Criterion A:

The scenario:

The client for my product is a friend of mine called Oliver Lindfors, he is a young adult who in his free time likes to participate in races that consist of problem-solving exercises and require the participants to get through the course as quickly as possible. In the upcoming competition, my client has identified that there will be a maze/labyrinth that he will need to navigate through quickly if he intends on winning. The labyrinth is already set, the quickest way to the centre can be the difference between first and second place. I realised this was a good opportunity for me to provide a solution, therefore, I offered to build a product to help him navigate the maze in the fastest way possible (original conversation in appendix).

Rationale for the product:

The product I create for my client will be a visualisation tool for the A* search algorithm. The aim is to accurately and efficiently calculate the fastest way route through the labyrinth. In order to visualise the algorithm I will be using the game development software Unity3D. In Unity I would have to use C#, which I have little experience programming, however, I determined that the ease that Unity provided in terms of visualisation would be preferable compared to the alternative of writing a possibly excessive amount of code in CSS.

Specific performance (success) criteria:

Success criteria	Justification
The user should be able to customise the grid (plane)	This is in order for the client to alter the program if there was a change in size (area) of the labyrinth.
The user should be able to alter the layout of the walls in the labyrinth	The client can use the product to alter the design of the maze repeatedly meaning the client can use it for different races.
The user should be able to move the location of both the starting position as well as the target position.	-
The product should generate the shortest path between the start position and end position taking into account the obstacles (walls) that obstruct the path.	The shortest path through the maze can be assumed to take the shortest amount of time therefore, the client can win the races.

Words(374)

Criterion B:

