

Final Year Project Report

Full Unit – Final Report

Labyrinth: Single-agent and multi-agent search in maze games

Thomas Roscow

A report submitted in part fulfilment of the degree of

BSc (Hons) in Computer Science

Supervisor: Farid Shahandeh



Department of Computer Science
Royal Holloway, University of London

April 05, 2024

Declaration

This report has been prepared on the basis of my own work. Where other published and unpublished source materials have been used, these have been acknowledged.

Word Count: 9,200

Student Name: Thomas Roscow

Date of Submission: 05/04/2014

Signature: T R . Roscow

Table of Contents

Abstract	3
Project Specification	4
Chapter 1: Introduction.....	7
1.1 Single-agent and multi-agent search in maze games	7
1.2 Aims of this project	7
1.3 Project overview.....	8
Chapter 2: Professional Issues: neurodivergence and project management	9
Chapter 3: Background and Literature Review	11
3.1 Importance of maze games in artificial intelligence research.....	11
3.2 Why games?	11
3.3 Multi-agent search algorithms.....	12
3.4 An overview of the graph search algorithms used.....	13
3.5 Recent advances	14
Chapter 4: Methodology	15
4.1 Tools and technologies	15
4.2 Software engineering techniques	15
Chapter 5: Implementation	18
5.1 Graphical visualisation of the maze game	18
5.2 Depth-first and breadth-first search for maze navigation	20
5.3 Uniform-cost graph search for maze navigation	23
5.4 A* algorithm for efficient path-finding.....	25
5.5 Comparison of A* and greedy search for single-agent path-finding.....	26
5.6 Multi-agent search with a reflex agent.....	28
5.7 Minimax algorithm for adversarial multi-agent search	30
5.8 Alpha-beta pruning for more efficient Minimax search	31
5.9 Expectimax algorithm for multiple agents with probabilistic behaviour	32
Chapter 6: Discussion	34
6.1 Overall evaluation and challenges encountered.....	34

6.2 Future work 34

6.3 Self-evaluation 35

Bibliography 37

Appendix..... 39

Abstract

Single-agent and multi-agent search algorithms have long been used for various graph applications including pathfinding, cooperation and competition that are applicable to a variety of real-world problems. Developing and testing them relies on smaller, controllable environments in which agents' behaviour can be understood and analysed, and maze games offer an ideal testbed for that. In this project, several single-agent algorithms were implemented to find efficient paths through a two-dimensional maze, and multi-agent adversarial algorithms were developed to model competition between agents. This report details the scope of the maze game, a background to search algorithms, the algorithms implemented, and a discussion of how successful the project was. It also discusses some professional issues related to the progress of this report concerning neurodivergence and project management.

Project Specification

Aims: The goal of this project is to implement general search algorithms and apply them to single-agent and multi-agent scenarios in maze games (e.g. Pac-Man).

Background:

Maze games are a video game genre in which the entire playing field is a maze. Usually, the player needs to navigate the maze to collect items and also performs other actions (e.g. escape monsters and outrace an opponent) within a time limit. The most popular example is Pac-Man by Namco (1980), but many other maze games exist.

You will model a maze game as a search problem and then design and implement agents that inhabit the maze. The agents will find efficient paths through the maze world, to reach a particular location, to collect food and to combat opponents.

Early Deliverables

Given a maze game of your interest (e.g. Pac-Man), design and implement a simple graphical visualisation for it.

Consider an agent inhabiting the maze world. Model the agent's navigation task as a search problem. Implement a graph search version of depth-first search and breadth-first search to reach a specific location in the maze. Run experiments with three different size maze: small, medium and large. Visualise the agent following the plan step-by-step.

Now change the cost function in such a way that traversing some areas is more expensive than traversing other areas (e.g. Pac-Man: you can charge more for dangerous steps in areas close to the ghosts). Implement the uniform-cost graph search algorithm to reach a specific location in the maze. Run experiments with three different size maze: small, medium and large. Visualise the agent following the plan step-by-step.

Final Deliverables

Implement the algorithm A* to find the shortest path through the maze that touches all four corners. Use two different, consistent heuristic functions. Run experiments with three different size maze: small, medium and large. Visualise the agent following the plan step-by-step.

Assume each cell in the maze has an item to collect. Formulate a search problem for collecting all items (e.g. in Pac-Man: eating all the food) and implement A* to do that in as few steps as possible. Also implement a greedy search and compare the results of the two algorithms in three different size maze: small, medium and large. Visualise the actions of the agent.

Now assume that you have additional agents in your maze, one is the main player and the other are the opponents (e.g. Pac-Man and the ghosts). Model this new problem and implement the player as a reflex agent that tries to visit all cells avoiding the opponents.

Implement an adversarial Minmax search agent for each agent; start with the player and one opponent and then increase the number of opponents. You need to write an evaluation function that evaluates states. Run experiments with three different size maze: small, medium and large. Visualise the actions of the agents.

Design and implement a new agent that uses Alpha-beta pruning to more efficiently explore the minimax tree. Run experiments with three different size maze: small, medium and large. Visualise the actions of the agents.

Design and implement an Expectimax agent to model the probabilistic behaviour of an agents who makes suboptimal choices. Run experiments with three different size maze: small, medium and large. Visualise the actions of the agents.

Suggested Extensions

Use value iteration and Q-learning to underpin the behaviour of the player.

Chapter 1: Introduction

1.1 Single-agent and multi-agent search in maze games

What are single-agent and multi-agent search?

An agent is “a computer system that is situated in some environment, and that is capable of autonomous action in this environment in order to meet its delegated objectives.” [1] An environment may be single agent or multiagent. An entity must be viewed as an agent when “[its] behaviour is best described as maximising a performance measure whose value depends on [another agent’s] behaviour.” [2] When an agent needs to select actions in environments that are deterministic, observable, static, and completely known, “the agent can construct sequences of actions that achieve its goals; this process is called search .” [2]

Why are single-agent and multi-agent search important/interesting/relevant in the real world?

The real world has environments that can be modelled with agents with problems that can be modelled as a search. Examples include Internet search engines, e-commerce, air traffic control systems, particle accelerator control, managing networks of spatially distributed sensors, and a business providing customers with quotes.

What are the challenges around search problems and their real-world use cases that make this area suitable for research and development? In other words, why is it not a straightforward problem with a straightforward solution?

Why are maze games used for search problems?

What are some important characteristics to consider for (maze) games when developing and testing single-agent and multi-agent search algorithms?

Briefly, what did you do in this project?

1.2 Aims of this project

What were the aims of this project?

How did the aims translate into things that you had to carefully consider? For example:

- choosing a maze game that is playable by both a single agent and multiple agents
- choosing a maze game that is non-trivial to solve and where several search algorithms could be applied and compared
- choice of programming language

- structure of the repo

How might this project help in your future career? (Some things to consider: programming skills; personal growth; developing an understanding of AI; independent, computational thinking about how to solve a problem; critical thinking when it comes to testing/debugging/optimising code; project management; self-directed learning) [200 words / 15 minutes]

1.3 Project overview

1.3.1 Choice of maze game

What maze game did you choose, and why?

Labyrinth, a board game.

1.3.2 How to run

<https://youtu.be/nbIiXv15DiE>

Chapter 2: Professional Issues: neurodivergence and project management

Attention deficit disorder (ADD) or adult attention deficit (AAD) commonly produces five symptom clusters: [4] [5]

- difficulty activating and organising work;
- difficulty sustaining attention and concentration;
- difficulty sustaining energy and effort;
- difficulty managing emotional interference;
- difficulty utilising working memory and accessing/recalling learned material.

Research has found a negative association between AAD and the operational effectiveness of project managers. [5] Research has found that on average disordered adults have lower household incomes at post-secondary and post-tertiary graduate level, and that annual income loss is similar to losses associated with drug and alcohol abuse. [6]

Coetzer [5] concludes that organisations need to proactively support employees with project management responsibilities to ensure timely completion of key tasks and the success of projects, due to the influence of AAD on time management capabilities and the operational effectiveness.

