Final Year Project Report

Full Unit – Interim Report

Single-Agent and Multi-Agent Search in Maze Games

Leonardo Da Silva

A report submitted in part fulfilment of the degree of

BSc (Hons) in Computer Science

Supervisor: Dr Eduard Eiben

Department of Computer Science
Royal Holloway, University of London

Declaration

Word Count: 10984

Student Name: Leonardo Da Silva

Date of Submission: 6/12/2022

Signature: Leonardo Da Silva

Table of Contents

Abstract

Mazes have always been seen throughout history and different media however one of the most significant involvements of the maze was in the 1980s, the most influential video game Pac-Man was released. This was a maze action video game where the player must achieve the highest score possible by navigating the maze eating all the dots, collecting items which are called "Power Pellets," and combatting ghosts using said items to achieve bonus points. If the player was caught without the items power up, then they would lose a life. This game quickly rose in popularity due to its inclusiveness being one of the first games that could appeal to both women and younger players and not just men [1]. Still holding the title of one of the highest-grossing and best-selling games of all time until this day [2].

In this project, I am tasked in researching and implementing general search algorithms as well as applying them to single-agent and multi-agent scenarios in maze games. This requires me to visualise and model a maze game as a search problem. I plan to use tools such as Unity engine to develop and increase the quality of the game [3].

Although Pac-Man seems simple, each of the four ghosts have their very own AI. In my maze game, I will similarly make my enemies smart however I will use minmax agents, so they can try to catch the agent faster while implementing randomisation [4]. While the Pac-Man ghosts and my agent's opponents might differ in intelligence both games are modelled as a stochastic search problem since it is not entirely predictable, and it is a multi-agent scenario [5].

The agent will be coded to efficiently path their way around the maze, reach specific locations, collect food, and combat opponents. I will be using a variety of informed and uninformed search algorithms as well as compare their heuristics. One of the basic uninformed search algorithms I will be using is the BFS which was first created by Moore to solve mazes in 1959 [4]. This search explores all the unexplored nodes near the root before exploring further away. Once explored it moves to the next depth to visit any other adjacent unexplored nodes. It keeps track of any child nodes using a queue. While this search algorithm works, it will not be as efficient as an informed search algorithms such as A*. Especially after adding a cost to the maze to traverse, areas closer to the opponents will be expensive meaning A* will avoid this path as it is looking for the cheapest one [4].

I selected this project because I am passionate about the ins and outs of video game development and believe this will be a large steppingstone for me in learning how to create unique personalities for different game genres. I aim to learn about different algorithms and hopefully come to understand them especially when applied to both single and multi-agent scenarios. I plan to produce a fully functioning action maze game, with unique agents that run using various implemented algorithms, and record experimental data demonstrating the comparisons between the algorithms. After completing the project, a final goal would be to implement a player-controlled agent difficulty of the maze game and thus making the game more entertaining.

Chapter 1: **Introduction**

1.1 Aims and Objectives

The goal is to build a game where different search problem algorithms are implemented in both single and multi-agent scenarios. These algorithms will be tested on different grid sizes and the results will be recorded to compare the efficiency. Simple algorithms such as BFS and DFS will first be implemented and tested, later evolving to more complex algorithms such as uniform-cost and greedy search, etc. I decided to take it upon myself to try using a game engine to complete this project rather than using python. This means along with the original aims of the project; I now have some additional goals such learn Unity and the language it uses C#.

This report will follow through my current progress in the project, findings in research, and explain the issues I have encountered during this project. Later down the timeline some of these ideas might change as I have a better grasp on the topic and have developed my understanding on the intricacies of how these search problem algorithms change depending on different circumstances. These vital conditions being the single or multi-agent scenarios and the introduction of items or cost to the maze.

1.2 Motivation Behind the Project

Video games have always been a core part of my life, whether it is playing, watching, and now learning to create them. While I have experience with most game genres, the maze game genre is the genre I have the least affinity for which is why I took in this project. I want to learn the ins and outs of this game genre and how the different search problem algorithms can affect the game environment. Creating a game was originally my first choice, however I decided to implement that foundation into this project. Although this project does not function as a game since there is no player input, developing an understanding in game creation with this project will be valuable for my future endeavours beyond university. As an optional extension., when the project is finished with extra time available, I will try to implement player input into the main agent of the game to fully flesh out this experience as a video game.

1.3 Unity and C#

Unity is a cross-platform game engine which has been used since 2005. In the Unity gaming report 2022, it was stated the number of games made on the unity platform increased by 93% in the last year. Taking data from over 750,000 different games and 230,000 developers [6]. I have decided to use the Unity engine for this project as I believe it will help develop important skills and deepen my knowledge of game developing rather than solely using python. Unity is seen to be engaging to developers due to its simple use and basic already implemented functionalities such as physics, collision detection and more [7]. However, using Unity comes with the extra baggage of learning a new language which is unfamiliar to me. With the increased popularity of game development, other code languages grew to be popularised in parallel. One main coding language being C#.

Like Python, C# is also a high-level, object-oriented language with a variety of uses. However, this is where the similarities end. The main reason for C# being my programming language choice lies with its compatibility and functions in unison with unity however there are still many other benefits. C# development is fast as well has better performance when compared to python. It is also a type-safe language meaning the compiler checks whether the right type has been used for a variable, if not it will not pass compilation and therefore data loss should not be an issue.

Considering my experience using other static programming languages such as java, C# would not be too large of a jump to learn.

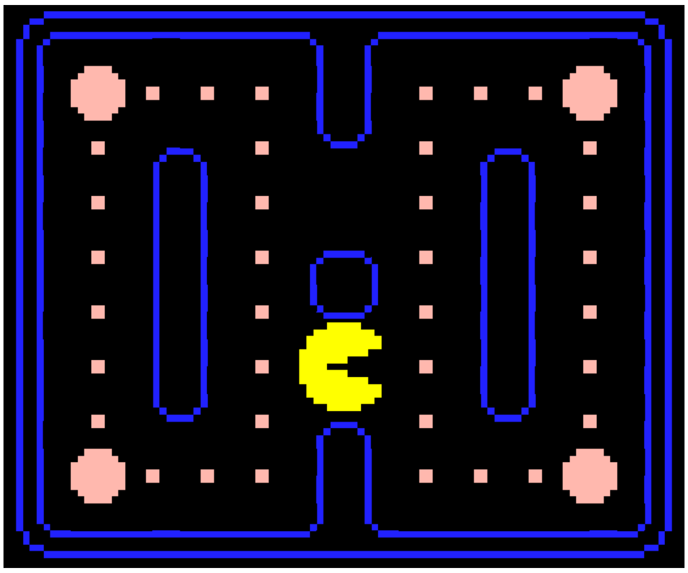
Figure 1. (old grid)

This is one of the six mazes I was able to create in Unity using the tile map tool. This tool allows me to create any custom grid I can imagine without much code needed. This maze has physical walls, nodes which will allow all agents to use the search algorithm logic and pellets which will be consumed, although that has not been implemented yet. The mazes built for the multi-agent scenario also include a small box in the middle of the grid to hold the ghosts, like the Pac-man game.