1. Flowchart diagrams of the program:

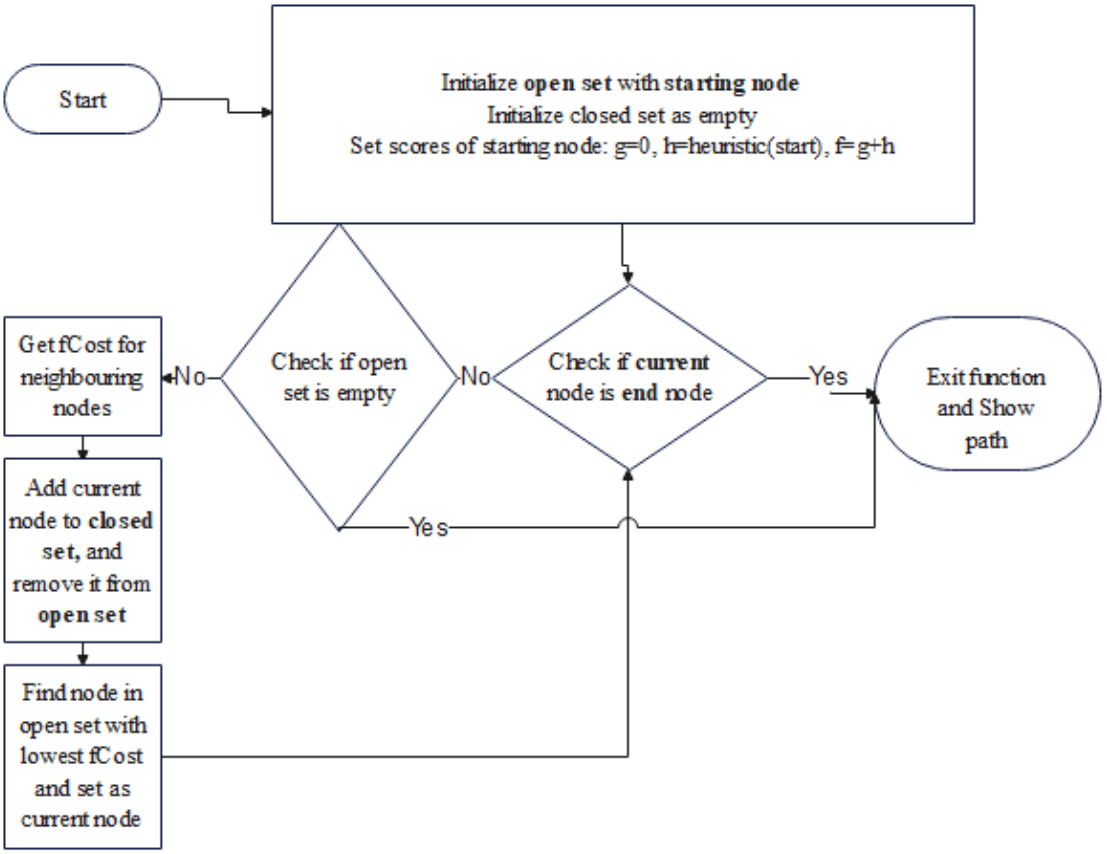


Figure 1: Relatively basic flowchart of the program

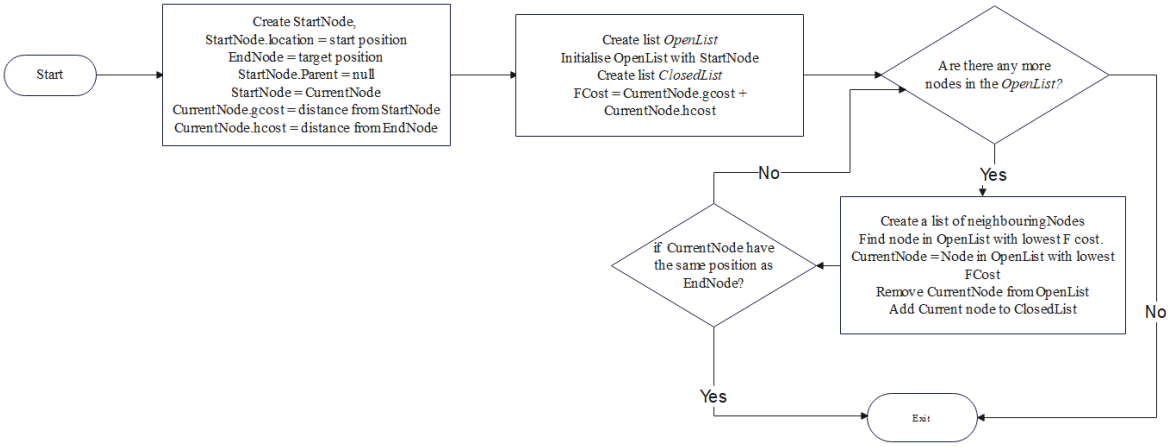


Figure 2: Slightly more complex flowchart of program

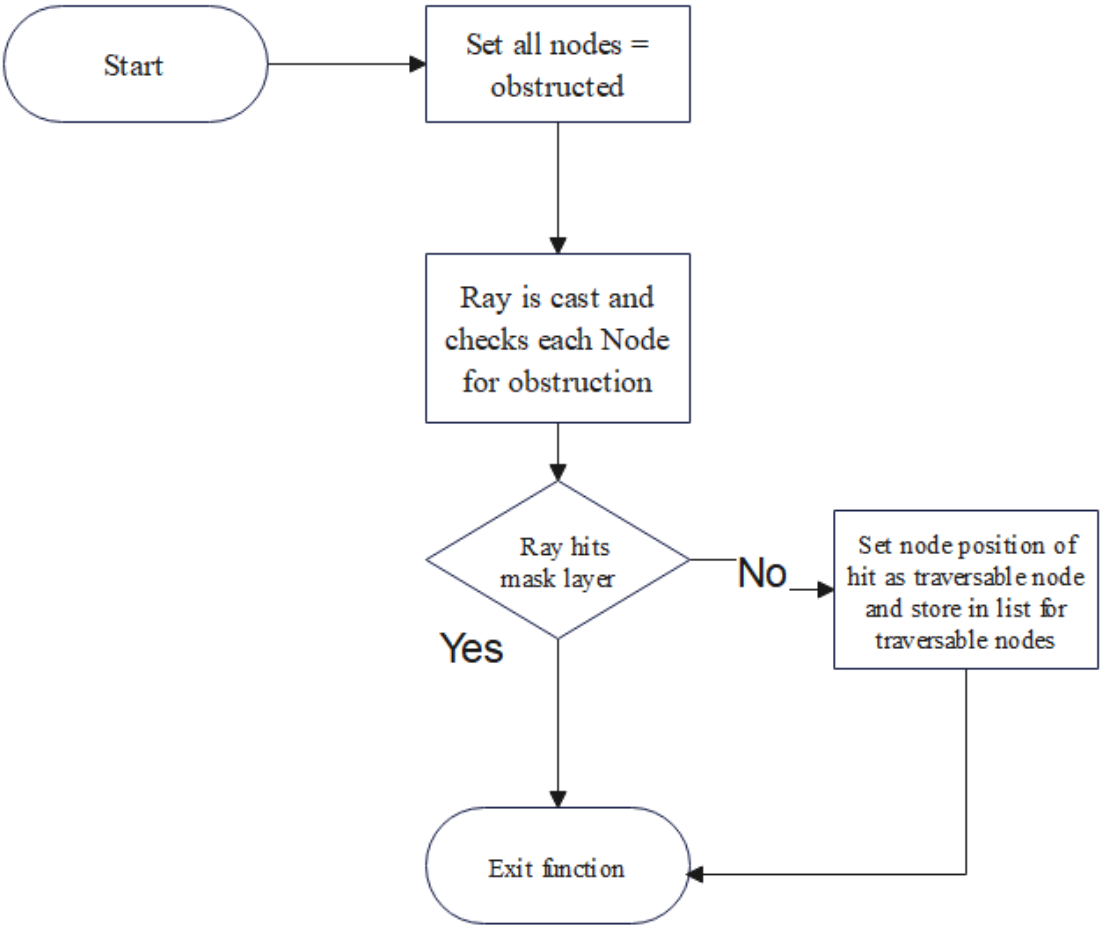


Figure 3: Flowchart of the program assessing if a node is obstructed

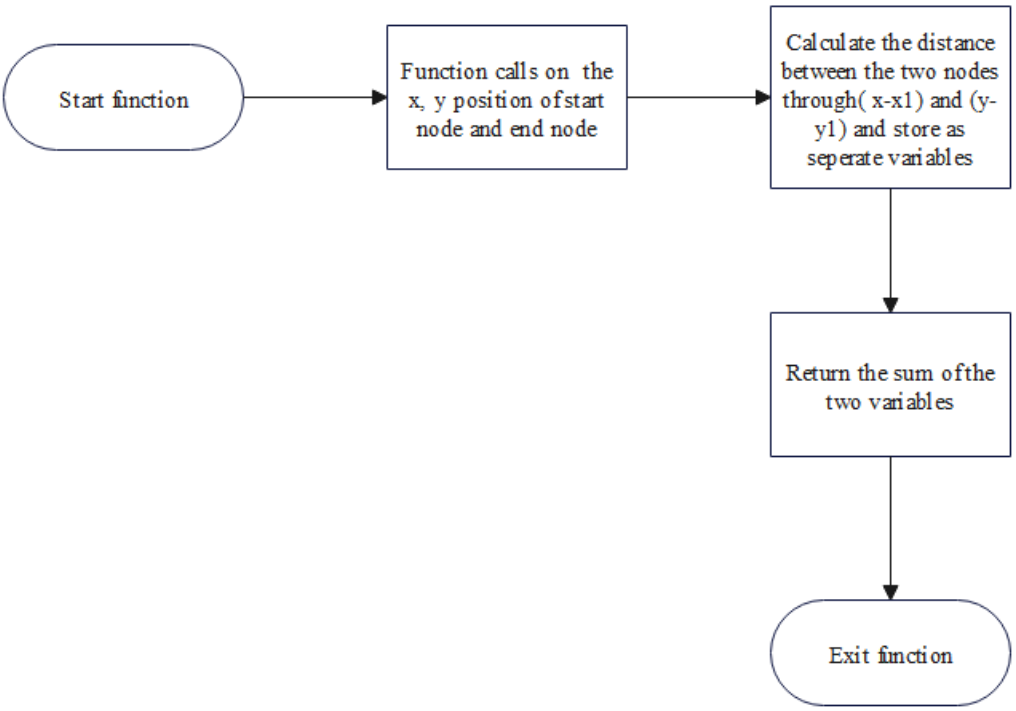


Figure 4: Flowchart of the program for manhattan distance

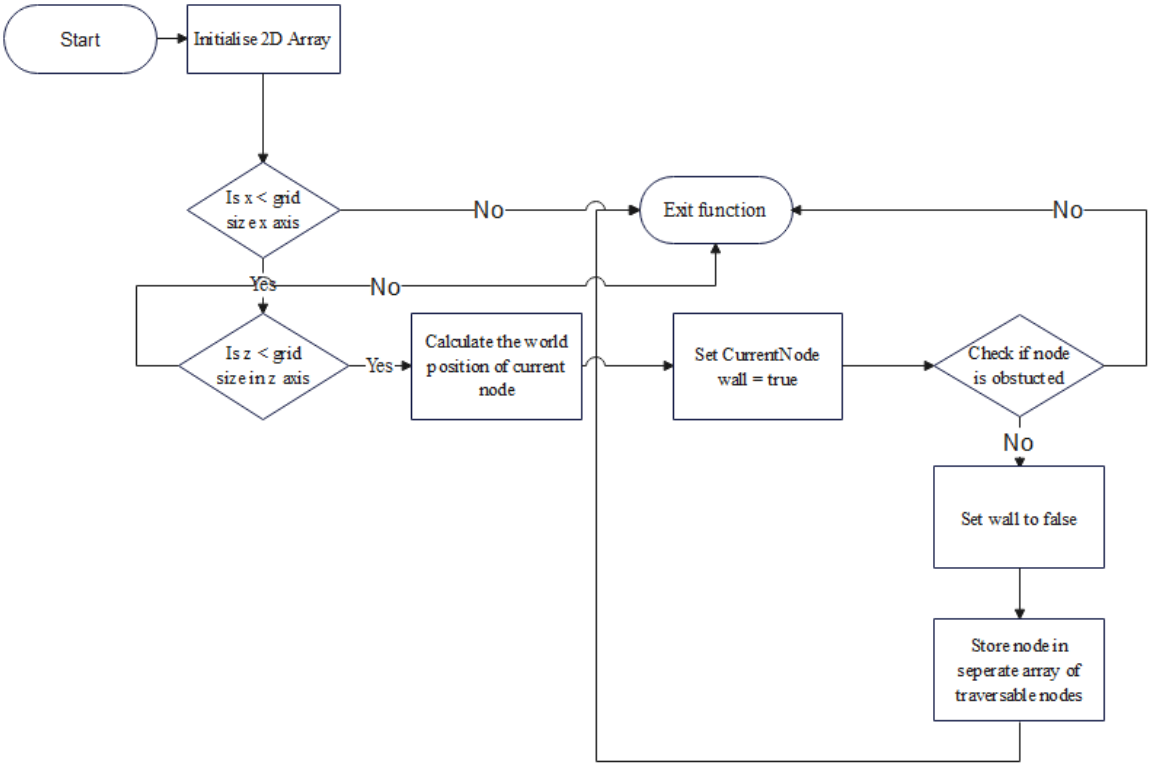


Figure 5: Flowchart of the program for creating a grid

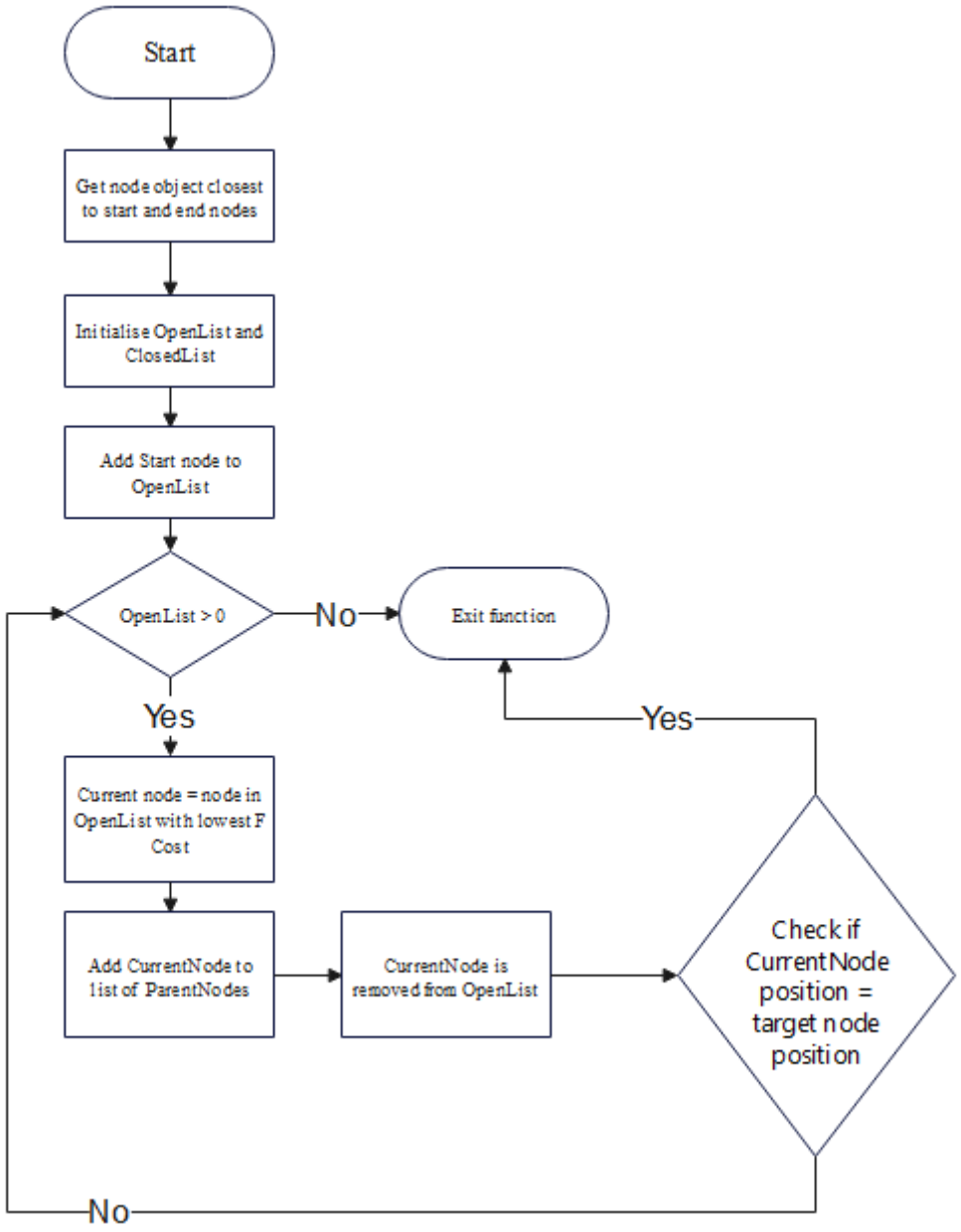


Figure 6: Flowchart for program for finding path

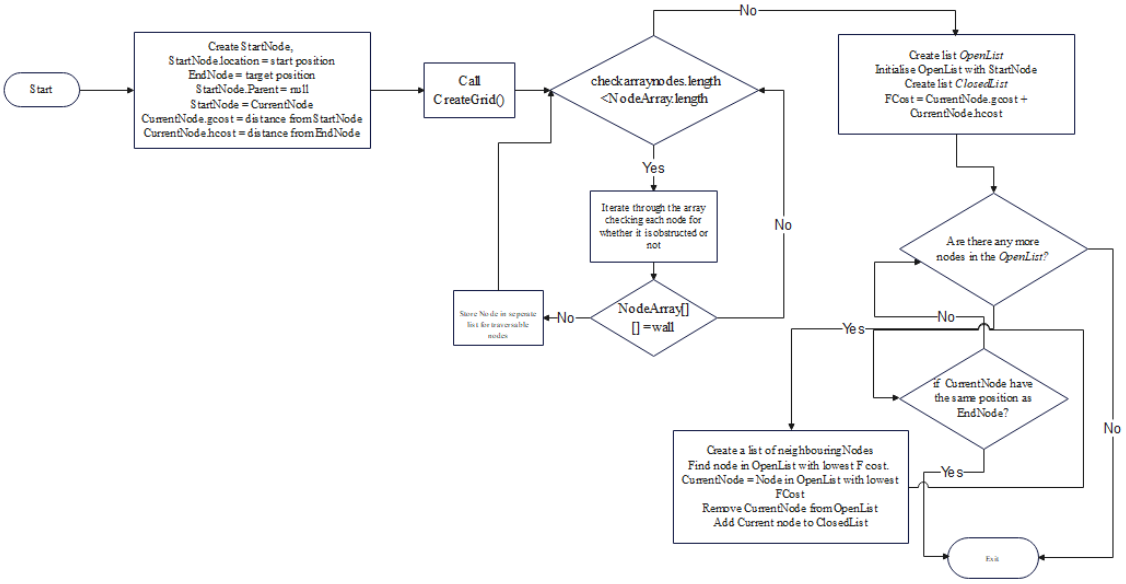


Figure 7: Flow chart of the full program

2. Pseudocode that goes into the algorithm

```
make an openlist containing only the starting node
make an empty closed list
while (the destination node has not been reached):
    consider the node with the lowest f score in the open list
    if (this node is our destination node) :
        we are finished
    if not:
        put the current node in the closed list and look at all of its neighbors
        for (each neighbor of the current node):
            if (neighbor has lower g value than current and is in the closed list) :
                replace the neighbor with the new, lower, g value
                current node is now the neighbor's parent
            else if (current g value is lower and this neighbor is in the open list ) :
                replace the neighbor with the new, lower, g value
                change the neighbor's parent to our current node

            else if this neighbor is not in both lists:
                add it to the open list and set its g
```