Neurodivergent individuals bring unique perspectives and strengths to computer science. Research suggests a positive association between ADD and creative achievement.

Reformulating a description of this project's requirements from the point of view of someone with ADD...

This project is self guided six months which ends on its own work with little oversight

Unstructured. Requires lot of design choices taking into account diff factors and open ended reading and synthesis of background information.

All these require the executive function that students with ADD lack. For example, time blindness which I suffer greatly from – at beginning you need to have a plan and break down into smaller tasks. Second barrier is even when having a plan, it's finding the self-motivation to work on non-urgent tasks. The appropriate and timely switching to other obligations (other deadlines). Background reading is compounded by this including time blindness, it requires organisation to select read and organise materials, and time management to complete these tasks.

A doctoral thesis entitled “Supporting Students with ADHD in Project-Based Learning” [7] whose literature review highlights some ways to make project-based learning more ADHD-friendly:

- shorter assignments / breaking up assignments into short pieces with short time limits
- consistent routines
- clear student expectations
- frequent and ongoing feedback

As a result, how was your progress and/or the outcomes different from your peers'?

Despite planning, two plans one in beginning and one in middle, problems with executive function meant that first commit was not made until 16 March, only twenty days before deadline. This impacts on quality of project.

What might be the consequences of not making appropriate adaptations for people with ADD (or other neurodivergence) on society?

Students with ADD become workers with ADD. Without appropriate adaptations and support, impacts include stress, low self-esteem and poor professional outcomes. For example, this study estimated that such workers earn \$4k-\$11k USD less per year. [8]

Lack of an appropriate project management framework for workers with ADD can easily result in missed deadlines, failure to meet expectations of employers collaborators and stakeholders, and in the computing world poor or rushed code quality. This has been described as untapped potential by study [8].

In professional computing settings, there is a range of project management frameworks that are commonly used, such as Agile, Scrum, Waterfall, Kanban. Their purpose is to improve the effectiveness of software professionals, teams and organisations. Typically they involve a software or tool and processes for planning and coordinating projects, and often require a person to keep a team on track. For example a project manager, scrum master or agile coach.

How appropriate is each of these project management frameworks for people with ADD or other kinds of neurodivergence?

Trello boards for setting tasks has the aim of making things easier for someone with ADD. Four key things it helps. The practice of breaking down into smaller tasks, which helps with executive function. Having a schedule or board or visual tool to encourage following the plan. Third is the agile processes so that even if you have a bad week the next one can be better. Fourth, and this is most important for myself, is having a person as a project manager. This is to tackle the problem of time blindness. Without a coach or mentor, such little work is done; with a mentor, progress can get on quickly, realising actionable objects, to unblock the person and makes the problem and the solution external. This also extends to other types of neurodivergence where having clear structure and expectations leads to positive outputs, this applies to autism for example among others.

In summary, how could a suitable project management framework have improved the outcome of this project?

As an example, in this project if these frameworks and mentor had been in place from the beginning, the product would have had more time and would have been more complete, and I would have had significantly less stress and other negative effects.

Chapter 3: Background and Literature Review

3.1 Importance of maze games in artificial intelligence research

Maze games play a significant role in artificial intelligence research, acting as a valuable tool for developing and testing AI algorithms.

They are used for developing problem-solving skills. Maze games are essentially puzzles that require navigation from a start point to a goal. This basic framework can represent many real-world problems. AI research uses mazes to develop algorithms that can solve complex problems by finding paths through obstacles, which is like navigating through the complexities of real-world environments.

Maze games are excellent for developing and testing pathfinding algorithms, like the A* algorithm. These algorithms are not just applicable in games but are used in robotics for navigation, in logistics for optimising routes, and in network traffic management.

Maze games provide a controlled environment for applying reinforcement learning techniques. In reinforcement learning, an agent learns to make decisions by performing actions and receiving feedback in the form of rewards. Mazes, with their clear objectives and possible penalties, are perfect for training AI models to learn strategies for exploration, navigation, and eventually reaching the goal efficiently.

As AI research advances, mazes become more sophisticated, simulating complex and dynamic environments. This helps in developing AI systems that can adapt to changing conditions and learn from interactive environments, essential for real-world applications like autonomous driving.

Maze games offer a standard platform for benchmarking AI algorithms. Researchers can compare the efficiency, speed, and adaptability of different AI models in navigating mazes, providing a clear metric for assessing advancements in AI techniques.

For students and newcomers to AI, maze games are an accessible and engaging way to learn about AI concepts and algorithms. They provide a hands-on experience in programming AI agents, understanding the principles of machine learning, and appreciating the challenges in AI research.

Successfully navigating through mazes requires an AI to generalise from past experiences and apply knowledge to new, unseen mazes. This aspect of maze-solving is crucial for developing AI systems capable of transfer learning, where a model trained on one task is adapted for another related task.

In summary, maze games are more than just a testing ground for AI; they are a microcosm of the challenges and opportunities in AI research, providing a clear, controlled, and scalable environment for developing, testing, and improving AI algorithms across a wide range of applications.

3.2 Why games?

Games provide an ideal trade-off between simplicity and complexity for artificial intelligence research. They are complex enough to pose challenges in planning, optimisation, decision-making that replicate real-world problems, but are controllable and well-defined enough that researchers can test them thoroughly and make robust and reproducible conclusions about their performance.

As Yannakakis & Togelius in the book cited below say, games “are formal and highly constrained, yet complex, decision making environments.” They are also useful for benchmarking against human performance, in use cases where an artificial intelligence algorithm needs to automate human decision-making without loss of accuracy.

There are a few interesting anecdotal examples of AI advancements being made with games:

- The modern field of deep reinforcement learning (a branch of artificial intelligence) was kick-started by the success of deep Q learning for solving Atari 2600 games based on pixel inputs - no explicit knowledge of the game structure at all
- The first computer to reach superhuman capabilities in chess (IBM’s DeepBlue) was underpinned by the Minimax algorithm
- Using games for AI research dates back to the very beginning: Alexander Sandy Douglas developed an algorithm for playing noughts and crosses as a computer game in 1952

Relevant references:

- <https://www.gameaibook.org/book.pdf> See the section starting on page 15, “Why Games for Artificial Intelligence”
- <https://arxiv.org/abs/1312.5602> the aforementioned deep reinforcement learning paper

3.3 Multi-agent search algorithms

What is the relevance of graph search algorithms and multi-agent systems for real-world use cases?

Most obviously, for game design itself. Although the main character(s) are usually human-controlled, non-playing characters are given behaviours that are interesting, intelligent and efficient by using some of these algorithms.

Path-finding and navigation: treating a transport system (for example, roads) as a graph allows you to find the most efficient route from one place to another. A* is commonly used to plan routes with GPS systems

Autonomous robots: including drones, self-driving cars, Roombas and search-and-rescue disaster response. They need to autonomously (i.e. using their own algorithms, not remote-controlled) find paths that are efficient, safe, and avoid obstacles, and dynamically update their routes as new information appears (e.g. a new obstacle) or priorities change. Often they need to collaborate with other autonomous agents too

- <https://www.academia.edu/download/91617823/13140.pdf>

Network analysis: any network that can abstractly be represented as a graph is amenable to graph-search algorithms, even if there isn’t a spatial layout to literally “navigate”. For example:

- finding the shortest path between two people in a social media network (for bot detection and tackling misinformation)
- hyperlink analysis of links between webpages to understand flow of information
- networks of people, for modelling infectious disease spread, or the Oracle of Bacon
- computer networks: breadth-first search is used for peer-to-peer networks like BitTorrent

3.4 An overview of the graph search algorithms used

Algo name	Time complexity (V nodes and E edges, b branching factor, d depth)	Guaranteed to find optimal path?	Applicable to weighted graphs?
Breadth-first search	Linear $O(V+E)$	Yes	No
Depth-first search	Linear $O(V+E)$	No	No
Uniform cost	Log-linear $O((V+E) \log V)$	No	Yes
A*	Linear (depending on heuristic) $O(E)$	Yes (depending on heuristic)	Yes
Greedy	Log-linear (depending on heuristic) $O((V+E) \log V)$	No	Yes
Minimax	Exponential $O(b^d)$	N/A	N/A
Minimax with alpha- beta pruning	Exponential $O(b^d)$	N/A	N/A
Expectimax	Exponential $O(b^d)$	N/A	N/A

References for these search algorithms:

- Breadth-first search: Moore, Edward F. (1959). "The shortest path through a maze". Proceedings of the International Symposium on the Theory of Switching. Harvard University Press. pp. 285–292.