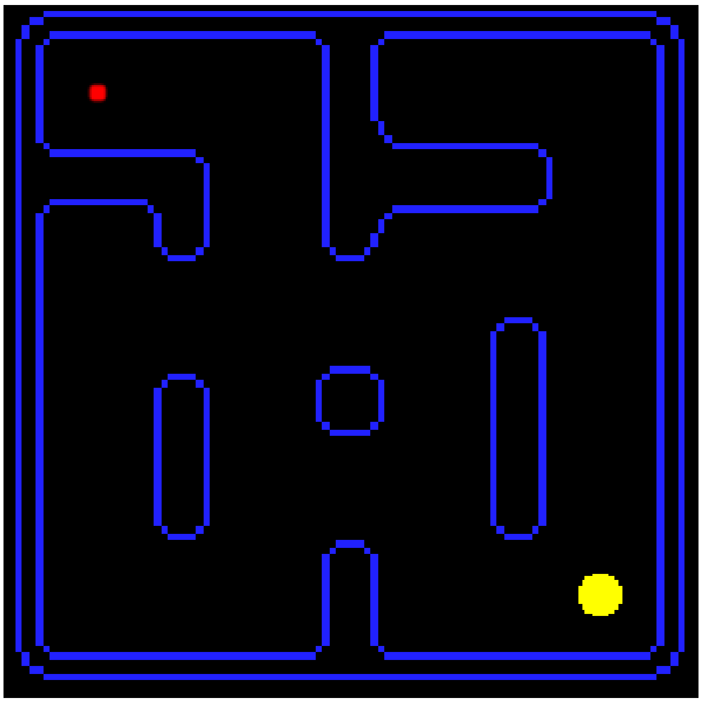
Figure 2. (new grid)

1.4 Literature Survey

There have been numerous studies that have investigated the efficiency of different general search algorithms in single and multi-agent search. Many of these studies demonstrate a strong result favouring the informed search algorithms which I expected. A* search is known as one of the best pathfinding algorithms [8]. It can be modified or changed in ways to keep up with the rapidly changing AI standards and even during these times it is being compared against newer algorithms despite being first discovered back in 1968 [9].

1.5 Milestones of Project and Planning

During term one, a lot of time was spent tinkering and playing with certain tools in Unity. This in hindsight slowed down development and lead me to fall behind on the original project plan timeline however I believe this experimentation phase will allow me to be more efficient in implementing the search algorithms in term two. Starting fresh with something new will always feel uncomfortable and unnatural so getting a grasp on these tools early is the best-case scenario. After gaining a basic understanding of Unity and C# I managed to develop 6 different mazes from which I planned for various scenarios. I did manage to code in the movement and animation script however meaning this is something I do not have to give much thought about next term. The movement is the groundwork for how all the agents will move around the maze as well as how they interact with the wall physics while the animation script will give the agents life and cycle through various frames of sprites to flesh out the game.

```
private void Next(){
  // Increment the frame
  this.animateFrame++;

  // If looping is enabled, use modulo to handle frame overflow and
resetting
  if (this.loop){
    this.animateFrame %= this.sprites.Length;
  }

 // Set the sprite renderer's sprite to the current frame of the
animation, if within the valid range
  if (this.animateFrame >= 0 && this.animateFrame < this.sprites.Length){
    this.spriteRend.sprite = this.sprites[this.animateFrame];
  }
}
```

The code above is what cycles through the sprites of the agent so that it can appear to be in motion, I had to make sure that if looping is enabled and overflowing occurs in the sprite’s length, it had to be reset back to 0 by using the modulo operator. The second part makes sure no index out of range exception occurs by setting the renderer’s sprite to the current frame of animation if it’s within range.

Current milestones achieved summarised:

- Researched tools such as Unity and the programming language C# used
- Created the subsequent mazes where the agents will explore

- Created the main agent who will explore the mazes
- Coded the basic animation and sprites scripts for all agents to use
- Added nodes to the maze which will be used for the search problem algorithms
- Implemented BFS, DFS, A* algorithms
- Created three different sized grids
- Implemented previous algorithms in multi agent scenarios

Term two allowed me to become much more familiarised with Unity and the tools involved. I used gizmo and debugging tools to solve any issues I was encountering. I developed a greater understanding of the algorithms I was implementing and how they worked in my game specifically.

Next milestones to be achieved summarised in my own time:

- Implement minimax algorithm
- Add functionality to the pellets to be eaten and increase score
- Add music

Chapter 2: **Unity and Game Development**

2.1 How it was used

The main goal of the maze game project was to create an engaging game that allows players to experience and compare the performance of different pathfinding algorithms, including breadth-first search (BFS), Depth-First Search (DFS), and A* with Manhattan heuristic. The game offers multiple grid sizes for the mazes, as well as single and multi-agent scenarios. To achieve this, the project heavily relied on Unity's features and tools, such as tilemaps for designing the mazes, matrices for grid representation, and a node creator for managing the agents' navigation.

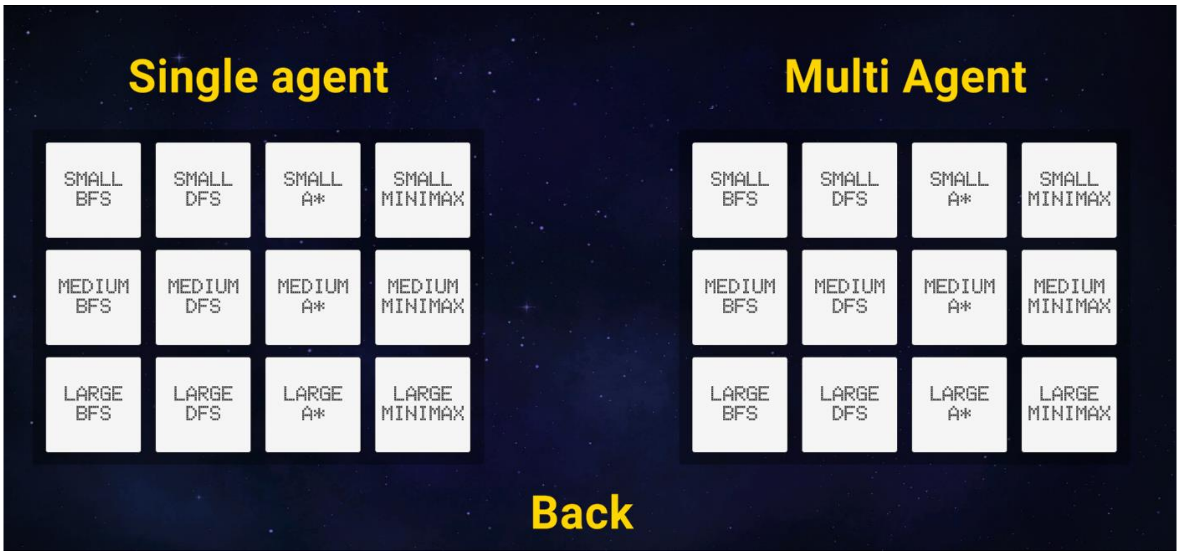

Figure 3.

A crucial aspect of the game is the level select screen as shown in Figure 3, which enables the players to choose between different maze sizes and the pathfinding algorithm they want to use. This screen utilises Unity's UI system, which offers a wide range of components, such as buttons, text, and images, to design scenes. Each button on the level select screen corresponds to a specific combination of maze size and algorithm as well as it being a single or multi agent scenario. When clicked, the button will trigger an onClickEvent that loads the appropriate scene and sets up the chosen algorithm for the agent's navigation.