Figure 3: Pseudocode for the A* algorithm written in a Python-like syntax

3. Algorithm

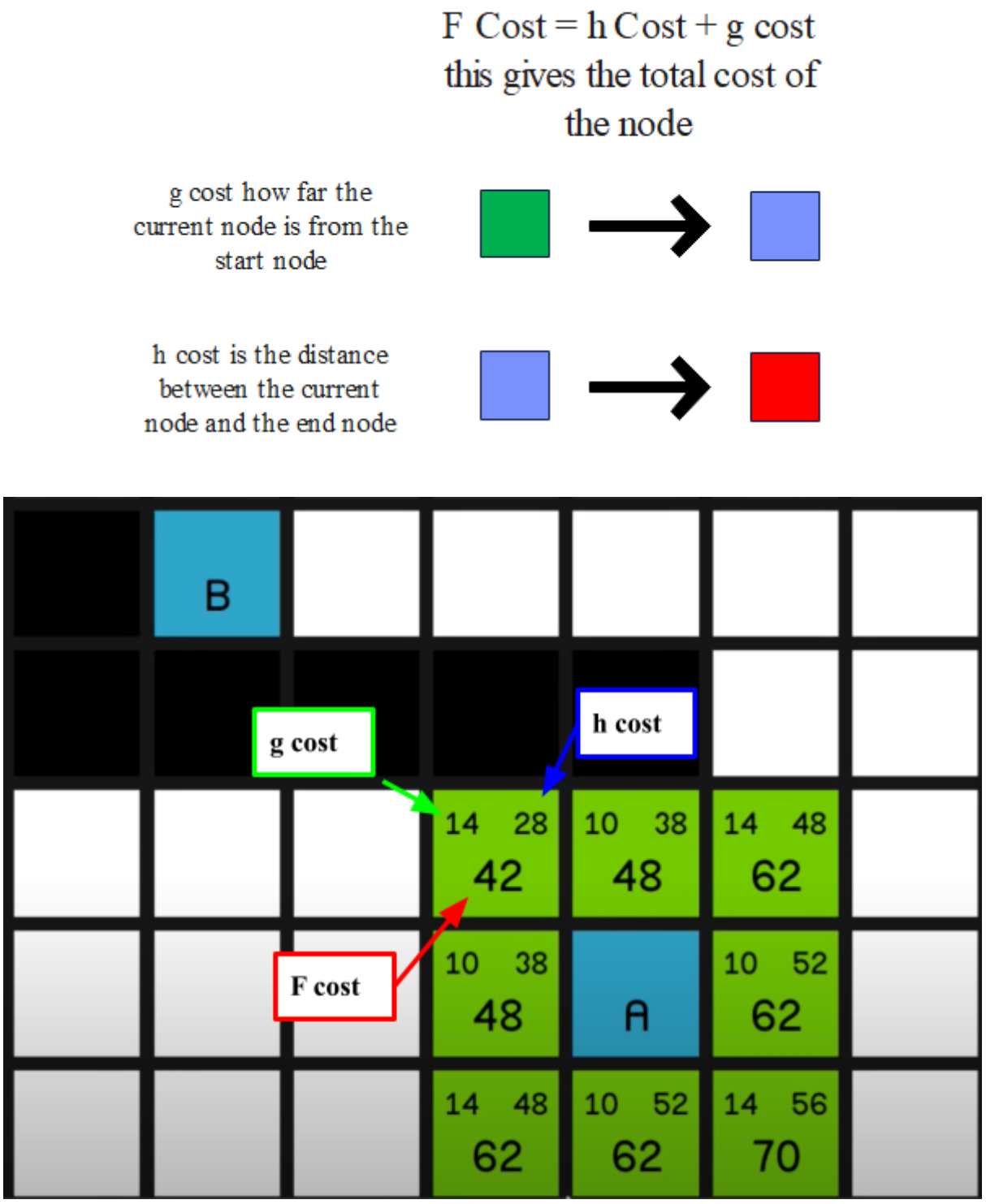


Figure 4: Pathfinding process visualised[1]

[1] Lague, Sebastian. "A* Pathfinding (E01: algorithm explanation)." *YouTube*, SebastianLague, 16 December 2014, https://www.youtube.com/watch?v=-L-WgKMFuhE&ab_channel=SebastianLague. Accessed 14 December 2022.

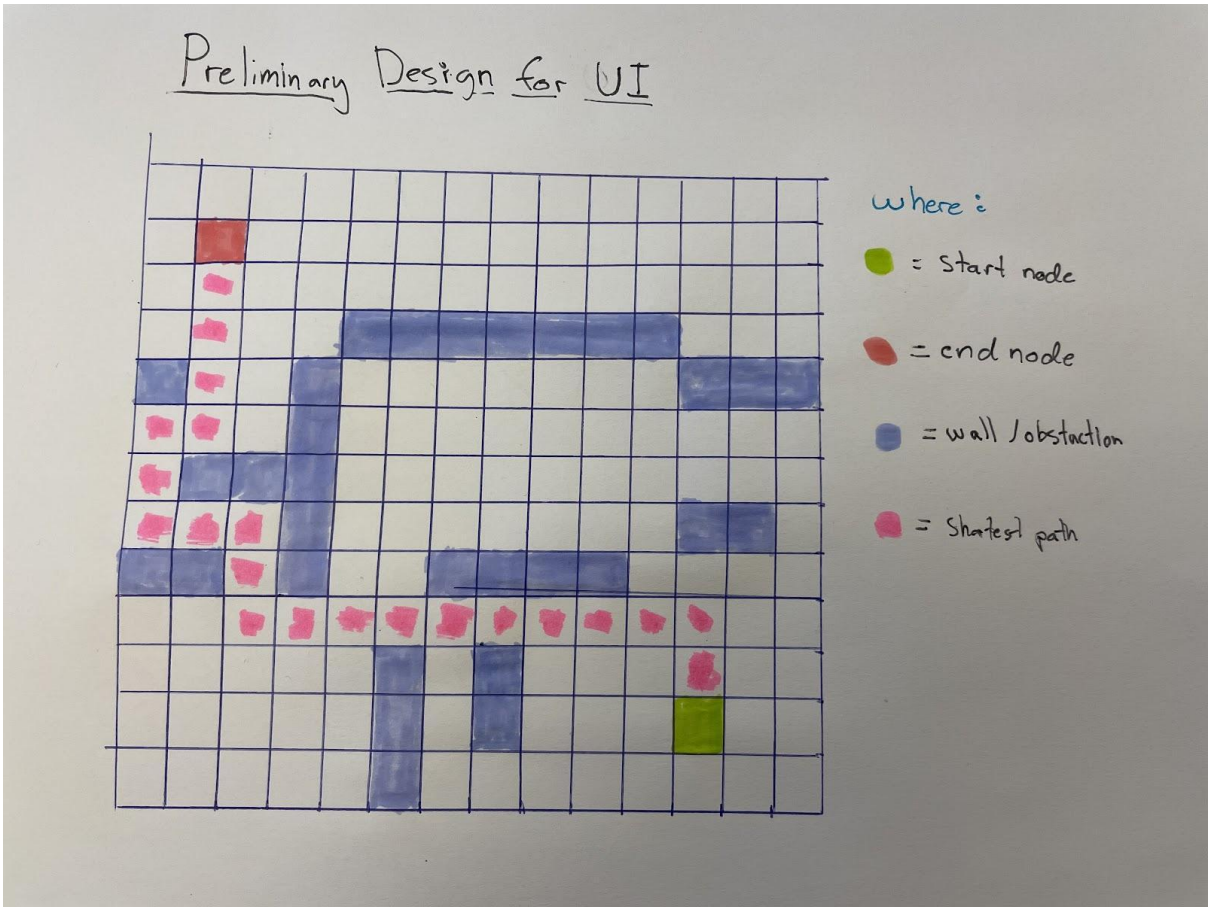
4. Test plan:

Test Number	Test	Expected input	Expected output
1	Client attempts to change the size of the grid	Client inputs the desired size of the grid.	Grid changes to client's input size
2	Client moves around the layout of the walls in order to "design" their own maze.	Client selects an obstacle in maze and drag around/change the shape of the walls to their liking.	The size/position of the obstacles changes to client's design.
3	Client changes the start and end node positions.	Client can drag start and end node around the grid.	The start and end position moves to the desired locations of the client in the grid.
4	Program runs the pathfinding algorithm.	Upon setting up the grid, obstacles and the start and end nodes to the desired preferences the client can run the algorithm.	The algorithm highlights the shortest possible path between the start node and end node while taking in to account the obstacles and grid size.