- Depth-first search: dates from the 19th century, no real citation
- Uniform-cost search:
- A*: Hart, P. E.; Nilsson, N.J.; Raphael, B. (1968). "A Formal Basis for the Heuristic Determination of Minimum Cost Paths". IEEE Transactions on Systems Science and Cybernetics. 4 (2): 100–7.
- Greedy:
- Minimax:
- Alpha-beta: citation unclear; invented several times
- Expectimax: Michie, D. (1966). "Game-playing and game-learning automata". In Fox, L. (ed.). Advances in Programming and Non-Numerical Computation. Pergamon Press. pp. 183–200.

3.5 Recent advances

Single-agent and multi-agent search algorithms have long been used for various graph applications, including pathfinding, cooperation and competition. However, they have limitations which modern techniques including deep reinforcement learning can address. What are these limitations, what deep learning approaches can be used to address them, and how successful are these deep reinforcement learning approaches to meeting the challenges of single-agent and multi-agent search?

Limitations of traditional search algorithms include scalability, dynamic environments, multi-agent complexity and lack of generalisation.

One way deep learning addresses these limitations is with deep Q-networks (revolutionary, enabling agents to learn optimal policies in discrete action spaces by using deep neural networks to approximate Q-values).

Another is multi-agent reinforcement learning approaches. Techniques such as Independent Q-Learning, Deep Q-Learning from Demonstrations (DQfD), and Counterfactual Multi-Agent Policy Gradients (COMA) have been developed to tackle the challenges of learning in multi-agent environments, focusing on cooperation, competition, and communication.

And hindsight experience replay which enables agents to learn from failures, a method of generalisation that allows the agent to understand and adapt to new scenarios more effectively.

Chapter 4: Methodology

4.1 Tools and technologies

What language did you use, and why?

I decided to use Python because of my familiarity with it and the confidence I had to create a robust program using it.

What other languages could you have used, and what difference would it have made?

I could have used HTML for the visualisation, but I chose to use matplotlib because I had used it before for something similar which I hadn't done with HTML, and matplotlib was able to make visualisation appropriately simple. I could have used Java over Python for the brains of the program; this might have encouraged an object-oriented approach from the beginning, which is a good approach for something like this with a lot of reusable parts. In the end I did pursue an object-oriented approach with some of the deliverables, but the program went well working functionally with Python before that.

In the end, was your choice of programming language a good one?

I believe it was good as it enabled me to work quickly through the tasks and it was very easy for me to understand, read and debug.

What core libraries did you use to code your project in, and why?

Matplotlib because it is a very quick, adaptable and simple way to visualise the maze. Seaborn is a common data visualisation library.

What IDE did you use? What capabilities does it have that supported your work?

I used Visual Studio Code, which with third-party extensions becomes so much like an authentic IDE that there is practically little difference in capability. It is lightweight which was suitable for my needs. It had built-in Git integration which made things much simpler.

Did you use any other tools and technologies not otherwise mentioned?

I used GitHub.

4.2 Software engineering techniques

4.2.1 Version control

Why is version control important?

Version control is important for review by myself, both during product creation and after, and by my supervisor marking the product. Were this a team effort, members being on the same page is vital by being able to see the most recent version at all times, and version control enhances visibility. By tracking changes in detail, it's easy to review, refine and comment, and encourages adherence to best practice standards.

What version control method(s) did you use?

I used GitHub, the leading version control system that empowers software collaboration, tracking and storing.

How successful was your approach to version control? Did you face any challenges?

I believe my approach to version control was successful. I made commits when the code I had written to complete a task achieved what it was supposed to with no errors or bugs, was tested and commented. These commits were frequent enough to track progress, but not too frequent that one change wasn't palpably different to the previous. I wrote thoughtful commit messages to thoroughly describe the changes.

Could you have done version control a different way? Hint: there are features of Git and GitHub that you didn't use, such as branching, pull requests and issues

Probably because this was a solo effort I did not use all the features of version control that are designed for collaboration. Features of GitHub that I did not use include branching, pull requests and Issues. Branches allow you to "develop features, fix bugs, or safely experiment with new ideas in a contained area of your repository." [3] Because I was the only person working on the product and generally focussed on one feature at a time, I felt no need to branch, merge or make a pull request. Similarly, Issues are used for planning and discussing, feedback in collaboration; as a solo effort there was no imperative and it was simpler to keep improvised personal notes and plans.

4.2.2 Coding standards and practices

What aspects of code quality did you prioritise while doing your project?

While doing this project, aspects of code quality I considered were readability, reliability, reusability, efficiency and testability. Readability: I always used clear naming conventions, added comments, type hints and docstrings. Reliability: I added statements to raise exceptions. Reusability: I made the code modular. Efficiency: I refactored throughout. Testability: I began making unit tests for test-driven development. This stopped because though this approach is thorough, it is exceptionally difficult without practice and it takes considerable time.

There were trade-offs made between developing quickly, the lack of needing to collaborate with others, maintaining the code base, and being able to add unforeseen new features in the future.

What coding standards or conventions did you keep in mind while doing your project?

I tried to keep the code modular and well refactored, to minimise duplicate code. I kept to repository structure standards with the folders. I opted for functional programming as that seemed most natural to me and seemed to fit the objectives well at the beginning which carried through. I also thought about object-oriented programming which would have been particularly useful for the tiles of the maze; however the functional approach was already in place by that time and I assessed that it wasn't worth rewriting at that point. I was happy to use the object-oriented approach in the minimax deliverable for the game state as it anticipates future states of the game in order to evaluate the best option for the agent to take. I kept consistency with naming of variables, functions and classes. I used type hints to really help understand and remember what kinds of variables are being passed into and returned from functions, especially as we got into – in the case of graph – a nested dictionary of integer tuples along with associated cost integer: Dict[Tuple[int, int], Dict[Tuple[int, int], int]]. I tried to keep informative comments, as well as docstrings to summarise functions. I used some error handling to catch exceptions.

How could you have improved your coding standards and practices?

I would have liked to employ linting to flag errors I may have missed more easily and to catch stylistic faults.

4.2.3 Testing methods

How did you use unit tests in your project?

I did to begin with. I wrote some unit tests for creating and configuring the graph data structure and the visualisation. This stopped due to time constraints. The aim would have been to write thorough unit tests for all scripts, functions and modules of the product. Test-driven development also has its benefits for creating robust code, so that would have been the ultimate goal.

How did you use the debugger in your project?

I used the debugger to track values as they change and are passed into functions, especially to see how tuples were handled because they are often in a list that gets iterated through and you have to ensure they remain as a list otherwise it is the tuple itself that gets iterated through which is not good for what is essentially coordinates and a list of coordinates needed to be processed.

How did you use print statements to check your code?

I used print statements to check the state of processes and understand when certain milestones were being reached in the algorithms. I passed them as “verbose” to have the option of printing them or not.

How could you have improved your testing methods?

Thorough unit tests throughout, and test-driven development.

Chapter 5: Implementation

5.1 Graphical visualisation of the maze game

5.1.1 Overview

What was the aim of this deliverable, and how did you accomplish it?