Unity's tilemap system was employed to create the mazes for the game. Tilemaps are a powerful tool for designing grid-based environments, like the mazes in this project, as they allow developers to create complex levels using a palette of tiles. Each tile represents a specific element of the maze, such as walls, nodes, or empty spaces. By drawing on the tilemap grid, the developer can easily create and modify the maze's layout. The tilemap system also includes features like collision detection, which was used to prevent agents from passing through walls, and a node-based navigation system that was leveraged for pathfinding.

Three pathfinding algorithms were implemented in the game: BFS, DFS, and A* using the Manhattan heuristic. Each algorithm was developed in separate scripts, enabling players to easily switch between them on the level select screen. BFS and DFS are fundamental graph traversal algorithms, while A* is an advanced search algorithm that finds the shortest path more efficiently by incorporating a heuristic. In this project, the Manhattan heuristic was used to estimate the cost of reaching the goal from the current position, guiding the A* algorithm towards the optimal path.

The grid representation of the maze was achieved using a matrix, a data structure that stores information in rows and columns. The matrix allowed for easy navigation and manipulation of the maze's cells, which was crucial for implementing the pathfinding algorithms. Each cell in the matrix represents a tile in the tilemap, with different values indicating the type of tile (wall, node, or empty space). By iterating through the matrix, the algorithms can explore the maze's layout and find the shortest path to the target.

The node creator is a custom script that manages the creation and positioning of nodes within the maze. These nodes serve as decision points for the agents, forcing them to choose their next action based on the chosen pathfinding algorithm. The node creator generates a large square of nodes that match the grid size required.

The A* algorithm, when combined with the Manhattan heuristic, proved to be the most effective and efficient algorithm for solving the maze in the game. The Manhattan heuristic computes the absolute difference in the x and y coordinates between the current cell and the target cell, guiding the A* algorithm towards the goal more directly than BFS or DFS.

The A* algorithm was implemented by first initializing an open set, which initially contained the starting node, and a closed set, which was empty. The algorithm then iterated through the open set, examining each node, and calculating its cost, including the cost of reaching the node from the starting point (known as the 'g' cost) and the heuristic cost (the 'h' cost) to the goal. The node with the lowest total cost ('g' + 'h') was then selected, and its neighbouring nodes were added to the open set. This process continued until the goal was reached or the open set was empty, indicating that no path could be found.

The A* algorithm with the Manhattan heuristic delivered faster and more optimal results compared to the BFS and DFS algorithms, as it prioritized exploring cells that were closer to the goal, reducing the overall search time and yielding the shortest path.

The development of the maze game using Unity highlighted the powerful tools and features the game engine offers for creating engaging and interactive experiences. The level select screen, tilemaps, algorithms, matrix representation, and node creator were essential components for building the game, allowing players to explore different maze sizes and compare the performance of various pathfinding algorithms. The A* algorithm with the Manhattan heuristic emerged as the most efficient and effective solution, showcasing the potential of advanced algorithms in game development and problem-solving.

Chapter 3: **Search Algorithms**

3.1 Uninformed Search

As the name states, uninformed search is a general search algorithm that has no additional information about the search space beyond what is initially known. Resulting in the algorithm exploring the space randomly, without any prior knowledge or heuristics to guide it. When little or no information is available, uninformed search is typically used. Some examples which use this are maze games or finding the shortest path between two points. The common uninformed search algorithms I will be using are BFS, DFS and uniform-cost search. In maze games, the efficiency of uniformed search algorithm can be reduced exponentially as the size of the maze increases. This is because there will be a parallel increase in search space subsequently leading to a higher number of exploratory steps and a longer overall search time [10]. Due to the fact it is a "blind" search algorithm they also have a high cost since it requires lots of memory.

Despite the low effectiveness of uninformed search, it does have certain advantages. It is much simpler than other algorithms and can be effective in certain scenarios such as smaller mazes where the search space is small. They are also complete meaning they will provide a solution if any solution does exist excluding DFS. BFS will also provide the minimal solution which requires the least number of steps if there are more than one solution.

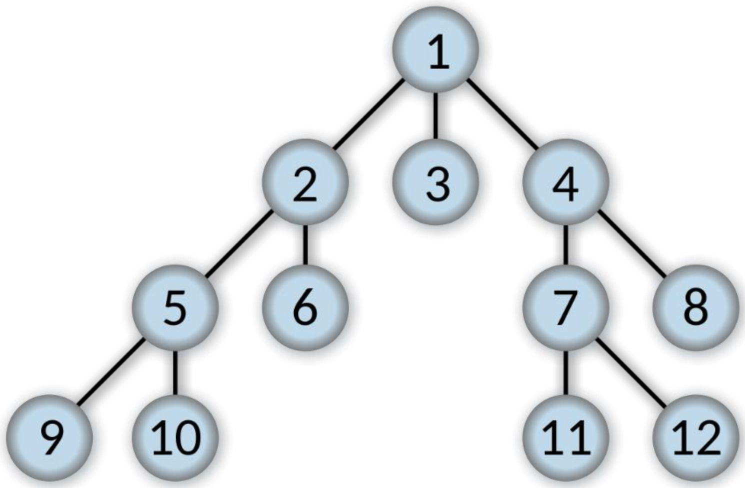

Figure 4.

Following BFS, it starts searching from the root node and expands all successors at the current level before moving down. The path travelled is 1-2-3-4-5-6-7-8-9-10-11-12. Some uninformed algorithms follow different rules such as DFS. If DFS was applied to Figure 2 the path would be 1-2-5-9-10-6-3-4-7-11-12-8. As you can see this is a recursive algorithm that uses backtracking, it starts from the root and follows the path to the full depth before moving onto the next. This allows DFS to use less memory and is much faster to reach the goal at a further distance, given the goal is on the current path traversing. Unlike normal uninformed search algorithms, some of DFS disadvantages mean that it could loop forever or never find a goal, meaning it is not optimal for finding the shortest path which is the uninformed search algorithms greatest strength.

3.2 Informed Search

In general, informed search algorithms are classified as more efficient than uninformed search algorithms since it makes use of additional information or heuristics to guide its search. This means that the algorithm can explore the search space in a more efficient and focused manner, by prioritising areas of the search space and avoiding others using their additional information. Opposite to the uninformed search, the informed search is typically used in scenarios where additional information is available. The informed search algorithms I will be focusing on are A* and greedy search. Informed search algorithm uses the idea of heuristic meaning it can also be called the heuristic search. This algorithm uses the heuristic function which finds the most optimal path. To find this path, the function takes the current state of the agent as the input to produce the estimation of the distance to the goal.

Informed searches will be useful in the multi-agent scenario due to their additional information to guide its search. Knowing the location of other opponents, it can prioritise a path that is less likely to encounter these other agents. They are also fast and cost less than the blind search algorithm, leaving them to be the more efficient algorithm. Computational requirements are also lessened when using informed search despite being able to manage larger search problems.

However, it is not perfect. In addition to DFS, the heuristic search also has the possibility of not being completed even with a solution available which lowers their efficiency. Other disadvantages include, too complex to implement and the performance of the search being influenced by the quality of the heuristic.