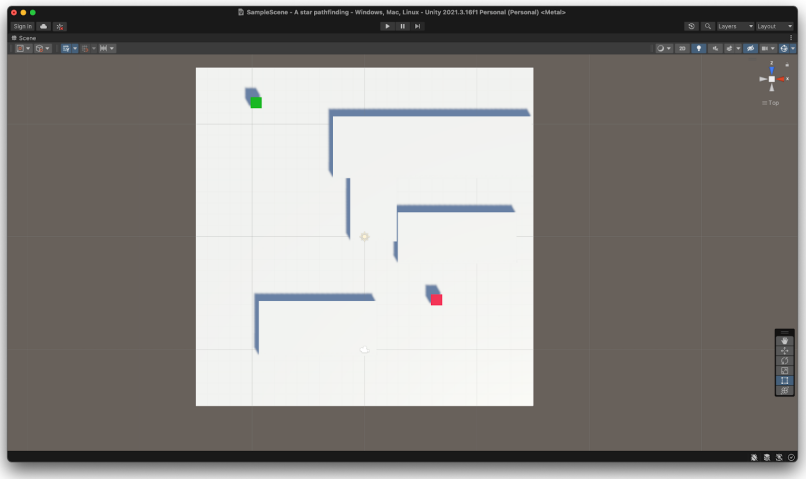
5. Design Overview:

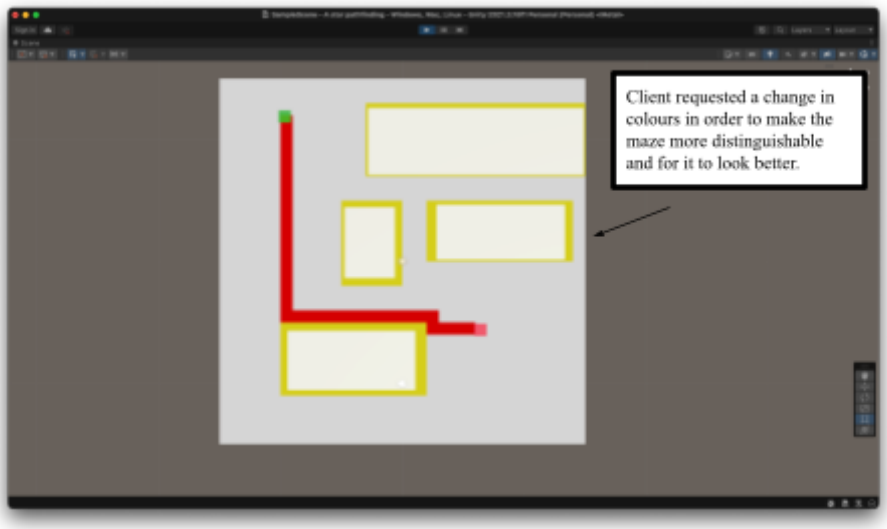
Visualisation of graphics

Initial:

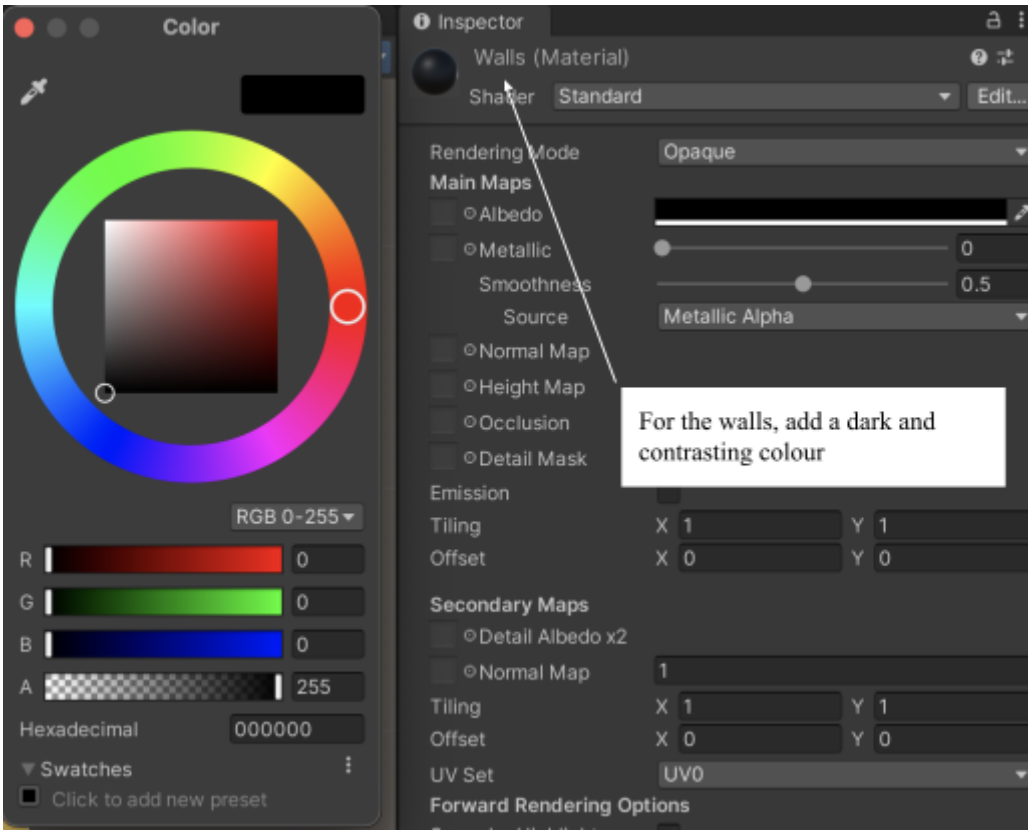


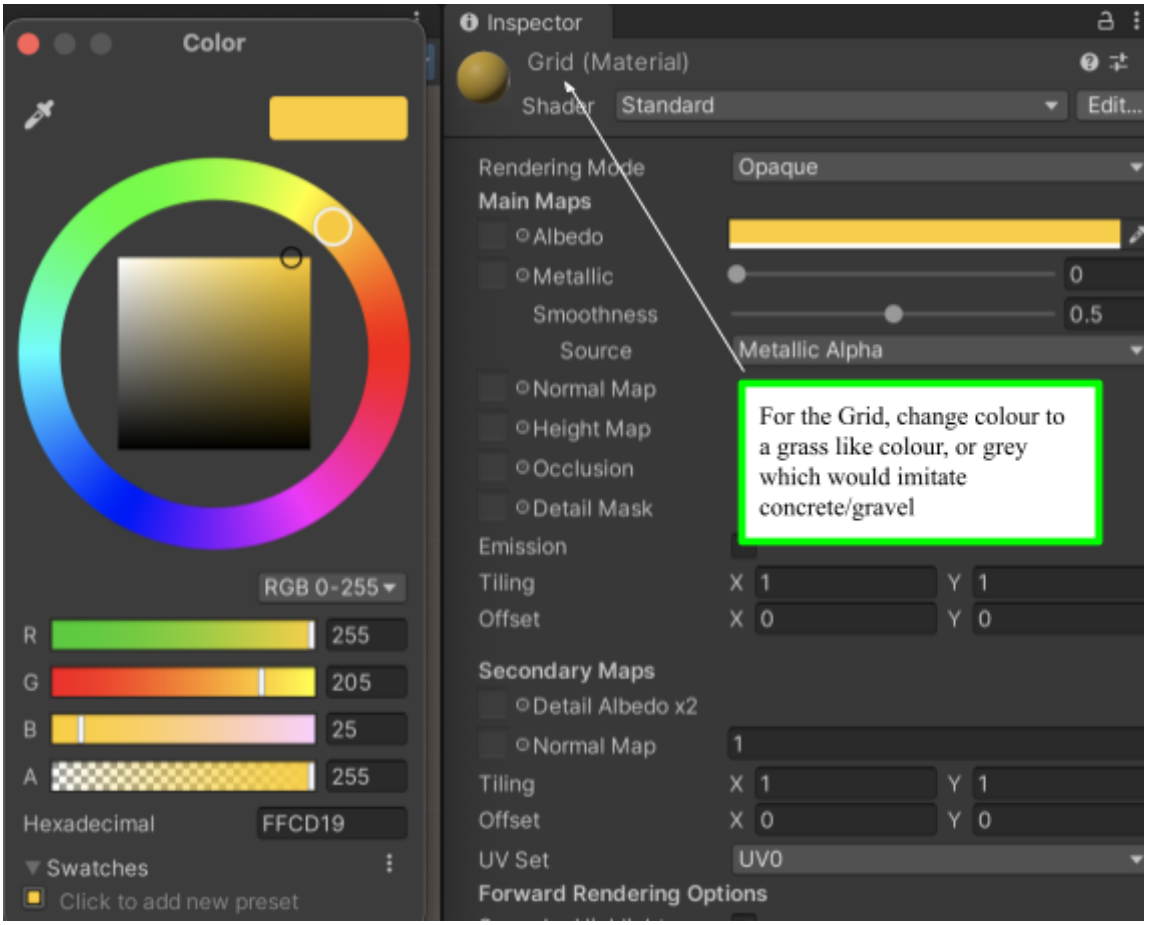
Previous versions:

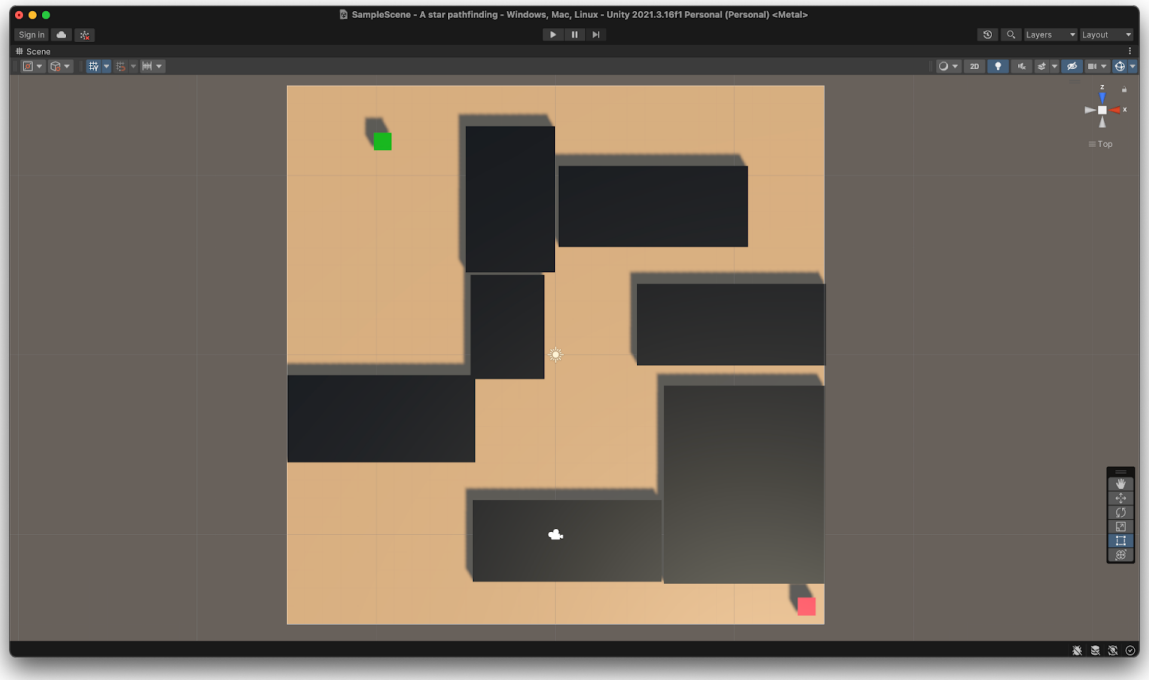


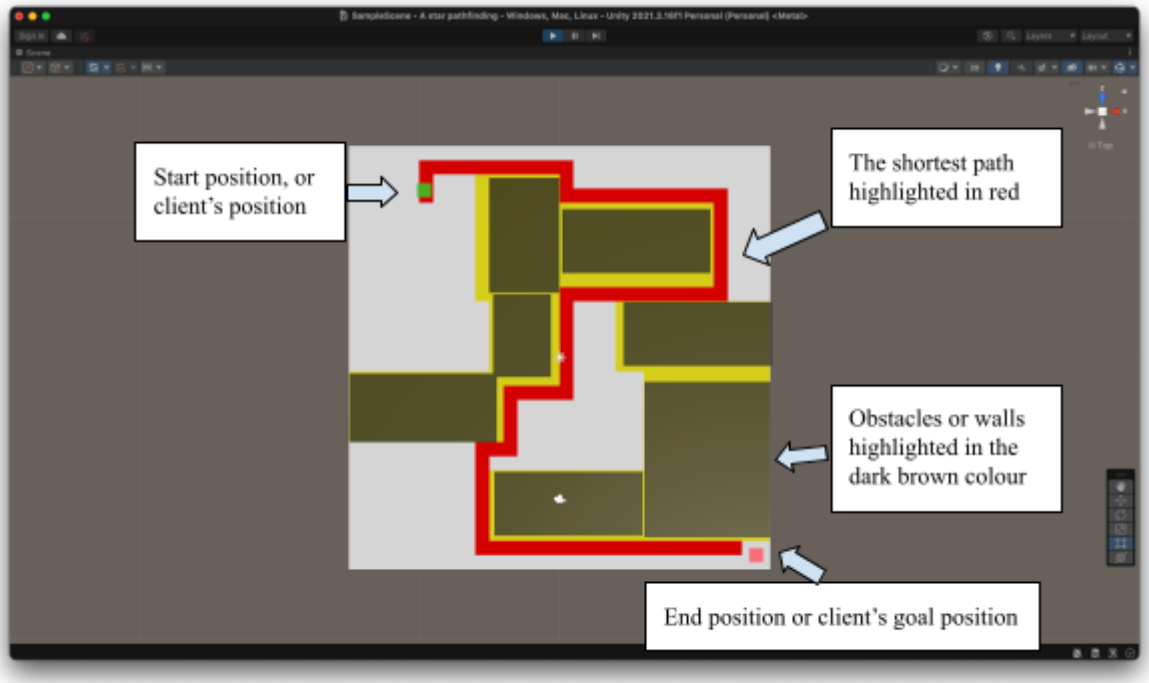


The final version of product:




Color
RGB 0-255
R 255
G 205
B 25
A 255
Hexadecimal FFCD19
Swatches
Click to add new preset
Inspector
Grid (Material)
Shader Standard
Edit...
Rendering Mode Opaque
Main Maps
Albedo
Metallic 0
Smoothness 0.5
Source Metallic Alpha
Normal Map
Height Map
Occlusion
Detail Mask
Emission
Tiling X 1 Y 1
Offset X 0 Y 0
Secondary Maps
Detail Albedo x2
Normal Map 1
Tiling X 1 Y 1
Offset X 0 Y 0
UV Set UV0
Forward Rendering Options
For the Grid, change colour to a grass like colour, or grey which would imitate concrete/gravel


SampleScene - A star pathfinding - Windows, Mac, Linux - Unity 2021.3.16f1 Personal (Personal) <Metal>



6. Evidence of versions:

1. Initial version for pathfinder:

```
using System.Collections.Generic;
using UnityEngine;

public class AStarPathfinder
{
    public static List<Vector2Int> FindPath(Vector2Int start, Vector2Int end, int[,] map)
    {
        // Create the start and end nodes
        Node startNode = new Node(start);
        Node endNode = new Node(end);

        // Create the open and closed lists
        List<Node> openList = new List<Node> { startNode };
        HashSet<Node> closedSet = new HashSet<Node>();

        // Loop until the end node is found or there are no more nodes to check
        while (openList.Count > 0)
        {
            // Get the node with the lowest F score from the open list
            Node currentNode = openList[0];
            for (int i = 1; i < openList.Count; i++)
            {
                if (openList[i].fCost < currentNode.fCost ||
                    (openList[i].fCost == currentNode.fCost && openList[i].hCost < currentNode.hCost))
                {
                    currentNode = openList[i];
                }
            }

            // Move the current node from the open list to the closed list
            openList.Remove(currentNode);
            closedSet.Add(currentNode);

            // If the current node is the end node, return the path
            if (currentNode.position == endNode.position)
            {
                return RetracePath(startNode, endNode);
            }

            // Check each neighbor of the current node
            foreach (Vector2Int neighborPos in GetNeighborPositions(currentNode.position, map))
            {
                // Skip this neighbor if it is already in the closed list
                if (closedSet.Contains(new Node(neighborPos)))
                {
                    continue;
                }

                // Calculate the tentative G score for this neighbor
                int tentativeGCost = currentNode.gCost + GetDistance(currentNode.position, neighborPos);

                // Create a new node for this neighbor if it isn't in the open list yet
                Node neighborNode = new Node(neighborPos);
                if (!openList.Contains(neighborNode))
```

The rest of the initial code can be found in the appendix.

2. The final version of code for the pathfinder algorithm:

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class Pathfinding : MonoBehaviour {

    Grid referencingofGrid;//For referencing the grid class
    public Transform StartPosition;//Starting position to pathfind from
    public Transform TargetPosition;//Starting position to pathfind to

    private void Awake()//When the program starts
    {
        referencingofGrid = GetComponent<Grid>();//Get a reference to the game manager
    }

    private void Update()//Every frame
    {
        FindPath(StartPosition.position, TargetPosition.position);//Find a path to the goal
    }

    void FindPath(Vector3 a_StartPos, Vector3 a_TargetPos)
    {
        Node StartNode = referencingofGrid.NodeFromWorldPoint(a_StartPos);//Gets the node closest to the starting position as they are not the same thing, the node is a seperate entity from the targ
        Node TargetNode = referencingofGrid.NodeFromWorldPoint(a_TargetPos);//Gets the node closest to the target position as they are not the same thing, the node is a seperate entity from the targ

        List<Node> OpenList = new List<Node>();//A list of the nodes in the open list
        HashSet<Node> ClosedList = new HashSet<Node>();//Hashset of nodes for the closed list

        OpenList.Add(StartNode);//Add the starting node to the open list

        while(OpenList.Count > 0)// a loop that runs the following program as long as there is something in the open list.

        {
            Node CurrentNode = OpenList[0];//Create a node and set it to the first item in the open list which in this case is the starting node.
            for(int i = 1; i < OpenList.Count; i++)//This loops through the open list
            {
                if (OpenList[i].FCost < CurrentNode.FCost || OpenList[i].FCost == CurrentNode.FCost && OpenList[i].hCost < CurrentNode.hCost)//If the f cost of that object is less than or equal to t
                {
                    CurrentNode = OpenList[i];//This will set the current node to the object (the node iterated through by the loop.)
                }
            }
            OpenList.Remove(CurrentNode);//Remove the current node from the open list
            ClosedList.Add(CurrentNode);//Add the current node to the closed list

            if (CurrentNode == TargetNode)//Check if the current node is equal to the afore set target node, to check if program is finished.
            {
                GetFinalPath(StartNode, TargetNode);// if the current node is equal to the target node, then call the GetFinalPath function to calculate the final path
            }

            foreach (Node NeighborNode in referencingofGrid.GetNeighboringNodes(CurrentNode))//Loop through each neighbor of the current node by calling on the GetNeighboringNodes() function
            {
                if (!NeighborNode.IsWall || ClosedList.Contains(NeighborNode))//If the neighbor that is checked happens to be a wall or has already been checked
                {
```

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class Grid : MonoBehaviour
{


    public Transform StartPosition;// This is the start position and is where the pathfinding algorithm will begin searching from

    public LayerMask WallMask;//This is the "mask" that the program will identify when searching for an obstruction in the path.

    public Vector2 vGridWorldSize;//A vector2 which is used to store the dimensions of the height and width of the plane,
    // used over a vector 3 because we don't need to deal in three dimensions when it comes to pathfinding.

    public float fNodeRadius;// Stores the client's desired size of grid / plane

    public float fDistanceBetweenNodes;//The distance that the squares will spawn from eachother.

    Node[,] NodeArray;//An array of all the nodes that the algorithm uses
    public List<Node> FinalPath;//This final path, is the route that the algorithm has determined to be the shortest route between the two nodes.


    float fNodeDiameter;
    int GridSizeX, GridSizeY;//x and y size of the grid in array units.


    private void Start()//runs the program
    {
        fNodeDiameter = fNodeRadius * 2;//the node diameter is equal to double what the given node radius is as set by the client
        GridSizeX = Mathf.RoundToInt(vGridWorldSize.x / fNodeDiameter);//This divides the grids "world co-ordinates" by the diameter to get the size of the graph in array units.
        GridSizeY = Mathf.RoundToInt(vGridWorldSize.y / fNodeDiameter);//Divide the grids world co-ordinates by the diameter to get the size of the graph in array units.
        CreateGrid();//Draw the grid
    }

    void CreateGrid()
    {
        NodeArray = new Node[GridSizeX, GridSizeY];//This declares the array of nodes.
        Vector3 bottomLeft = transform.position - Vector3.right * vGridWorldSize.x / 2 - Vector3.forward * vGridWorldSize.y / 2;
        //Get the real world position of the bottom left of the grid.

        for (int x = 0; x < GridSizeX; x++)//This loop, loops through the array of nodes which are declared by the client. Loops through the x coordinates.
        {
            for (int y = 0; y < GridSizeY; y++)//Loop through the array of nodes. Loops through the y coordinates.
            {
                Vector3 worldPoint = bottomLeft + Vector3.right * (x * fNodeDiameter + fNodeRadius) + Vector3.forward * (y * fNodeDiameter + fNodeRadius);//Get the world co ordinates of the bottom l
                bool Wall = true;//Make the node a wall (therefore, by default every node is a wall)

                //If the node is not being obstructed
                //Quick collision check against the current node and anything in the world at its position. If it is colliding with an object with a WallMask
                //The if statement will return false.
                if (Physics.CheckSphere(worldPoint, fNodeRadius, WallMask))
                {
                    Wall = false;//The node is not a wall
```