The aim of this deliverable was to land on a maze game of my interest, and design and implement a simple graphical visualisation for it. I brainstormed some possible maze games to implement, bearing in mind personal intrigue, requirement satisfaction and suitable complexity. I settled on childhood favourite, family-created board game *Labyrinth*, which has a simple 2D design, size scalability, entertainment value, and ability to add more characters and rewards for complexity. This will help differentiate my project because it is based on a unique work, but I will also not be bound to expectation from trying to replicate a well-known work.

For the visualisation, I used matplotlib because it made visualisation appropriately simple, and the way *Labyrinth* works is each maze is composed of tiles, so each tile is a subplot on a figure; each tile has four sides which can be exits or walls, implemented as 4x4 grid on a tile where the corners are either black to indicate blocked, the centre is always white to indicate traversable, and the draw tile function takes as input the exits so the exits can be white and any side that isn't an exit can be black. The tiles will correspond to nodes on a graph, and the exits will correspond to edges. For the time being I added crown and knight images to indicate an item that can be collected and the player, respectively.

5.1.2 Design choices and technical decision-making [500 words / 20 minutes]

What decisions did you have to make, what factors did you take into consideration when making the decision, and what did you decide in the end? What alternatives were there?

I decided to use matplotlib over HTML because of my familiarity with it and how simply it could work. I had to decide how to draw each tile with its walls and exits, and how to show each tile apart from another. I decided to hold all tiles' exits in a 2D array, and chose to randomise each tile's exits, ensured one exit so a tile wasn't a separate room, and allowed the choice of dead ends in a tile having one but not two exits. The default for the maze is without dead ends to increase the flow of the maze.

5.1.3 Evaluation

I think the visualisation looks good to the user, if a little unrefined and not descriptive. It works and is simple enough – it's a search algorithm project not a visualisation project, so it's no less than what you expect. Easily adaptable to different maze configurations; you set the seed so it can be different each time, but can reproduce it from the seed. Reproducibility is good for testing, testing different behaviours on the same configuration. But of course you need to test the algorithms on different configurations. Easily adaptable to different graphs, and different sizes. Simple correspondence between the node on the graph and the tile on the subplot.

It's not entirely faithful to the original game narrative, where the maze is discovered by the player tile by tile, but these kinds of adaptations had to be made to fit the project brief and different fundamental nature of the product.

Did you make the right technical decisions in the end?

It's not the quickest. There might be libraries that are quicker.


What could you have done better?

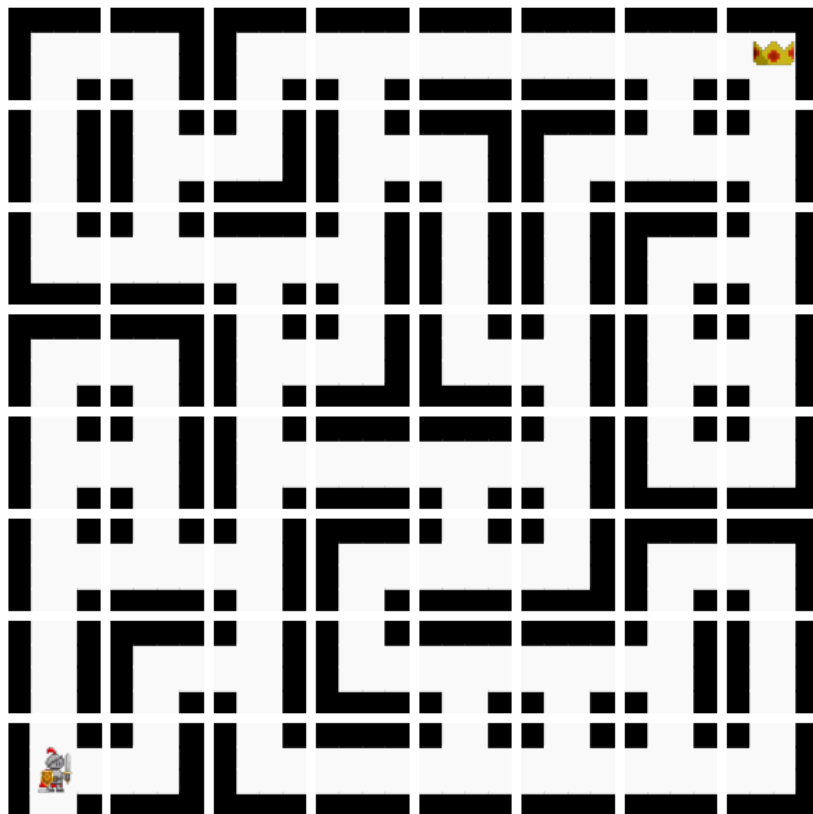
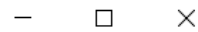
It was a short time after completion of this deliverable that I had the thought that an object-oriented approach to the maze would bring its benefits, with each tile an instance of a class that could hold information like its exits and what entities are located on it. I decided that the current implementation worked as it was and would suffice for the future development.

Text banner showing game score or what algorithm is running.

Screenshots and images as necessary

The screenshot below shows a random configuration of a medium 8x8 maze, with the hero player character as a knight in the bottom left at the 'entrance' of the maze and the crown as the goal at the opposite corner. Walls are marked as black, with white gaps as exits of each tile. Each tile's exits (north, east, south and west) are randomly decided, always with a minimum of 1 to prevent entirely cut-off tiles, and by default minimum 2 to avoid one-tile dead ends.

 Figure 1



5.2 Depth-first and breadth-first search for maze navigation

5.2.1 Overview

What was the aim of this deliverable, and how did you accomplish it?

Model the agent's navigation task as a search problem. Implement a graph search version of depth-first search and breadth-first search to reach a specific location in the maze. Run experiments with three different size maze: small, medium and large. Visualise the agent following the plan step-by-step.

I implemented a simple image of agent that moves and updates every time. Implementation of BFS and DFS was good and worked every time.