A* search is widely known to be one of the best algorithms for finding the shortest path between two points on a graph. This search combines the strengths of two other algorithms: both Dijkstra's algorithm and best-first search algorithm. A* makes the lowest-cost path tree from the start to the goal and uses a function that gives an estimate of the total cost of path. It uses the function: f(n) = g(n) + h(n) to expand paths. The total estimated cost of path through node n is f(n), cost so far to reach n is g(n) and the estimated cost from n to the goal is h(n). One of its key features is that it can efficiently handle obstacles and other constraints. This will be particularly useful in my project as it can efficiently navigate complex environments and avoid obstacles.

In the project, the A* algorithm was implemented using the Manhattan heuristic to provide a more efficient pathfinding solution compared to the other algorithms. The A* algorithm with the Manhattan heuristic not only improved the performance of the agent's navigation but also consistently generated the optimal path to the goal. A* is a popular and commonly used algorithm in both AI and game development, as it combines the benefits of BFS and a heuristic approach to find the shortest path between two points. The Manhattan heuristic is an effective choice for grid-based environments like my maze game, as it calculates the sum of the absolute differences between the current and target positions along the x and y axes.

```
private int ManhattanDistance(GameObject[,] matrix, int nodeIndex1, int
nodeIndex2)
    {
        int matrixWidth = matrix.GetLength(1);
        int i1 = nodeIndex1 / matrixWidth;
        int j1 = nodeIndex1 % matrixWidth;
        int i2 = nodeIndex2 / matrixWidth;
        int j2 = nodeIndex2 % matrixWidth;

        return Mathf.Abs(i1 - i2) + Mathf.Abs(j1 - j2);
    }
```

```
foreach (var neighbor in adjList)
            {
                if (visited[neighbor])
                {
                    continue;
                }

                int tentativeGScore = gScore[current] + 1;
                if (tentativeGScore < gScore[neighbor])
                {
                    gScore[neighbor] = tentativeGScore;
                                fScore[neighbor]   =   gScore[neighbor]   +
ManhattanDistance(matrix, neighbor, finalIndex);

                    if (!openList.Contains(neighbor))
                    {
                        openList.Add(neighbor);
                    }
                }
            }
```

Chapter 4: **Experimental Results**

I will measure the efficiency of algorithms through tests which I will run on three different sized mazes. I have also created mazes to account for the multi-agent scenario as they will differ. I understand in theory some algorithms should be faster however testing these algorithms in different scenarios could prove otherwise [11]. Using the score system that I will implement, I am going to record each test and observe the total amount of steps it took. The multi-agent scenario will differ slightly as there is a possibility of losing which is not present in the single-agent tests as there are no opponents. Therefore, the multi-agent tests will provide different results for efficiency as there is a factor of success to account for. To measure this, I will repeat each test at least ten times, so I can gather some results and find the percentage of success for that given agent. This will also be beneficial as it counteracts any anomalies as I have multiple results, so outliers can be managed.

For the single agent tests I believed A* [10] would take the least number of steps. In theory while BFS is great for finding the shortest path, it will consume a lot of time while DFS has a chance of not even finding a solution despite having less time and space complexity than BFS. As for the multi-agent tests, I believe minimax would have been the most efficient algorithm if it was implemented. However, out of the three algorithms I did implement, A* search was the best pathfinding algorithm for both scenarios.

4.1 Single-Agent Tests

Algorithms	Steps (small grid)	Steps (medium grid)	Steps (large grid)
BFS	52	89	142
DFS	52	89	146
A*	43	77	122

4.2 Multi-Agent Tests

Algorithms	Success % 10 tries each
BFS	87
DFS	47
A*	93

Chapter 5: **Software Engineering**

5.1 Agile Methodology

My project has been following the agile development methodology throughout the course of term one and continuing to term two. This is especially important since agile is what help keeps a project running smoothly and satisfies the client through early and continuous delivery of software. It is a flexible methodology which welcomes any change in requirements, even later in development. In this project, the developer and stakeholder relationship are simulated through regular scrum-like meetings with my given supervisor (Dr Eduard Eiben). During our meeting we discussed actions I should take as well as giving any feedback on any changes he thinks I should make. This allowed me to focus on continuous improvement before the next short deadline similarly to a sprint and in the end ensures the final product is of high quality and meets the client's needs.

5.2 Redundant/Useless Code

Ideas are constantly thrown around in game development both being removed or integrated however this can result in poor or misguided programming through code smells. These occurrences are not actual bugs, it is indication of potential breaches of code discipline, however it can cause issues down the line if useless code is not removed or leave incomprehensible code [12].

Eliminating dead code is made simple using refactoring. In fact, refactoring is one of the most effective ways of removing code smells. It is used to make code more concise, clean, and efficient without ultimately changing its functionality. Many IDEs have tools available to automate refactoring which is useful in keeping good code hygiene regularly. Refactoring is performed by breaking up code and placing the remaining blocks in different methods. In any circumstance, if code is repeated or not needed then it can be deleted otherwise this would also leave a code smell.

Game development will always require a clean codebase and constant maintenance since the interconnected systems involved can quickly diminish readability and lead to decreased maintainability. This effect can lead to confusion between development teams as they will spend unnecessary time figuring out the purpose of the observed code and whether it can be modified.

The largest problem for both gamers and developers which can stem from useless code, is the increased potential for game breaking bugs from modifying it. The game industry has many practices in place to prevent these problems from rising and this project utilises some of these practices such as version control systems, tools, and code reviews from project supervisors.

5.3 TDD Vs DDT

Test-Driven Development (TDD) and Data-Driven Testing (DDT) are two different methodologies used in the software development and testing process. Both serve different purposes and are implemented during different stages of the development lifecycle.

DDT is a testing methodology that focused on testing the functionality of an application with various existing sets of input data. This usually helps to verify that an application is behaving g as expected with different inputs and normally reduces the time and effort needed for repetitive tests. DDT has test cases, which are designed using a data source such as a database. The database will provide different input values along with expected output values in which the test scripts use this data to test the application. This methodology is mostly used for functional testing and its scope is limited to validating the application's behaviour using various sets of input data.

TDD is a software development methodology in which developers will write tests before writing any actual code. This ensures that code meets the specific requirements and allows for robust and clean codebase which is the primary purpose of this method. Firstly, a failed test will be implemented, which is then followed by the minimum amount of code needed to make the test pass. This cycle will be repeated throughout the entirety of the development process with refactoring when necessary, ensuring that all pieces of functionality is covered by tests. TDD is more flexible as it can be applied to both functional and non-functional aspects of an application. It covers several forms of testing such as unit testing, integration testing, etc.

Both TDD and DDT provide different benefits to software, yet neither are commonly used in game development. However, between both methodologies, TDD is generally more common. Primarily because TDD helps developers create a robust and maintainable codebase which is extremely important in the complex and interconnected environments of game development.

Game development involves many different systems and components, such as AI, user input, physics, and Internet protocols. In order to release a fleshed out and working game, all these different systems must be working efficiently in tangent with each other. Therefore, games usually need specific requirements in order to behave as expected which is the main purpose of TDD as discussed earlier. In addition, games usually experience various updates, new content, bug fixes and more: where refactoring and code changes are important which TDD will have covered, reducing the likelihood of new bugs or issues.

