Project Evaluation Report

Artificial Intelligence for Games

Academy of Interactive Entertainment

Roy Batty, August 17 2019

# Project Overview

Use this section to give a high-level overview of your project and its development.

**Briefly (in one or two sentences) describe what your project is.**

The project is a simple simulator that allows the user to spawn in agents and other objects as well as a build a level. The user was able to modify agent’s attributes. Some of these attributes include movement speed and movement costs.

Then address the following questions in report form (i.e., write well-formed paragraphs that have a logical flow, taking note to avoid spelling or grammatical errors).

* **Did development adhere to your pre-planned timeline?**

Development for the most part did indeed adhere to the original pre plan. Minor changes were made such as the project transitioning from a maze solver app to a maze runner like app with enemies.

Originally, the simulator was supposed to take an image of a maze. The A\* algorithm will then solve that maze and show the steps to solve it.

Now there a different agent and powerups such as a speed boost, player agent, zombie agent and chaser agent. The player agent uses A\* to go to wherever the user left clicks.

* **What A.I. algorithms did you implement, or attempt to implement?**
* **What difficulties did you have in implementing these algorithms?  
  Possible difficulties worth mentioning might include:**
  + **Difficulty in understanding all details of the algorithm**
  + **Difficulty with programming / debugging**
  + **Performance issues, including memory management**
  + **Unexpected or incorrect agent behaviour**

A\* was the chosen algorithm to implement. Writing out the code was the easy part; the more challenging part was fixing up bugs. A\* was the chosen algorithm as it seemed at the time to be the most suitable.

The reason for this was because an agent would always have a start and end point. Therefore, there was no need to check a large quantity of nodes for no reason, just the neighbouring nodes from point A to point B.

Some bugs which needed to be fixed during implementation was the path node distance calculations and making sure bad nodes were skipped when getting neighbour costs.

Since a little bit of blackboard behaviour was implemented, it also meant the number of agents to blackboard had to be limited. Whilst testing many agents (around 300), FPS would have a major drop once every single agent was told to calculate a path to the player.

One solution which was used at the time was to just limit the number of agents that could be blackboarded at that given time. This meant that out of the 300 agents, only 100 would move to the player’s position.

# Performance Analysis

Use this section to analyse the performance of your algorithm(s) or techniques.

**Provide a brief description of the memory footprint of your agent class(es). Explain if this is efficient or could be improved upon.**

**Analyse you A.I algorithm and identify any performance bottlenecks or places for improvement. If possible, list the efficiency of your algorithms using Big O notation.**

Possible topics for inclusion in this section are:

* **Is it efficient for a lot of agents to use the same pathfinding algorithm?**
* **Should pathfinding be done every frame?**
* **How can you improve the performance of your pathfinding algorithm in the context of your game?**
* **Are all your algorithms efficient? Why/why not.**
* **How many agents could you have in your game before you start seeing performance issues, and have you tested this?**

The memory footprint of the agent class was efficient but has room for improvement.

Chaser agents have a static list of chaser agents as a variable. This means they have no reason to communicate with other classes thus reducing overhead. Since it is static, this also means that every agent shares a single pointer to one List, instead of every agent having its own instance of the list.

An improvement would probably be to split up an agent into multiple classes to follow Unity’s component-based architecture.

Around 800 agents were when the game’s performance was heavily impacted. Fps was averaging around 8. Fortunately, there isn’t really any way for the user to spawn that many, unless they want to spend a long time right clicking over 800 times. Surprisingly, this was not because of the A\* star algorithm, but instead due to Unity’s sprite renderer and box collider.

When sprite rendering and box colliders were disabled, it was able to handle about 1800 agents before fps dropped heavily.

The A\* algorithm was used effectively, as it was made sure that a new path was not calculated every frame, but instead after a set number of times in seconds.

Every agent has a movement speed variable, which is how many seconds until can they take their next step towards their goal path. An agent will not be required to recalculate its path until it reaches the goal.

This means that an Agent that has been put in the patrol state calculates its path once. Once the agent has reached its destination, or for whatever reason cannot, it will then calculate a new path. This is a lot more efficient than constantly calculating the same path every frame for no reason at all.

The A\* algorithm currently has the big o notation of O(n^4). This is because whilst it is going through the open list, it must then loop through each node in the open list to find the lowest cost node. Then it must go through a nested for loop to create a list of neighbouring nodes. Finally, it will then have to loop through that neighbouring node list.

An improvement for the A\* Algorithm would be to incorporate flow fields. Flow fields would have reduced the need for every single chaser agent to calculate their own paths. They instead, would have shared information between each other, which is a lot less overhead when compared to A\* path calculation.

Overall, the design choices and use of the A\* algorithm were adequate. They got the job done and fit their purpose in this project. There are a lot of ways to further improve the design, however, that will not be necessary unless this project were to simulate an absurdly higher number of agents at a given time.

# Future Improvements

**Did you get enough time to completely implement your A.I. as planned? What work did you not complete (and why)?**

**Can you see ways to improve your program/algorithms?**

**Were your algorithms good choices? Do you plan to use them in future projects (why/why not)?**

The AI was implemented as planned with the time provided. This is because during the design phase, attempts at implementing the A\* algorithm was already made.

Unity also cut a lot of the workload since all that needed to be implemented were the algorithms as well as agent class structure.

I would use A\* in the future whenever required to make a 2D based game that requires pathfinding (such as an RTS). It is an efficient and easy to use algorithm for a lot of different use cases.

One thing for next time would have to be implementing proper steering movement instead of tile-based movement. Steering movement would give more life to the agents and just be overall more satisfying to see.

Flee would also be improved as it ended with just the agent trying to run to a random position.

One big improvement to keep in mind would be to utilize Unity’s DrawMesh function for the tiles. This will mean that less work would be required rendering each Game Object individually thus heavily improving performance.