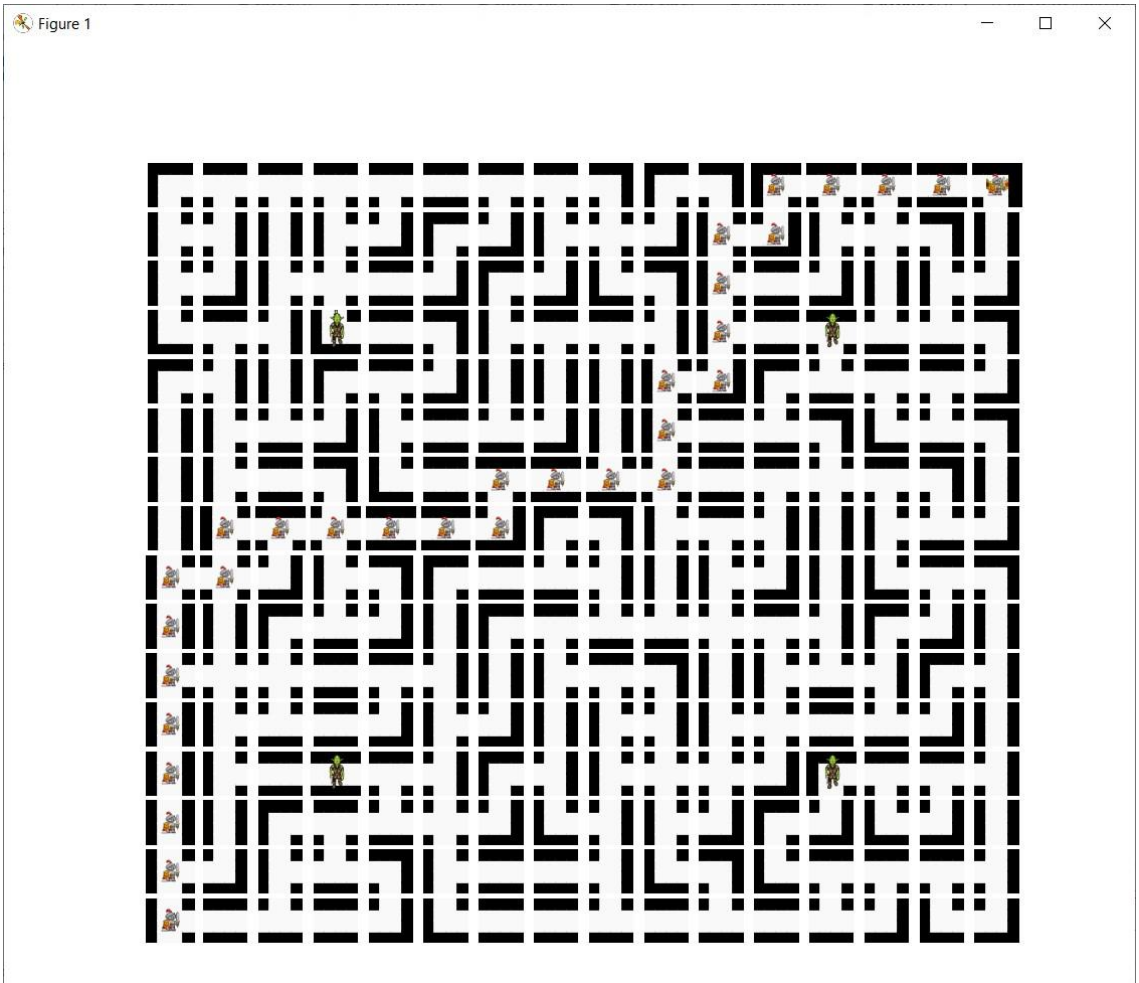
5.2.2 Design choices and evaluation

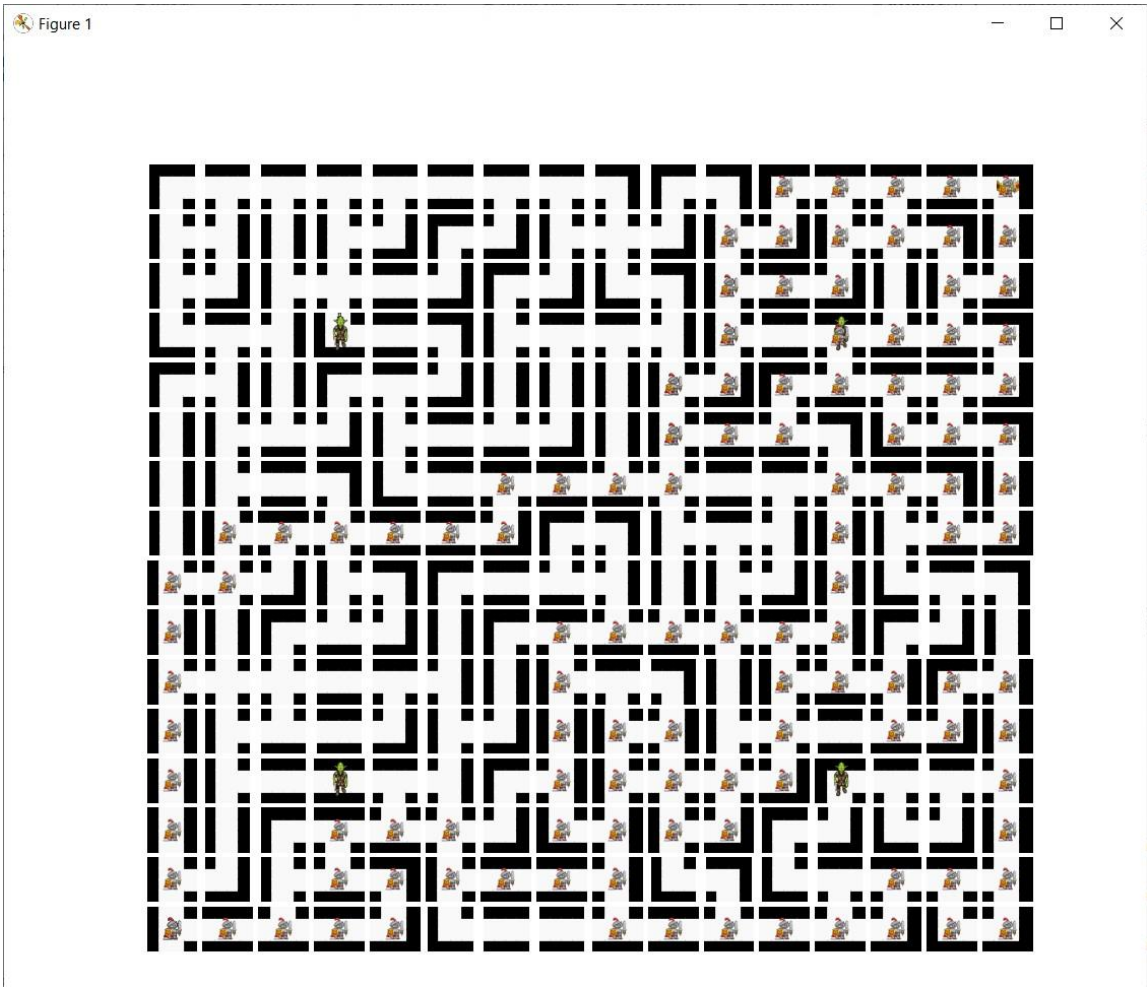
Had to erase previous path which wasn't most efficient probably.

Smoother knight moving animation. Faint version of knight while searching.

But these are optional extras, the deliverable was met.

The first image show the BFS path in full on a large maze. The second shows the same maze configuration with DFS path in full printed on top. Please be aware that the BFS path is shown as well as the DFS path in the second image. The DFS path, as the first successful path found to reach the crown, is far more winding than the BFS, which runs every possible path until it finds the shortest. DFS is speed over BFS's optimality.





5.3 Uniform-cost graph search for maze navigation

5.3.1 Overview [100 words / 5 minutes]

What was the aim of this deliverable, and how did you accomplish it? [100 words / 5 mins]

Change the cost function in such a way that traversing some areas is more expensive than traversing other areas. Implement the uniform-cost graph search algorithm to reach a specific location in the maze. Run experiments with three different size maze: small, medium and large. Visualise the agent following the plan step-by-step.

5.3.2 Design choices and evaluation

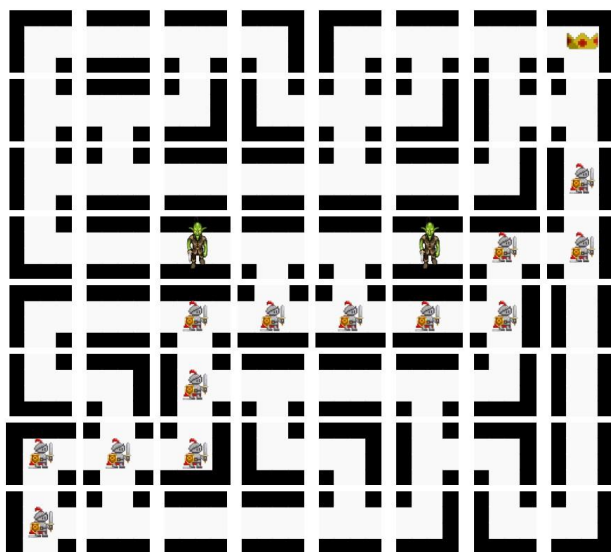
Faithful to how game works: in real game, the player is a character like the knight and the goal is to get the crown – I successfully translated that to an algorithm. There are also monsters – I successfully translated. This is important that I understand how these algorithms relate to agents in the real world, reflected in how the agents work in the framework of the game.

For UCS in particular I translated idea of monster into the graph by assigning edges leading to the monsters' tiles higher costs. Monsters were static at this point.

I took into account that these edges were bi-directional: the cost of going from A to B is not necessarily the same as going from B to A – the cost of going towards monster is higher than going away from monster.

Made choice to have uniform cost everywhere except edge going to monster. Could have had gradient, but for sake of expediency I didn't. Nothing to stop player going close to monster as long as the game isn't over without getting the reward. But since monsters are currently static, it's an appropriate representation of problem.