In the game development industry, the use of formal testing methodologies like TDD and DDT will vary depending on different factors. These factors can be either of the following, size of the development team, complexity of the project, and company culture. Moreover, the fast-paced and iterative nature of game development makes adhering to these methodologies quite difficult which is why many teams will use a mix of formal and informal testing techniques to suit their needs.

In my circumstances, I have no real specific requirements or professional standards to uphold however, following TDD can help me develop skills which can be used later in more strict development environments. TDD can help with sufficient testing with the algorithms which can be vital in gaining an understanding about these algorithms and how they function. Not testing the algorithms could result in my game not functioning correctly as the agent does not follow the expected path. Unfortunately, I rarely used much TDD since other forms of testing was available to me through the Unity engine.

5.4 Version Control Systems

In any form of software engineering, not just this project, version control systems (VCS) play a crucial role by helping teams of developers manage and track changes to the source code over any length of time. They provide a range of benefits that improve aspects such as collaboration, maintainability, and overall development efficiency. Although, this report was done solely by me, being able to look at the history of commits and changes to the code, I can remember what I need to do next as well as change the device I work on, so I am not limited in work environments.

The worst-case scenario for any developer is the loss of work through physical or electrical malfunctions. VCS will serve as a backup of the codebase, so that any problems will have no effect on my code and leave no permanent loss in work or functionality. Although, this project didn't have much room for branching and merging since it is solo work, it is good practice to always create branches and merge branches even when working on the project alone to develop that habit which will not be forgotten in grouped work.

Most if not every project whether professional or educational, will have a due date where the project must be completed. VCS can help streamline the development process by automating many tasks, such as merging changes, detecting conflicts between previous and more recent work and managing code history. The automation achieved from version control systems helps reduce the time that developers spend on manual tasks, leaving more time to improve and write code.

There are many popular control systems used in software engineering in this day and age including Git, Subversion (SVN), and Mercurial. While they are all different VCS, they essentially all function the same despite having different forms of operations such as committing. Version control systems is normally always considered a best practice in software development and a crucial skill to learn in your software development career.

5.5 Unity Debugging and Gizmos

Throughout my time in this project learning and using Unity, I have discovered that unity offers a range of debugging tools and gizmos that can help the developer identify and fix issues relatively easily. On top of this, it helps game developers understand the behaviour of their game or application. Similarly to many IDEs, the engine provides a console window which will display messages, warnings, and any errors generated during runtime or while working on the project. Various functions such as Debug.Log can be used to log custom messages or variables, which can help developers track the game's current state and potentially identify any issues. I used these functions several times throughout my project when the code was not working as I expected it to.

```
Debug.Log("BFS NumSteps: " + numSteps);
```

This line of code is one of the examples I used during the development of my project. I used this function to determine the underlying issue with my project. In the game's scene, it was not displaying the number of steps the main agent had to take in order to find the goal, therefore using the debug function above I figured out that the project was counting and displaying the number of steps taken however, the text object was not changing in the game scene. This problem might have never been discovered without the use of Unity's debug functions.

```
private void FindStepsInd()
    {
var comp= transform.GetChild(0).GetComponentsInChildren<RectTransform>();

        foreach(var comp in comps)
        {
            if (comp.name == "Steps")
            {
                stepsInd = comp.gameObject;
                Debug.Log("Steps GameObject found!");
            }
        }
    }
```

Looking at the code, my "Steps" object was not being updated with the correct number of steps since in the Unity's hierarchy, the object was not displayed first under canvas in the scene. The GetChild(0) line states that the first object under the parent will be the comp variable but since the first child was not "Steps", the steps object will not properly display the correct number of steps.

Unity also supports integration with popular debuggers, such as Visual Studio Code which is my main form of working on code. This integration allows developers to step through the code and inspect the different variables and objects while the game is running, making it easier to identify any problems and allow simultaneous work to be done.

Gizmos are another powerful tool provided by the Unity engine. They are visual representations in the Unity Scene view which can provide insight into the game objects, components and algorithms. They are crucial for understanding the game's structure and behaviour during development which in turn makes it easier for developers to design, debug, create the game.

I used the gizmo tools in this project to help visualise the node structure that I was trying to build in the scene using tilemaps which was a massive problem for me. The gizmos showed red lines which spanned off every node showing the possible paths which can be taken as well as whether nodes were disconnected or not. In fact, for my project specifically, I was able to visualise whether the search algorithms were functioning properly or not as the gizmos can draw lines to display the path the algorithm will take. This was done through creating a custom gizmo using the OnDrawGizmos() function. This special gizmo can represent information such as path finding nodes which was needed in my specific scenario. These various functions and tools were my main form of software engineering as I did not use formal testing forms such as TDD or DDT.

5.6 Maintenance

Maintenance refers to the process of modifying, updating, and improving software after it has been deployed to ensure it continues to function correctly as planned. This type of software engineering practice is not just limited to game development, but every software and application released. This is because maintenance is a critical phase of the software development lifecycle and often accounts for the largest portions of time spent and total cost on a project. No developers plan to release projects or applications without any support after release which is why maintenance is essential.

There are different forms of software maintenance with the most known being corrective maintenance. This form of maintenance involves the standard tasks of fixing bugs, errors or defects found after the release of the software. This is often seen in game development, also known as the "day 1 patch". With the release of a new game, users will spend hours upon playing the game and might encounter numerous bugs which can either be game breaking or benign. Developers will listen to feedback given and implement the corrective maintenance to fix all the issues.

Perfective is another form of software maintenance that with the use of user feedback, it focused on improving the software's existing functionality or performance. Involving adding new features, optimising algorithms for better performance or refactoring code for readability. Considering my algorithms could be improved, this form of maintenance can be applied to my project and hopefully allow for a more developed algorithm to be developed through continuous maintenance.

During and after deployment, software maintenance can be effectively managed using various practices. Using collaboration tools such as Trello, can help a team of developer prioritise and plan, and allow allocation of resources to any critical issue on the list of tasks of higher priority. Code standards can prove to be beneficial in the maintenance of a project as it allows projects to be easier to approach, identify and fix issues as well as add new functionality or performing other tasks.

Chapter 6: **Professional Issues**

6.1 Artificial Intelligence and Their Issues

Various professional issues and concerns have been raised with the evolving capabilities of artificial intelligence (AI), especially relating to my project the context of replacing human labour seems imminent. AI has exponentially grown throughout the recent years and holds the potential to revolutionise various industries [13], automate tasks, and drive innovation, it also presents future challenges that must be carefully addressed [14]. My project has allowed me to learn about different algorithms and how their efficiency is compared with one another however, it is easy to forget the real-life implications these projects can lead to in the coming years with one of the most discussed issues being job displacement and unemployment that will rise with the implementation of AI which already has started taking effect in different industries.

With no evidence of slowing down, AI has become so advanced that they can now automate a wide range of tasks, such as transportation [15], customer service, manufacturing, etc. This can potentially lead to job displacement and unemployment for those in affected roles. Prototypes for automated vehicles has been discussed and shown with the increased popularity of car brands like Tesla with its self-driving and parking features. The increasing popularity of AI agents can prove to influence game design and development. These agents will continuously become more advanced and capable of solving more complex mazes or tasks, increasing the reliance on AI agents while also reducing the need for human designers and testers, since AI will eventually generate and test new content more efficiently.

