locomotion" is the movement or adjustment itself and by extension in games the animation. Examples of AI context-free steering include the behaviours: seek(Fig.7), flee and flock. Seek and flee I've mentioned before,Flock works in a similar manner to seek except a group of objects will have one designated as the leader while the others will follow in whatever direction it goes with a bit of an offset. This is extremely useful for simulating bats or birds(Fig.8) or random enemies in a level.



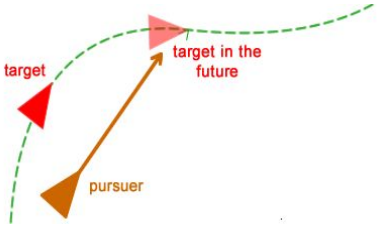
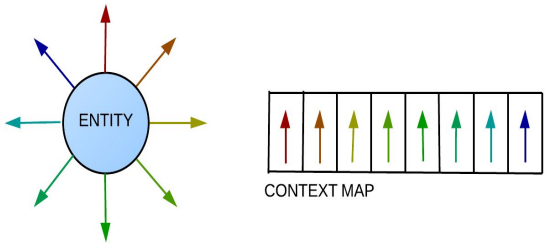


Fig.6[6] Fig.7[7] Fig.8[8]

Context-sensitive steering:

Context-sensitive steering tries to simplify the decision making process by representing the "context" of a decision numerically or "if the application requires a small number of entities that interact individually with the player, like a racing game"[8]. Andrew Fray explains how he did this on a game he worked on in his paper. The directions one can travel in, the desirable outcome and the danger are converted into a numerical value. This really simplifies the steering decision down into "go in the direction of the largest value" and really cuts down computational costs associated with steering algorithms. Direction can vary in complexity and will be specific to each game but will simply be a range of variables representing a given direction. The desirable outcome refers to what the autonomous body is trying to do such as chasing another entity or winning a race etc. Danger can be many things but in regards to steering it's primarily obstacles. In Fig.9 the circle represents the AI each different coloured arrow represents the different directions it can go in and are coloured simply for illustration purposes. The context map will be used for interest and danger with each direction represented numerically. In Fig.10 we can see an example of this in use. The largest direction on the danger map is the light blue arrow which we can see is directly perpendicular to the object. That would mean it's numerical value would be greater than any other direction so if the range was 0-1, then the blue arrow might be estimated at 0.8 and the only other arrow in the danger map would have to be less than that. On the interest map we can also see that the direction of the shortest path to a target has the largest arrow on the interest "context map" and again this arrow would have the largest numerical value in the interest "context map".



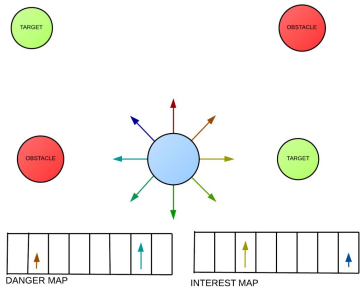


Fig.9[8] Fig.10[8]

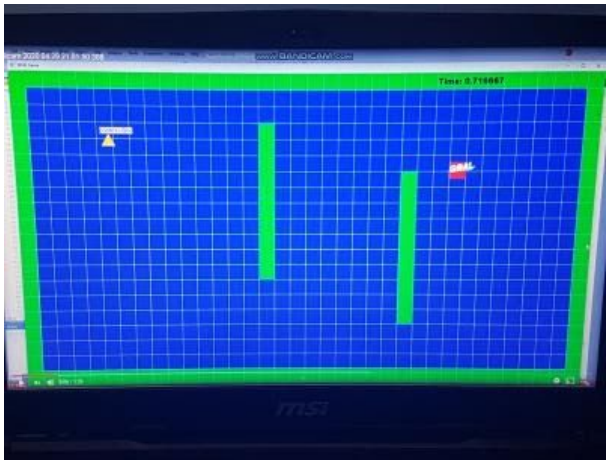
Study:

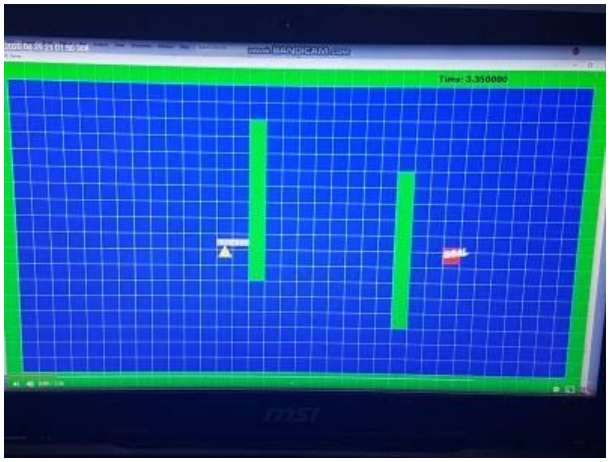
There were a couple of issues with my implementation that resulted in slightly inaccurate results but for the most part my simulation was quite standardised. I ensured that each of the AIs in the simulation began at the exact same position(100,105) and they all moved at the same speed which was 200 pixels. I made the decision to double the speed of the wander simulation because it could sometimes take 5-8 minutes to find the goal but I doubled the speed of time too.

Some issues which I should address should I continue the project is how I've recorded time. I opted for a manual counter as opposed to using delta time which is obviously flawed because even though the frames were limited to sixty per second it won't always be at sixty on every machine. My machine was pretty powerful though so for the most part it was at sixty consistently. Secondly all of the obstacles should have been dynamic to accentuate the advantages of the context-sensitive steering over the others and potentially include shapes that would give the other AI solutions a lot of issues like a U or a bowl shape. The final inaccuracy was I made the goal dynamic, which was the correct decision but my implementation left the goal moving throughout each simulation so there was a little RNG involved in where the goal would be at the start of any given simulation after the first.

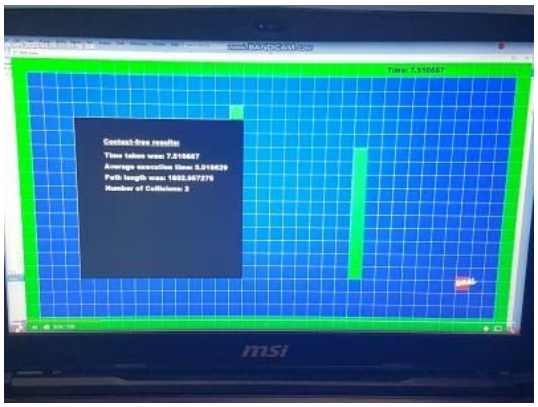
Project Description:

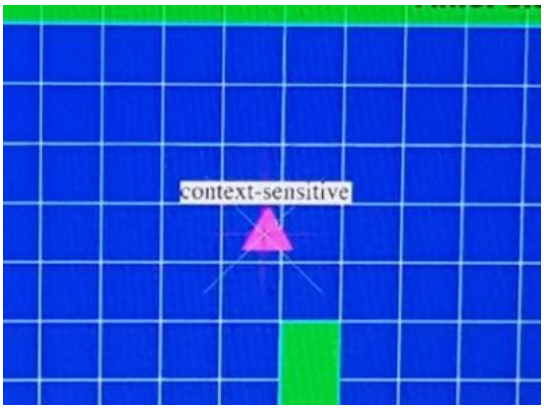
The final version of the project was a blue tile/grid surrounded by green walls. There were also two walls in the centre that didn't move. The red square labelled goal moved around and this is the object each AI tried to reach. The AI .





Between each simulation a black menu will appear and display the time taken, average execution time, path length and number of collisions.





The context-sensitive implementation does not work as outlined because I kept running into issues with raycasting that I couldn't diagnose what the issue was but simulation-wise the project looks like what I set out to do. Although I had originally planned for all of the AIs to run at

the same time but my supervisor suggested the change of the running separately and I'm very happy with the change. It look a lot less cluttered.