Before simple adjacency matrix that either had an edge or not, but very complex to introduce nest dictionary data structure that held not only identity of adjacent nodes but also weight on that adjacency. At that point should have decided to make it object-oriented way as a class and would have been easier but carried on with nested dictionary which was fine in the end.



```
PS C:\Users\Tom\Documents\GitHub\CS3821-fyp\src> python temp.py --size small
BFS shortest path from character to reward: [(7, 0), (6, 0), (6, 1), (6, 2), (5, 2), (4, 2), (4, 3), (3, 3), (3, 4), (3, 5), (3, 6), (3, 7), (2, 7), (1, 7), (0, 7)]
BFS first path from character to reward: [(7, 0), (7, 1), (7, 2), (7, 3), (7, 4), (7, 5), (6, 5), (6, 6), (7, 6), (7, 7), (6, 7), (5, 7), (4, 7), (3, 7), (3, 6), (4, 6), (5, 6), (5, 5), (5, 4), (5, 3), (5, 2), (4, 2), (4, 3), (4, 4), (3, 4), (3, 3), (3, 2), (3, 1), (3, 0), (2, 0), (1, 0), (1, 1), (2, 1), (2, 2), (2, 3), (2, 4), (2, 5), (2, 6), (1, 6), (1, 7), (0, 7)]
UCS least costly path from character to reward: [(7, 0), (6, 0), (6, 1), (6, 2), (5, 2), (4, 2), (4, 3), (4, 4), (4, 5), (4, 6), (3, 6), (3, 7), (2, 7), (1, 7), (0, 7)], Cost: 14
A* shortest path touching all corners, length: 26, cost: 25- Heuristic: Custom
```

Medium maze showing UCS path in full, with static monsters as high cost edges, with the path output in the terminal.

5.4 A* algorithm for efficient path-finding

5.4.1 Overview

What was the aim of this deliverable, and how did you accomplish it?

Implement the algorithm A* to find the shortest path through the maze that touches all four corners. Use two different, consistent heuristic functions. Run experiments with three different size maze: small, medium and large. Visualise the agent following the plan step-by-step.

5.4.2 Design choices and technical decision-making

The nearest location heuristic is effective in minimising the immediate travel cost, however it doesn't account for the subsequent steps and can lead to suboptimal paths in the bigger picture.

The sum of locations heuristic

5.4.3 Evaluation [400 words / 20 minutes]

How well did you accomplish the aim(s) of this deliverable in terms of time taken, code quality, algorithm efficiency, faithfulness to the game narrative, complexity of the solution, and user experience? [150 words / 10 minutes]

A* right function for that because it's guaranteed to be optimal path.

Did you make the right technical decisions in the end? [100 words / 5 minutes]

What could you have done better? [150 words / 5 minutes]

Add in screenshots or other images as necessary, with labels (e.g. Figure 1) and captions [5 minutes]

5.5 Comparison of A* and greedy search for single-agent path-finding

5.5.1 Overview [100 words / 5 minutes]

What was the aim of this deliverable, and how did you accomplish it? [100 words / 5 mins]

Assume that you have additional agents in your maze, one is the main player and the other are the opponents. Model this new problem and implement the player as a reflex agent that tries to visit all cells avoiding the opponents.

5.5.2 Design choices and technical decision-making [500 words / 20 minutes]

What decisions did you have to make, what factors did you take into consideration when making the decision, and what did you decide in the end? What alternatives were there? [500 words / 20 minutes]

5.5.3 Evaluation [400 words / 20 minutes]

How well did you accomplish the aim(s) of this deliverable in terms of time taken, code quality, algorithm efficiency, faithfulness to the game narrative, complexity of the solution, and user experience? [150 words / 10 minutes]

One is supposed to be more efficient – I had this hypothesis but wanted to test experimentally. Search problem became really complex once agent had to visit all tiles, so started having problems with memory efficiency. Problem with touching tile more than once – right thing for touching all corners because reduces the complexity of the search, but when objective changed so had to visit every node, because of objective changed, impossible to meet objective while keeping the constraint. So had to reformulate algorithm, and way managed to make it simple enough to not run into memory errors, was two things – one was allow revisiting of a node, but only visiting twice (revisiting once) – even with that ended with such big frontier (not problem on small but medium) so big 100,000s of nodes in path with even relatively modest size maze. So instead of heap turned into limited priority queue, a better data structure more suitable for task at hand. Because only stores 500 paths with lowest cost. Important to acknowledge that final path found not guaranteed to be optimal anymore, possible that one path kicked out was the optimal one (discarded out of queue) unlikely because only keeping paths with lowest costs, but in theory it's possible to have path with really high cost at beginning and then low so ends up most optimal. But a necessary trade-off between computational complexity and optimality in final solution.

Compare on efficiency and effectiveness – had to define for myself what they are. Begin with sad effective is length of path eventually found, but realised sometimes no path found at all, so actually effective is also rate of finding path since both things are important. In both it's important to think about how they scale with size of maze/graph – some things might be efficient on small but doesn't scale to larger – so have to test on three maze sizes. Not enough to test on one maze configuration so had to test on enough to test it's robust on several different configurations. So specified seed so each algorithm is represented with same set of 50 graphs. This means that someone else can come along and reproduce my results with same seeds – this is important scientifically.

Used new library seaborn, which makes it easy to make plots from a dataframe. Only really one line of code so really simple. Puts algorithm in different colours which is really nice and readable with minimal effort.

COMPARISON RESULTS: (View appendix) Efficiency decreases with size so algorithms don't scale as well, but greedy scales far worse than A*. Greedy also seems less effective.

It scaled quite steeply with size of graph, because huge time increase for small to medium to large. I can give my opinion which is better algorithm based on these results. Maybe not better.

Efficiency decreases with size so algorithms don't scale as well, but greedy scales far worse than A*. Greedy also seems less effective

Neither algorithm is completely effective because both of them fail to return a path for larger mazes, but A* is more effective on that measure. *And it doesn't matter much for the small maze but these are very important considerations for large mazes/graphs because the differences become dramatic*

Did you make the right technical decisions in the end? [100 words / 5 minutes]

What could you have done better? [150 words / 5 minutes]

Add in screenshots or other images as necessary, with labels (e.g. Figure 1) and captions [5 minutes]

5.6 Multi-agent search with a reflex agent

5.6.1 Overview [100 words / 5 minutes]

What was the aim of this deliverable, and how did you accomplish it? [100 words / 5 mins]

Had to introduce reflex agent – this is an agent that makes decisions based on the state of the game. Simple reflexive decisions. Greedy for player because most efficient and when near a monster the program removes the edge then recalculates. Think of the edge as a mental model, so removing the edge removes from the agent's mental model of maze. It's not very sophisticated but that's by design because the point of deliverable was to do something interesting with monsters.

The whole deliverable was a big step up in complexity because now there are multiple agents. It was the first time there is dynamism in the game. A single path in a static maze, static objective, static monster, static goal. But now two agents whose movements affect each other – dynamic.

For the player agent to avoid monsters, preplan the path and follow that unless and until it reaches monster, then avoids going directly onto tile where monster is. For monster agent, their objective goal state is the player but because player is moving on every turn they have to update the path they are following. We know A* search is not simplest for researching from scratch every time the agent moves because that would take forever, so I needed a new algorithm that could cope with change in goalstate. I went on a wild goose chase for D*Lite algorithm that I thought would be answer but that algorithm was more about an unknown maze than a changing maze, like a Mars rover not this labyrinth. I then discovered A* search with path reuse was right. It looks at where new goal state is, find place on current path that's closest and redoes A* search from there rather than where monster currently is. So reusing path that is still suitable. This works well when goal doesn't change much, which is the case as the agents only move one tile at time – it's not like agent wildly changing each time it has to recalculate. It can reuse most of path every time; the remaining search will be very short, so much more efficient.