These issues will not just be restricted to development in game industry but also to clients who enjoy these games. Competition with human players will be undermined with the use of AI agents in maze games or other competitive environments as it can show a superior performance which inevitably would impact the role of human players [16]. The outperformance of AI agents against human players could potentially diminish the appeal of selected game genres altering the future of human competition in gaming.

The rise of AI may necessitate the development of new skills to make up for the skill gap, as workers need to adapt to changing job requirements or transition into new roles all-together with the possibility of job displacement. AI is growing at an alarming rate which leaves the struggling workers trying to acquire the necessary skills or education to remain competitive in the job market against AI agents and their vast skill pool.

Future implementations of AI agents in automated vehicles also pose the dangers of accountability and liability. It can be a tough decision to determine who is responsible for AI systems' actions or decisions, especially in the scenarios where these systems have been left with full faith replacing human decision-making. It is essential to understand who would be responsible in these situations to establish trust in these new AI systems when they are eventually integrated and be left with power to make sure organisations or individuals are held responsible.

These professional issues may not be completely unique to maze games or my project however, they illustrate the larger concerns and problems of AI replacing human roles in different environments and work places. Fully understanding and accepting these issues along with implanting appropriate regulations and policies can help create standards and promote responsible AI use both in work and life.

6.2 Licensing and Plagiarism

During the development of my project, I had access to infinite resources and knowledge. I watched videos, read articles and sites regarding different algorithms in game development, both with and without Unity and in other coding languages. This professional issue arose during my time spent watching these resources using services which need to be paid for or claiming work as your own has large repercussions and rules are in place due to its ethical and practical importance both in and outside education. Addressing the issues involving both licensing and plagiarism is crucial for maintaining professional integrity, protecting intellectual property rights, and fostering innovation and quality work.

In Unity, packages and other resources can be used for your own projects, some packages have licensing rules placed where it cannot be used in the scenario of making money from this project using their resources, different names such as copyright can encompass this area. Having respect for intellectual property involves understanding licensing and proper attribution of sources for the work of others as well as acknowledging their contributions in your work. This respect is an essential part of professional ethics and promotes a sense of fairness between two parties and cooperation in the professional community. This was important to me as I was planning on using a package called the A* project package which consists of scripts and code to help program enemy AI to follow a target of your choosing using the A* search algorithm which I thought would be useful in my project. In the end, I decided not to use this project due to the basic functionality of my game which might be encompassed by this enemy AI making it too difficult for my algorithms to run. Therefore, I used random movements instead for the enemy, although I think this resulted in a completely different problem than I imagined. If I did implement this package however, I would make sure to properly cite the creators work and explicitly explain that is it a package made by another developer which I decided to use to help with my game.

Not acknowledging other people’s work or claiming them as your own can tarnish the academic and professional integrity standards which is another vital ethical importance. Upholding these ethical standards by both adhering to licensing terms and avoiding plagiarism is vital. Keeping the integrity in these standards can help build trust and credibility amongst peers, employers, and clients. These standards are not solely seen in education alone, as they are important in professional roles such as work or even through personal endeavours. Licensing and addressing other people’s work also ensures a level playing field for all professionals by preventing any unfair advantages that may arise from using work other than your own without proper permission or citing.

Licensing and plagiarism do not only hold ethical importance but practical importance too. Not following either of these standards can potentially leave you in legal disputes and financial penalties resulting in a large sum of money. Complying with licensing terms can protect both the user and the creator of intellectual property, while addressing plagiarism can help to prevent copyright infringement and the legal ramifications that come with it.

Awareness about these issues can be spread using various means such as providing education and training on these topics, establishing clear policies and guidelines on how to address licensing compliance and plagiarism prevention as well as try to cultivate a culture of integrity and respect for the work of others.

Chapter 7: **Project Reflection**

7.1 The Final Product and Changes Made

The final product is a small simulation game where the user can select different AI modes and watch the different algorithms complete the maze and the number of steps it took to complete the maze. I managed to implement three different algorithms consisting of BFS, DFS, and A* in three different grid sizes as well as their multi-agent counterpart. I planned to also implement player inputs and the minmax algorithm however I was not able to. I believe minmax would have provided the best results in the multi-agent scenario as it can account for the different agents while traversing. I am happy with the final product as it symbolises the work I put in and the journey I took in making it happen despite all the issues I encountered. This project has taught me many valuable skills as well as tools that I have developed a familiarity with and hopefully continue using throughout the years whether in my career or personal.

I originally had many different grids than what I currently have, and they were reminiscent of the original pacman grid but in different sizes along with the tunnel scrapped. My mazes now have a more traditional look to them rather than following the pacman grid standard, although visually I still used pacman and grid colours. To select the goal the algorithm would find, I originally wanted to implement a point and click function where you can select the node at which the goal would be with the path being displayed before actually moving however, I scrapped that idea and used a button to pathfind to the designated goal I hard coded in. Testing would also be done through timing originally, where I would calculate the average time taken to find a path using each algorithm however the use of steps would be more interesting to me as that would display the algorithms' efficiency.

The initial renditions of my mazes also used tilemaps and nodes were implemented using those tilemaps with prefabs however this caused many problems which led me to create node game objects instead which will spread throughout the grid. However, there was one problem with this new direction, the first maze had a grid size of 12x12 which meant a total of 144 nodes and the largest maze was 20x20 with 400 total nodes. Manually placing each node on the grid tilemap I made would be exhausting and counterproductive, therefore I made a NodeCreator that would allow me to use unity tools and efficiently build a node network of any size desired automatically.

```
public class NodeCreatorWindow : EditorWindow
{
    [MenuItem("Window/Node Creator")]
    public static void ShowWindow()
    {
        GetWindow<NodeCreatorWindow>("Node Creator");
    }

    private int numRows;
    private int numCols;
    private GameObject nodePrefab;
    private Transform parentNode;

    private void OnGUI()
    {
        GUILayout.Label("Node Creator Settings", EditorStyles.boldLabel);

        numRows = EditorGUILayout.IntField("Number of Rows", numRows);
        numCols = EditorGUILayout.IntField("Number of Columns", numCols);
            nodePrefab   =   (GameObject)EditorGUILayout.ObjectField("Node
Prefab", nodePrefab, typeof(GameObject), false);
```

```
            parentNode = (Transform)EditorGUILayout.ObjectField("Parent
Node", parentNode, typeof(Transform), true);

        if (GUILayout.Button("Create Nodes"))
        {
            CreateNodes();
        }
    }

    private void CreateNodes()
    {
        if (nodePrefab == null || parentNode == null)
        {
            Debug.LogError("Node Prefab and Parent Node cannot be
null.");
            return;
        }

        for (int i = 0; i < numRows; i++)
        {
            for (int j = 0; j < numCols; j++)
            {
                GameObject newNode = Instantiate(nodePrefab, new
Vector3(j, -i, 0), Quaternion.identity);
                newNode.transform.SetParent(parentNode);
            }
        }
    }
}
```

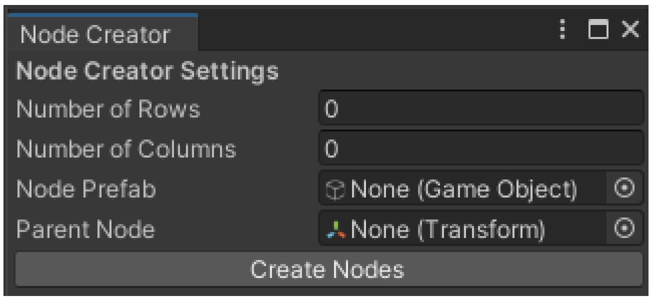
Node Creator
Node Creator Settings
Number of Rows: 0
Number of Columns: 0
Node Prefab: None (Game Object)
Parent Node: None (Transform)

Figure 5.