Project Milestones

16-10-19: Started Project.
23-10-19: Added base classes and created game loop, constructors and render functionality.
30-10-19: Added grid based maze and obstacles.
11-11-19: Implemented context-free steering and collision detection.
09-12-19: Added Context-sensitive prototype and drafted SRS and TDD.
12-02-20: Reformatted simulation and changed AI functionality.
18-03-20: Lockdown in effect, had some issues with the new machine and had to change the dimensions of everything. Additional work on context-sensitive steering. Added a vertex array representing 8 directions(north, north-east etc).
25-03-20: Created the functionionality that would allow raycasting to objects as well as calculating distances and getting a unit vector.
22-04-20: Implemented seek and wander AIs and added labels to each of the AIs.
29-04-20: reformatted the entire simulation so instead of the AI all running around at once they took turns, also calculated scores and displayed them between each simulation and compared all at the very end.

For the most part the schedule was not adhered to. I met most of my deadlines but a lot of issues such as getting stuck on the raycasting meant I context-sensitive steering kept having to be revisited and my initial proto-type had to be completely redone as it was just incorrect.

Results and Discussion

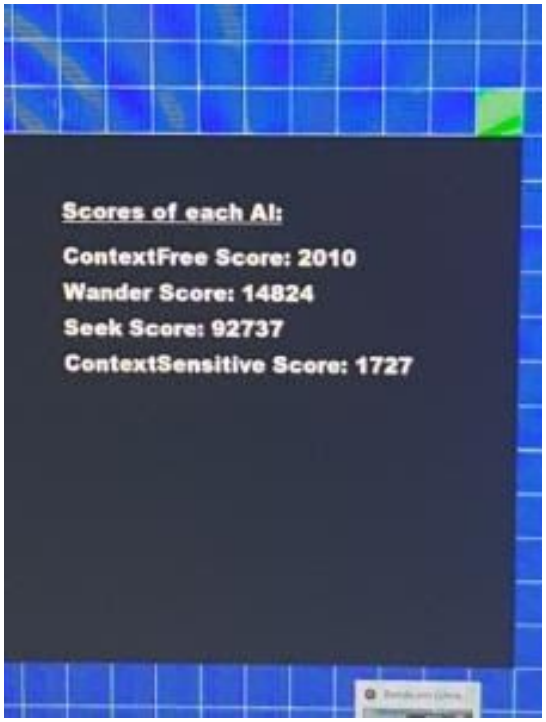
As mentioned in the study part of my report there were a couple of factors that mean my results have to be taken with a pinch. There were also some issues with calculating the path length for seek as it travels along a curve but even with all those consistencies the context-sensitive AI seemed to win outright once there was a little weighting implemented to collisions which I valued at 200.


ContextSensitive results:
Time taken was: 8.000000
Average execution time: 0.016631
Path length was: 1719.880127
Number of collisions: 0


Wander results:
Time taken was: 44.733334
Average execution time: 0.016660
Path length was: 11279.266602
Number of collisions: 18


Context-free results:
Time taken was: 7.516667
Average execution time: 0.016629
Path length was: 1602.957275
Number of Collisions: 2


Seek results:
Time taken was: 9.233334
Average execution time: 0.016636
Path length was: 92727.820313
Number of collisions: 0



Although the margin is fairly slim and without the weighting on collisions context-free would probably win out by a hundred points or so I feel like with a more dynamic environment or the more difficult shapes I outlined earlier in the report would accentuate the strengths of the context-sensitive steering particularly a “U” shape as the context-free AI would go into the U and could potentially get stuck in there whereas the context-sensitive would never enter.

Project Review and Conclusions

What went right: While it could definitely have been improved upon setting benchmarks in an attempt to standardise the test as much as possible was successful and the more standardised the test became the more I learned and the more accurate the results became. Unfortunately the smallest thing could somewhat corrupt the results but once they’ve all been addressed I feel the data would be extremely accurate with time being the important one because working off frames instead of time was obviously flawed. I wish I had done a little more research on standardizing tests before designing the project.

