**Project Evaluation Report**

**Artificial Intelligence for Games**

**Fredrick Bancan, 18 OCT 2020**

# 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.

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?
* 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
* If you did not experience difficulties implementing the A.I algorithms, then explain why you feel these algorithms were easy to implement

**My project is a 2D top down stealth game where the player must navigate a maze/base. The player must obtain a special object in the base and return to their starting position. The special object is guarded by enemy ai which can path towards the player and upon touching them, the player loses.**

**In development I mostly did adhere to the pre-planned timeline apart for some necessary changes, such as re-doing the enemy ai flow chart.**

**The enemy uses a path finding algorithm (Dijkstras shortest path) which I successfully implemented. However, it was difficult to gain intuition on how to properly implement said algorithm. In the process of implementing it, I also had to debug my previously implemented navigation node system and ray vs AABB detection.**

**Upon successfully implementing the path finding algorithm, it was quite slow and caused freezing when calculating a path. I later made two major optimizations; the largest was making it so path calculations stopped when the end node was found. Secondly I replaced a std::list being used with a single array of Booleans which is initialized on launch. The Boolean array instead of the original std::list reduced a part of the algorithm from O(n) complexity to O(1) complexity.**

# 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?

**My A.I logic loops are performed on a on-tick basis. In the context of my game this is limited to 30 times a second. The main portion of the A.I logic is done using a finite state machine style, keeping it relatively clean and fast. The path finding has been optimized to an extent where I am not aware of any more optimizations to be done.**

**Paths are calculated only once on average every few seconds or so in real time. There are no paths being generated per tick or per frame, however, line of sight with the player is tested each tick when an enemy is seeking the player.**

**One area which could be improved is AABB collisions and Ray vs AABB testing. Currently when testing collisions and rays, each AABB or Ray is tested against every wall in the level. This is O(n). With certain culling techniques, such as space partitioning, this could be reduced closer to O(log n) complexity. Rays are particularly important in the A.I algorithms since they are used to determine line of sight with the player and detecting the player each tick.**

**On the machine I developed the game on, my code supported about 200 agents at once before there were any noticeable performance drops. Though I did not determine if this was due to the rendering framework I was using to render the agents. A picture containing engineering drawing

Description automatically generated**

# 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)?

There is no word limit for this report, but it is expected that you provide enough detail for 1 to 2 full pages.

**The A.I was implemented as planned, however it was later than desirable. In the future, better time management for the developer(me) would improve or remove this issue.**

**It is difficult to find any ways to further improve the program and algorithms apart from further abstraction of code in certain areas for neatness and readability, and implementing rigorous culling techniques for AABB and ray calculations. However, for the scope of this project, these seem to not be necessary. On my machine this game ran at ~2000 frames per second with the standard 12 enemies in the level. Dijkstras could be replaced with a different faster path finding algorithm, such as A\*, but I only had time to study and implement the more simplistic Dijkstras algorithm for this project.**

**I ended up liking the simplicity and effectiveness of Dijkstras algorithm for path finding and may use it in similar scenarios in future applications. The line of sight and finite state logic for the Enemy A.I proved to be very effective, and I will definitely be using this in future projects.**