Figure 2 shows the GUI of the script that would allow me speed up the process of node network creation. This can be found on the unity menu bar under Window > Node Creator.

7.2 Issues and Mistakes

Various problems were encountered during the project's development and each time a new issue was found it was as hard as the last. Some of these problems relate to the code and project however some problems were real life physical issues which caused my project development to slow down. Most of the physical real life problems were just restrictions to my workspace with builders working in my room replacing the windows but despite that most of the other problems were project related. The biggest problem being the tilemaps for nodes, which virtually made it impossible to implement any sort of algorithm as none of the nodes would connect with each other. Using the gizmo tool, I was able to visualise the issue and saw that certain connections weren't being made with each other as well only a singular node being recognised by the agent to use traversal.

Eventually, I used a grid instead of the tilemap for nodes, and it immediately worked with rarely any issues regarding the nodes themselves. I would occasionally experience an out of bound error from my agent being outside the grid as I would forget to change the starting position for the agent in different sized grids. Most of the errors I encountered were due objects being outside the grid resulting in out of bound errors and objects not being placed correctly in the hierarchy. This was seen with the steps' indicator as the game was not displaying the steps properly until I moved the steps object into the correct position. Some issues are still left in the project which have not been resolved yet that I have identified.

I have two main problems in my project which need to be addressed, the first being my enemy object (spirit) seems to double in speed every time the start button is pressed down. This problem was unseen until later during the result recordings where I had to take a sample size of five tries for each algorithm and grid, realising the spirit was taking more movement inputs each time the button was pressed. Although I thought I solved the steps' indicator, it appears it is still not recording the steps of the agent taken. In numerous simulations, especially with DFS, the agent would visibly take a much longer path than either BFS or A* however the step indicator would suggest an equal path to BFS and only a little slower than A*. A* proved to be fastest with the least number of steps taken in each grid, it also had the issue of taking the same path to BFS visually yet took fewer steps.

7.3 Next Steps

Although the project is finished, there are still some features and improvements that can be added to enhance the game further. The project is still missing core functionality such as a player controlled mode in its own unique grid with multiple enemies. This can provide the user with a fun alternative game which can be played and reach a high score on, implementing this feature would involve creating some new scripts for player input inside pacwoman which will detect keyboard inputs and apply them to the agent's movement. The minimax algorithm was also missing from the game I developed, this algorithm would use decision-making in a multi-agent scenario. This algorithm can help improve the AI for the enemy agents, making them more challenging and strategic or can be applied to the main agent to try to evade all the enemy agents. I would still have to research the minimax algorithm and understand its implementation before trying to adapt it into my game.

The game was also lacking in personality, sound implementation with sound effects, background music can really enhance the game experience for users. Some examples of sound could be used for agent movement, goal reached and agent interactions. I would need to use Unity's built-in audio system to trigger and manage the sounds. In addition to this, despite adding the UI visual for volume control, it did not alter sound despite having no sound in the first place. I used brackeys tutorials for main menu scenes in Unity and thought it would be fun to implement volume control.

Polishing and optimising the game would be a good next step. Many issues are still rampant as discussed earlier which need fixing, reviewing my game's performance and reducing unnecessary object instantiation might help solve some of the other ongoing issues or with any video game release, testing and bug fixing might do the trick in solving the issues missed earlier. I would have to conduct thorough testing of the game to identify and fix these underlying bugs or issues, involving play tests of different scenarios and addressing any performance bottlenecks.

The final step would be to release the game to the public, so it can be played and enjoyed as all games are meant to be.

Diary Log

This will be my project diary.

Every week I will write in this diary twice where I will elaborate what I had to do, what I am going to do next and anything I have struggled with.

12/10/2022
6th October 2022: Finished and uploaded the project plan. For this week I will do some research on how to create a maze game as well as how to use gitlab.

15/10/2022
Started my project report which will be filled out during the project's duration. Used the template for overall layout and headings.

23/10/2022
Cloned the gitlab repo to vscode so I can work in an IDE I am familiar with.
I initially struggled with the cloning however after doing some research was able to follow through. Also researched unity and how to work with it along the repository. I was able to commit the unity project to the repository.
Next week I will try to develop a rough design for the agents and maze using unity and other tools such as blender.

2/11/2022
I researched a fair bit on how to write up and use different search algorithms such as bfs and dfs using python.
I'm currently conflicted because I want to use python as I'm more comfortable with this language however unity uses C#.
This has made it difficult to start coding on my project. I'm not sure if I should bin the unity idea or try learning C#.
Hopefully next time I'd have started on the project and decided on a language to use.
Most of the research had shown all the algorithms in python.

8/11/2022
Worked on interim report and also had to change the unity version of the project to support 2D packages
on unity. Working on developing the game first and then hopefully the search agents of pacman after. I decided to stay working with C# and unity
despite it being more foreign to me. I will undoubtedly encounter more bugs though in the future.

9/11/2022
Created my first maze which is similar to the original pacman game. This will be the largest maze. It was a little time consuming but should make coding easier in the long run with unity.
I need to learn how to make multiple mazes in the game using unity. Along with finding out a system to pick out different maze sizes along with a search algorithm.
Later down the line I will develop some UI where you can select single/multiple agents, different map sizes, and specific search algorithm.

10/11/2022
I added nodes to the maze. These nodes will be used by the ghosts so that at each node they are forced to decide their next action rather than continuously follow the same behaviour.

I have also created some classes/ scripts where I will code the various objects in the maze game. The nodes were relatively simple like the pellets and creating the maze, but I think I'm going to change some sprites
so, they do not resemble pacman as much. This will also be my first-time coding game objects in unity so I will need to learn fast.

11/11/2022
Created multiple scenes in the game which will be the different maze sizes. Along with an alternative identical maze but made for multiple agents. Going to research on youtube how to make game main menus on unity and hopefully
develop a working main menu where you can select maze size, single or multi agent game, and which search algorithm to use. It's been hard to fully write my own C# code from scratch since I'm still relatively
new to unity and writing in C#, So I've been modifying code used in youtube videos which I will reference in my project. The youtube channel Brackeys has been a really good source of information when learning unity.

12/11/2022
Designed all the mazes for the different sizes and whether they hold one or multiple agents. Was simple however a little time consuming. I next plan to start scripting the different properties of the game such as pellets, agents, etc.
Then hopefully start on the BFS/DFS searching algorithm for the agent without opponents. I will script in a score for the game and try some experiments using the different algorithms.

17/11/2022
Worked on my report. Need to finish Main Menu/ UI to select the different grid sizes. Struggled a little with time management recently.

27/11/2022
Getting my report ready for a draft to be given to my supervisor to review. Will then input the feedback after. I am behind on certain goals but hopefully I will be able to catch up.