What went wrong: Time management was pretty poor, When large group projects that spanned a week or two came up, there was little to no work done. Some of the code functionality was in the wrong place and this led to some very dirty code close to crunch time which could have easily been avoided. The context-sensitive AI should have been addressed from the first week as it wasn’t apparent until later how many issues there would be. I definitely should have spent more time proto-typing and maybe having a look at some public repos of other peoples solutions, would have saved me a lot of time in the long run. I’m still happy with what I’ve learned about context-sensitive steering and steering approaches as a whole however.

What is still outstanding: The context-sensitive AI is incomplete and operates closer to context-free with a little bit more knowledge. I had originally planned to implement a hybrid AI(context-sensitive & A*) but as I failed to implement the context-sensitive AI working properly I didn't in the end. I also wanted to set up the simulation in such a way that someone could dynamically change the environment at run time but never got around to it.

If starting again how I would approach: I would definitely load the maps using a JSON file and write cleaner implementations instead of confining a lot of the functionality to my game class. I would spend a lot more time looking at other implementations and repositories, particularly context-sensitive solutions. I would manage my time better and get the bulk of the project done earlier so if I encounter any issues I can handle them earlier.

Advice for someone attempting a similar project in future: Create a function for calculating distance and unit vectors or include a library that does so as you'll end up doing it a lot. Ensure you use delta time and not counters for measurements. Somewhat randomise steering events instead of creating AIs that favour one direction(my seek AI always steered down upon collision). Use a "look ahead vector" instead of collision detection to more accurately depict results and maybe smooth steering over several frames.

Were your technology choices the right or wrong ones?: Not really but I feel like I definitely could have utilised some libraries to save myself some work such as the THOR library but there's never any harm in doing things yourself as long as you've done it correctly. It definitely would have been a little easier to do in csharp but I mostly fell down in poor time management.

If a student were to undertake a similar project but their aim was to investigate the kind of shapes that give standard AI solutions the most issues and quantify that I feel like it could contribute to the improvement of some game designs. People in the gaming community all have some experience with "cheesing" an AI of some kind by standing on or near some object and abusing the fact the AI can't traverse the object.

References:

[1]In 1956, Edsger W. Dijkstraitte created a "shortest path first" algorithm and published it in 1959. Pathfinding, wikipedia accessed at: https://en.wikipedia.org/wiki/Pathfinding (accessed 21st Oct 2019).
[2] Pathfinding simulator accessed at: https://qiao.github.io/PathFinding.js/visual/ (accessed on Nov 6th 2019).
[3] It was published by Peter Hart, Nils Nilson and Betram Raphael in 1965. Pathfinding, wikipedia accessed at: https://en.wikipedia.org/wiki/Pathfinding (accessed 21st Oct 2019).

[4] A* comparison at theory stanford accessed at: http://theory.stanford.edu/~amitp/GameProgramming/AStarComparison.html (accessed on 9th Nov 2019).
[5]Pathfinding, wikipedia accessed at: https://en.wikipedia.org/wiki/Pathfinding (accessed 21st Oct 2019).
[6] Craig Reynolds GDC paper 2019 accessed at: https://www.red3d.com/cwr/steer/gdc99/ (accessed on 9th Nov 2019).
[7]Understanding steering behaviours at gamedevelopment.tutplus.com accessed at: https://gamedevelopment.tutsplus.com/tutorials/understanding-steering-behaviors-pursuit-and-evade--gamedev-2946 (accessed on 9th Nov 2019)
[8] Context sensitive steering Andrew fray Accessed at: https://andrewfray.wordpress.com/2013/03/26/context-behaviours-know-how-to-share/ (accessed on 26th Oct 2019)
[9] SteerBench: a benchmark suite for evaluating steering behaviours accessed at: http://www.cs.uu.nl/docs/vakken/mcrws/papers_new/Singh%20et%20al%20-%202009%20-%20SteerBench.pdf (accessed on 29th April 2020)