5.6.2 Design choices and technical decision-making [500 words / 20 minutes]

What decisions did you have to make, what factors did you take into consideration when making the decision, and what did you decide in the end? What alternatives were there? [500 words / 20 minutes]

5.6.3 Evaluation [400 words / 20 minutes]

How well did you accomplish the aim(s) of this deliverable in terms of time taken, code quality, algorithm efficiency, faithfulness to the game narrative, complexity of the solution, and user experience? [150 words / 10 minutes]

Did you make the right technical decisions in the end? [100 words / 5 minutes]

What could you have done better? [150 words / 5 minutes]

Add in screenshots or other images as necessary, with labels (e.g. Figure 1) and captions [5 minutes]

5.7 Minimax algorithm for adversarial multi-agent search

5.7.1 Overview [100 words / 5 minutes]

What was the aim of this deliverable, and how did you accomplish it? [100 words / 5 mins]

Minimax is a way of making player agent's movement more interesting and complex.

However there was a problem: the monster's goal is just to catch hero but the hero's goal is to collect items while avoiding monsters, and that's not how minimax works. So I had to sacrifice the faithfulness to the game to match what minimax can accomplish.

So one evaluation function for both that is items collect * 2 – average Manhattan distance to monsters. This was an assessment of balancing both agents' different agendas. One thing I realised only at the end was that Manhattan distance doesn't take into account walls; it should have been graph distance but I wasn't able to implement the change.

5.7.2 Design choices and technical decision-making [500 words / 20 minutes]

What decisions did you have to make, what factors did you take into consideration when making the decision, and what did you decide in the end? What alternatives were there? [500 words / 20 minutes]

5.7.3 Evaluation [400 words / 20 minutes]

How well did you accomplish the aim(s) of this deliverable in terms of time taken, code quality, algorithm efficiency, faithfulness to the game narrative, complexity of the solution, and user experience? [150 words / 10 minutes]

Did you make the right technical decisions in the end? [100 words / 5 minutes]

What could you have done better? [150 words / 5 minutes]

Add in screenshots or other images as necessary, with labels (e.g. Figure 1) and captions [5 minutes]

5.8 Alpha-beta pruning for more efficient Minimax search

5.8.1 Overview [100 words / 5 minutes]

What was the aim of this deliverable, and how did you accomplish it? [100 words / 5 mins]

Minimax can be quite slow. There's a tradeoff because short depth means agents not moving in intelligent way for artificial intelligence, but if long depth, the search path expands exponentially with depth so it becomes inefficient for long depth.

Each extra depth increases it exponentially. Depth is a really key parameter to tune - short depth and agent won't be intelligent because it's not anticipating much about future, but long depth gets exponential increase in amount of searching so it gets very slow.

Alpha-beta pruning is a compromise so it can achieve with depth without exponential growth. It works out which child nodes are not going to be best and discards them. This means it doesn't increase exponentially with depth. It's not linear but somewhere between linear and exponential which is already a big improvement.

5.8.2 Design choices and technical decision-making [500 words / 20 minutes]

What decisions did you have to make, what factors did you take into consideration when making the decision, and what did you decide in the end? What alternatives were there? [500 words / 20 minutes]

5.8.3 Evaluation [400 words / 20 minutes]

How well did you accomplish the aim(s) of this deliverable in terms of time taken, code quality, algorithm efficiency, faithfulness to the game narrative, complexity of the solution, and user experience? [150 words / 10 minutes]

Did you make the right technical decisions in the end? [100 words / 5 minutes]

What could you have done better? [150 words / 5 minutes]

Add in screenshots or other images as necessary, with labels (e.g. Figure 1) and captions [5 minutes]

5.9 Expectimax algorithm for multiple agents with probabilistic behaviour

5.9.1 Overview [100 words / 5 minutes]

What was the aim of this deliverable, and how did you accomplish it? [100 words / 5 mins]

Unlike the Minimax algorithm where the outcome is determined by the choices of both players, Expectimax is designed for situations where the outcome is influenced not only by the decisions of multiple agents but also by probabilistic events.

The significant difference to Minimax is it incorporates "chance" nodes in addition to "max" (player) and "min" (monster) nodes. These chance nodes represent points in the game where a random event occurs, affecting the game's state. The outcome of these chance nodes is determined by a probabilistic distribution rather than a decision made by a player.

It's useful because it handles uncertainty, gives more realistic decision-making, is applicable to a wide range of problems, it optimises outcomes based on probabilities, and is has flexibility in evaluation.

5.9.2 Design choices and technical decision-making

What decisions did you have to make, what factors did you take into consideration when making the decision, and what did you decide in the end? What alternatives were there? [500 words / 20 minutes]

5.9.3 Evaluation

How well did you accomplish the aim(s) of this deliverable in terms of time taken, code quality, algorithm efficiency, faithfulness to the game narrative, complexity of the solution, and user experience? [150 words / 10 minutes]

Did you make the right technical decisions in the end? [100 words / 5 minutes]

What could you have done better? [150 words / 5 minutes]

Add in screenshots or other images as necessary, with labels (e.g. Figure 1) and captions [5 minutes]

Chapter 6: Discussion

6.1 Overall evaluation and challenges encountered

Overall, how well does your repo fulfil the deliverables? [50 words / 5 minutes]

Overall, I think I did very well in the time devoted. There were challenges related to not knowing or understanding the algorithms, but I researched them and put time in to understand and implement and I was satisfied with the results until the end.

As for unfinished deliverables, all can say is I ran out of time, as a consequence of working very intensely due to time pressure that meant I was burnt out in the final stretch and I had to sacrifice the deliverable. This however displays having a good handle on timing that I prioritised working on the final report, taking the most reasonable, rational and sensible action at each step given the new confines of the end stretch of the window. I had planned out which deliverables to be completed by which day, but my condition at the end made me reassess and take pragmatic action to move on. This demonstrates a kind of resilience and levelheadedness within the situation given the past couldn't be changed. There were competing priorities between improving existing deliverables, completing the later deliverables that I was working on, and writing the report that I weighed up and decided what was achievable to get the most things to the best standard possible.

What patterns do you notice in the "Evaluation" sections of each deliverable? [50 words / 10 minutes]

What were your top 2-3 most difficult challenges with this project, and how did you overcome them? Hint: consider lack of knowledge and referring to books, YouTube and existing code repos to teach yourself; time and project management and finding a mentor to structure your time; debugging and problem-solving and a mentor to help review your code; time- and memory-efficiency of the algorithms. [200 words / 10 minutes]

6.2 Future work

Given more time, how would you improve the project? [100 words / 5 minutes]

Improve code quality. Design refinements.

Given more time, how would you extend the project? [100 words / 5 minutes]

Other improvements I would make, linting formatting/ certain packages that auto format code corrects whitespace changes order of imports all this stuff automatically/ if time I would implement these packages, pep8 black vscode can do for you.

Unit tests for everything. Docstrings and type hints. Sowing aware of them. Instead of rushing in now.

Extensions: value iteration and q learning. Q learning iterctviely through trial and erorr)erorr beign caught by monster) don't explicitly learn rules, rules areppolicy, rules giving for folow path avoid mosntser is a ploicy, q leranning learns this policy for itself through tiral and eror over many games. One adv is – thin kminimax works if know what adversary is going to do- gets morecomplex if unexpected tings in environemtn incl other gents because have to hard coede policies and if don't now everythign about plicies get s hard. So if don't know q learnin is good because it learns from real data or behaviour of agents, real dynamciso f environment. Don't have to know d of e yoursled/beforehand. Scales well to large env too. . good but scale very badly to alrge graphs or mazes. Q learning and v it can be more efficient. Acquire data learn from expeirnece give ltos of games to work from. But scale better better than my algorithms, (don't search through everything?)

Anotehr extension: think about real game of labyrinth. So far one objective, real game goals are hierarchical meet this goal then this then this (key crown kil). SO implemetn hierarchical goals which would be intersting.

Another: differentiate monsters. Her ethey behave same way and have sam econsequence for encournering. Somemove fastr, don't cause game to be over, gain strgength when gain item then monster loses strenght, diff mo take diff strength.

In real game raised paltofrm where rules diff. had diea that edges of graph is diffeentn because higher costs.