01/12/2022
Been working on report recently to have it completed for the due date however was able to take an extension. With the extra time I will try to implement at least BFS to the agent so I can record a video of the agent navigating the maze.
Time management has been my biggest issue so far and will try to bypass this next term using Trello and other apps. Hopefully I'll be able to use the BFS algorithm to solve the search problem without many issues.

03/12/2022
Added wall collision, Pacwoman collision as well as scripts for both movement and animation. Going to try to implement BFS at least for the report submission.

09/02/2023
Started touching up on the report after receiving feedback from supervisor.

16/03/2023
Had a lot of issues with authentication and disruptions at home, managed to make a main menu scene where you will be able to select what algorithm to run or if you want to play yourself.

28/03/2023

Forgot to write in diary about the new features added from the past week. I have implemented a level select where the user can pick between single or multi agent levels as well as which algorithm will be selected. I tried implementing auto generated grids instead of readymade however I have had issues on this for multiple weeks.
I have also had many problems implementing BFS into this project for some reason however hopefully when that is implemented DFS should be easy since it is just using a stack instead. A* should not be much of an issue either however I am really struggling with implementing these algorithms. I have added BFS however having problems with the path being generated. Target nodes are being found but I am always being left with path length 0 therefore using Unity's Debugging to figure out the issues.

29/03/2023
Tried numerous methods of trying to implement BFS including various ways of movement however using a goal object will be the easiest method despite wanting to use mouse click for goal as an option. I will probably have to leave out some features out too.
A lot of the issues I am currently having is with my original implementation of nodes using tilemaps however the tilemaps only provide a single node for some reason leaving no path for BFS therefore I will have to scrap the node tilemap idea.
The nodes will have to be GameObjects themselves.
Tried various methods of implementing BFS however still unsuccessful so will try implement A* instead using videos from brackeys, etc. Hopefully after implementing A* the other two will be easier.
I realised what the potential issue might have been so planning on trying it one last time despite being short on time. I believe the starting position I coded for the agent was out of bounds from the matrix since I forgot to put it within the matrix size.

30/03/2023
Finally managed to get BFS working with rotations of sprite functional. Planning on transferring to the different sized mazes now. I plan to use the A* project package which can help with enemy AI.
I will credit the people who made that package as it is commonly used in games. DFS now needs to be implemented which hopefully is not too difficult and A* too.

Managed to implement both BFS and DFS to all sizes of mazes. Now need to implement A*, minmax might not be possible due to time constraints.
Hopefully will still be able to add multi-agent scenarios, even if it just one enemy. Hopefully the enemy can use the A* project package, if not I will make them random.

Also fixed the steps not displaying properly however it doesn’t seem to be counting properly. A* has been added and seems to work fine however due to the size of the map it seems to work the same as BFS, however.
the steps indicator shows less despite showing the same path.

Was not able to include minimax algorithm however managed to implement multi agents although it is quite basic. If I was able to work on it more, I'd implement A* search to the enemy, I said I was going to use a package but made random movement instead.
Fixed an issue with the spirit causing problems in none multi scenes.

Going to finish report now to submit everything.

31/03/2023
Worked on report until the submission date.

Bibliography

[1] A. L. McDivitt, *Gender and the Early Video Game Industry in the United States (1950s–1980s)*. Berlin, Boston: De Gruyter Oldenbourg, 2020. doi: doi:10.1515/9783110668575.

[2] "Pac-Man - Wikipedia." https://en.wikipedia.org/wiki/Pac-Man (accessed Dec. 02, 2022).

[3] A. Majumder, "Pathfinding and Navigation," in *Deep Reinforcement Learning in Unity: With Unity ML Toolkit*, Berkeley, CA: Apress, 2021, pp. 73–153. doi: 10.1007/978-1-4842-6503-1_2.

[4] S. Russel and P. Norvig, *Artificial Intelligence: A Modern Approach*, Third Edition. 2010.

[5] M. Gallagher and A. Ryan, "Learning to play Pac-Man: an evolutionary, rule-based approach," in *The 2003 Congress on Evolutionary Computation, 2003. CEC '03.*, 2003, pp. 2462-2469 Vol.4. doi: 10.1109/CEC.2003.1299397.

[6] "Unity-gaming-report2022."

[7] B. Nicoll Benjamin and Keogh, "The Unity Game Engine and the Circuits of Cultural Software," in *The Unity Game Engine and the Circuits of Cultural Software*, Cham: Springer International Publishing, 2019, pp. 1–21. doi: 10.1007/978-3-030-25012-6_1.

[8] D. Foead, A. Ghifari, M. B. Kusuma, N. Hanafiah, and E. Gunawan, "A Systematic Literature Review of A* Pathfinding," *Procedia Comput Sci*, vol. 179, pp. 507–514, 2021, doi: https://doi.org/10.1016/j.procs.2021.01.034.

[9] G. E. Mathew, "Direction Based Heuristic for Pathfinding in Video Games," *Procedia Comput Sci*, vol. 47, pp. 262–271, 2015, doi: https://doi.org/10.1016/j.procs.2015.03.206.

[10] M. J. Pathak, R. L. Patel, and S. P. Rami, "Comparative analysis of search Algorithms," *Int J Comput Appl*, vol. 179, no. 50, pp. 40–43, 2018.

[11] A. Noori and F. Moradi, "Simulation and Comparison of Efficency in Pathfinding algorithms in Games," *Ciência e Natura*, vol. 37, pp. 230–238, Dec. 2015, doi: 10.5902/2179460X20778.

[12] S. Romano, C. Vendome, G. Scanniello, and D. Poshyvanyk, "A Multi-Study Investigation into Dead Code," *IEEE Transactions on Software Engineering*, vol. 46, no. 1, pp. 71–99, 2020, doi: 10.1109/TSE.2018.2842781.

[13] M. Song, X. Xing, Y. Duan, J. Cohen, and J. Mou, "Will artificial intelligence replace human customer service? The impact of communication quality and privacy risks on adoption intention," *Journal of Retailing and Consumer Services*, vol. 66, p. 102900, 2022, doi: https://doi.org/10.1016/j.jretconser.2021.102900.

[14] G. H. Kasap, "Can Artificial Intelligence ('AI') Replace Human Arbitrators? Technological Concerns and Legal Implications," *Journal of Dispute Resolution*, vol. 2021, no. 2, pp. 209–254, 2021, [Online]. Available: https://heinonline.org/HOL/P?h=hein.journals/jdisres2021&i=226

[15] R. Chai, A. Tsourdos, A. Savvaris, S. Chai, Y. Xia, and C. L. Philip Chen, "Review of advanced guidance and control algorithms for space/aerospace vehicles," *Progress in Aerospace Sciences*, vol. 122, p. 100696, 2021, doi: https://doi.org/10.1016/j.paerosci.2021.100696.

[16] S. Krakowski, J. Luger, and S. Raisch, “Artificial intelligence and the changing sources of competitive advantage,” *Strategic Management Journal*, vol. n/a, no. n/a, doi: https://doi.org/10.1002/smj.3387.

Acronyms

DFS- Depth-First Search

BFS- Breadth-First Search

UCS- Uniform Cost Search

TDD- Test Driven Development

AI- Artificial intelligence