Extensions that have algorihtmic consequences that more for more interetsing agents .

6.3 Self-evaluation

What did you do well in this project that someone else might not have done as well as you? [50 words / 5 minutes]

Attention to detail, self reflection. See commit messages and project diary in appendix. General code quality. Given didn't' thaven time to write consitenyl in every script and function, general tandards are good. Modular. Creativity, went beyond minimum to go beyond just basic Pac-Man and chose game that was flexible with monsters and to fit kins of algorithm to be tested within the game. Basic skills and stuff I did really well. See examples of good practiceeven I fnot through like examles of type hints and docstrings.

What did you do badly in this project that someone else might have done better than you? [50 words / 5 minutes]

What's lacking I sdidnt start on time. If I imagine workin gon project and deliverables since October imaine how good it would be because I had all the basics there.

If you could start again, what would you do differently? [100 words / 5 minutes]

Have a mentor from the beginning, with close oversight and keeping to plan. There's a great deal in knowing what it will look like because at the beginning it wasn't even fathomable because I'd not done something like this before. I would also seek more help when I was stuck on something, because I found the time lost at beating myself up over not being able to figure out part of an algorithm didn't resolve anything when speaking with a friend even explaining the problem brings a new perspective to help figure it out.

What have you learnt about AI and search algorithms through this process? [75 words / 5 minutes]

Learned the algorithms. Gained appreciation of how complex behaviours can be represented as graph search problems. And the links between abstract data structures and tangible applications.

What have you learned about software development through this process? [75 words / 5 minutes]

The collaborative process is so important. Being honest about where you are with something because hiding it leads to a bigger problem later.

What have you learned about project management through this process? [75 words / 5 minutes]

I did plan, a lot in fact. I had the plan from the project plan at the start of the year, a plan in January when I attempted to get things going, and a plan in March. See appendix for these. I was happy with these plans and will endeavour to use them again to prepare well.

What have you learned about yourself through this process? [75 words / 5 minutes]

Bibliography

- [1] Wooldridge, MultiAgent Systems
- [2] Russell Norvig, AI
- [3] <https://docs.github.com/en/pull-requests/collaborating-with-pull-requests/proposing-changes-to-your-work-with-pull-requests/about-branches>
- [4] Brown, T.E. (1996/2001), Brown Attention Deficit Disorder Scales for Adolescents and Adults, The Psychological Corporation: Harcourt Assessment Corporation, Hamden, CT
- [5] Coetzer, G. (2016), "An empirical examination of the relationship between adult attention deficit and the operational effectiveness of project managers", International Journal of Managing Projects in Business, Vol. 9 No. 3, pp. 583-605.
- [6] Biederman, J., Faraone, S.V., Spencer, T.J., Mick, E., Monuteaux, M.C. and Aleardi, M. (2006), "Functional impairments in adults with self-reports of diagnosed ADHD: a controlled study of 1001 adults in the community", Journal of Clinical Psychiatry, Vol. 67 No. 4, pp. 524-540.
- [7] Supporting Students with ADHD in Project-Based Learning. https://d-scholarship.pitt.edu/43510/1/smithDawn_etd.pdf
- [8] <https://www.tandfonline.com/doi/pdf/10.1080/23311975.2016.1271384>

Appendix

Project plan timeline, October.

Timeline

Preliminary research into field – week 2

Project plan – week 3

Design and implement graphical visualisation of maze – week 4

Model inhabitant agent as search problem – week 5

Experiment with different sizes of maze – week 6

Explore cost function differences across the maze – week 7

Compile interim report – week 10

Implement A* algorithm – week 14

Add items to the maze field – week 15

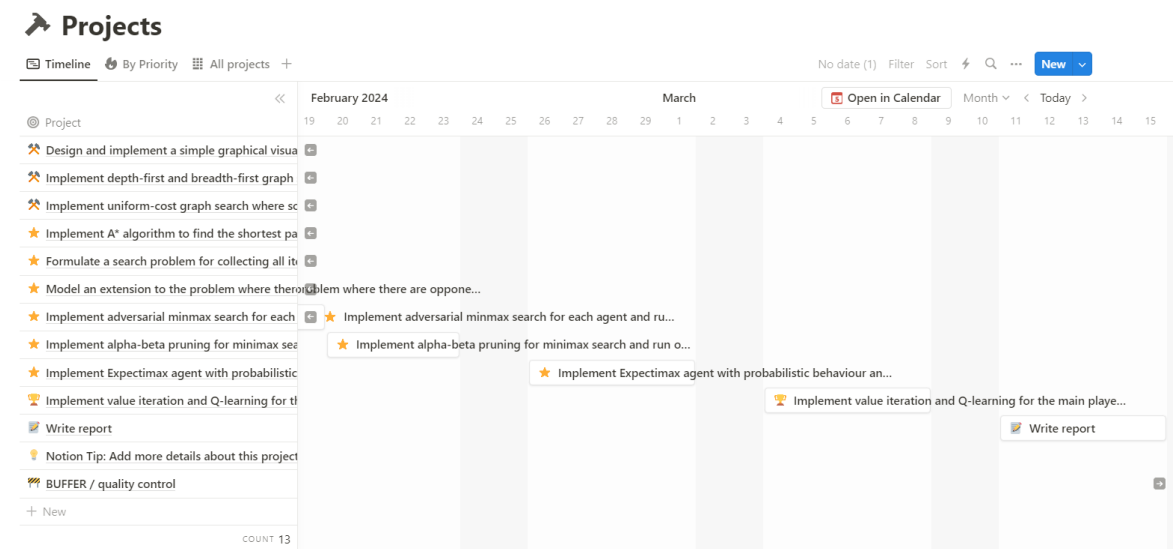
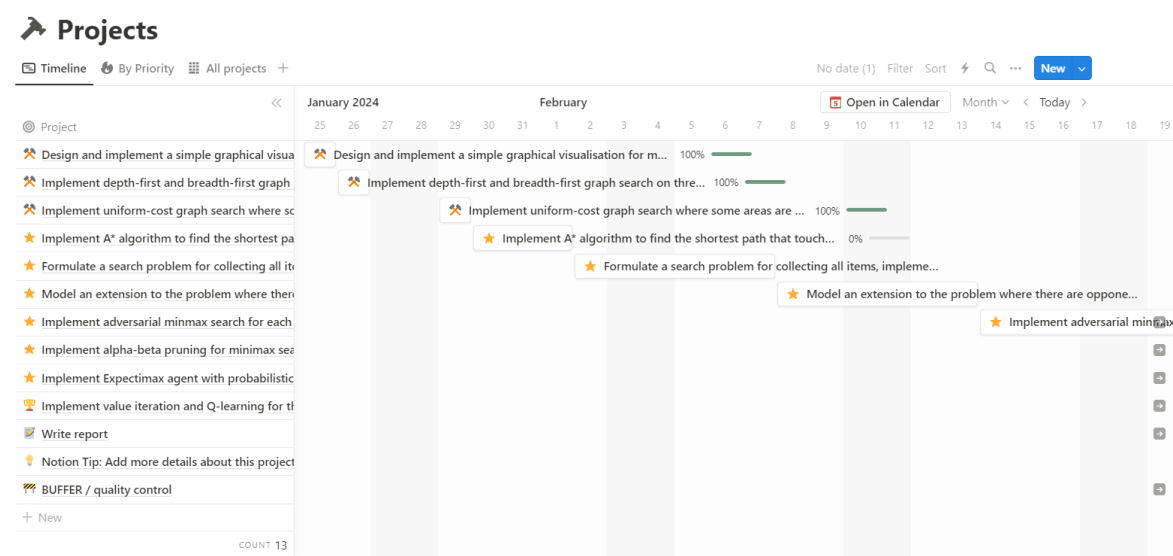
Minimax search – week 16

Alpha-beta pruning, Expectimax – week 17

Q-learning – week 19

Compile final report and demonstration – week 21

Gantt chart, January



Final Gantt chart