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class Node {

    public int iGridX;// The X Position of a node inside the Node Array
    public int iGridY;// The y Position of a node inside the Node Array

    public bool IsWall;//Tells the program if this node is being obstructed // OR in other words if the node is obstructed as a wall.
    public Vector3 vPosition;//The world position of the node.

    public Node ParentNode;//This parent node is for the A* algorithm and will be stored in order to identify what node the node came from originally in order to trace the shortest path


    public int gCost;//The cost of moving to the next square.
    public int hCost;//The distance to the goal from this node.

    public int FCost { get { return gCost + hCost; } }//a simple get function is used to add the h and g cost together, seen as the FCost will never need to be edited, therefore, a funct

    public Node(bool a_IsWall, Vector3 a_vPos, int a_igridX, int a_igridY)//Constructor
    {
        IsWall = a_IsWall;//This indicates to the program whether a node is being obstructed.
        vPosition = a_vPos;//The world position of the node. Vector coordinates hence the denotion of v.
        iGridX = a_igridX;//X Position in the Node Array. Is renamed as it is to be called upon in future.
        iGridY = a_igridY;//Y Position in the Node Array Is renamed as it is to be called upon in future.
    }
}
```

```
1   using System.Collections;
2   using System.Collections.Generic;
3   using UnityEngine;
4
5   public class MoveTargetPositi : MonoBehaviour
6   {
7       public LayerMask hitlayers;
8       void Update()
9       {
10       Vector3 mouse = Input.mousePosition;//will take the mouses position
11       Ray pointPlacement = Camera.main.ScreenPointToRay(mouse); //This will then cast a "ray" to the position of the mouse
12       RaycastHit hit;
13       if (Physics.Raycast(pointPlacement, out hit, Mathf.Infinity ,hitlayers))
14       {
15           this.transform.position = hit.point; //move the target to the mouse position, allowing the program to run in real-time
16       }
17      }
18  }
```

(Full code can be found in the appendix).
Code was rewritten and split up because of significant additional code upon client specification changes.

Words(314)

Record of tasks:

Date	Action	Details	Comments/Follow up	Date completed	Criterion
09/09/2022	Discussed the problem with the client (my friend).	I was told of my friend's issue of not knowing the exact shortest route to get through the mazes for his location. I suggested that I could fix this problem and would get a rough sketch for him soon.	I went home, started writing out a preliminary sketch for the solution, and worked on my proposal to give to my teacher.	10/09/2022	A
14/09/2022	Discussed the proposal with my teacher.	I discussed the proposal with my teacher and discussed the software and algorithms to use for the project.	Need to research the algorithms and report back to the CS teacher on the details of the project.	14/09/2022	A
14/09/2022	Research on which algorithms and software to use.	I did research on the different algorithms that can be used for pathfinding and decided to use the A* algorithm to make a web app for the visualisation of the project on Visual Studio Code.	I need to discuss success criteria and design with the client.	21/09/2022	A
28/09/2022	Discussed specifications and design for the product with the client.	Discussed the specific success criteria for the product with the client.	Most criteria were set, agreed that they might need to be revised throughout the design and production of the product.	01/10/2022	A
05/10/2022	Finalised choices on what algorithms to use (A*) and software	After having tried to implement the program as a web app it was decided that code was too complicated for project and Unity3D was chosen as software	Need to inform the client of the switch and the possible changes this might have on the product.	0710/2022	B

10/10/2022	Client has been informed and has given the all clear to proceed with creating the product.	After discussing with the client the change in software and the repercussions of this were brought up.	Need to study C# more as I have no experience with the programming language and it is needed for programming on Unity3D.	01/11/2022	B
05/11/2022	Course on Unity3D, C# and pathfinding	Took a basic course on C# on Youtube in order to improve my knowledge on the subject. Started making notes on DanielUnity A* pathfinding video as well	I should start writing the flowcharts and pseudocode for the algorithms that I intend on using	12/11/2022	B
11/11/2022	Finished flowcharts for the program and pseudocode	Wrote flowcharts for most of the complex programs used.	Design the GUI for the product	13/11/2022	B
13/11/2022	Finished the GUI plans and visualisation	Drew an initial GUI plan for visualisation for the client. Then shown to the client for thoughts and ideas.	Create the Test Plan table and show to the client.	17/11/2022	B
16/11/2022	Finished the Test plan and showed it to the client	Was given the all clear from the client on the test plan.	Finish up Criterion B document	18/11/2022	B
18/11/2022	Finished the Criterion B document	Criteria B document finished, only thing left is record of tasks which will be completed as time progresses.	Begin Criterion C document	20/11/2022	B
20/11/2022	Criterion C document	Began writing the code for the techniques used starting with the Raycast	Finish the code for the Raycast function which lets the program run in real-time to the adjustment of start and end node (more functionality)	30/11/2022	C
02/12/2022	Raycast function	Raycast function is fully written	Finish a function which creates a grid	20/12/2022	C

		following a guide on how to use it and the Unity documentation for the operator	of nodes for the pathfinding algorithm to use		
21/12/2022	Create Grid function complete	Following a tutorial and the pseudocode and flowcharts written for the program the create grid function was written.	Finish a function that finds the path (pathfinder)	06/01/2023	C
05/01/2023	Pathfinder complete	After more research on heuristics and how they work, how to implement then, and a guide on the code in C# for it the pathfinder algorithm was largely completed with most of the functionality written.	The guide that I followed for the pathfinder had a very efficient method of creating a node class. This needs to be done for the pathfinder that ive written to work. Finish Node class code.	20/01/2023	C
20/01/2023	Node code complete	Following the tutorial for how to design the class and information found in public pseudocode for the A* pathfinder.	Finalise the code and try to make the code run.	25/01/2023	C
25/01/2023	Product works and finished	After working around the code the product finally works and has full functionality.	Show the client the product and get feedback on their input.	26/01/2023	C
26/01/2023	Client given feedback on possible extensibility of product.	The client suggested a change of colours as well as an alert message when product doesn't work.	Finish the client feedback changes	28/01/2023	C
28/01/2023	Finished changes for the client	Finished the clients desired changes in the colour and the alert message.	Begin Alpha Testing	31/01/2023	C
04/02/2023	Alpha Testing	Completed Alpha testing with my IA	Debugging and feedback	10/02/2023	C

		supervisor and other computer science students			
10/02/2023	Debugging	Take feedback from the Alpha testing and make any necessary changes to the code because of it.	Finish Criteria C document	20/02/2023	C
19/02/2023	Finished the Criteria C document	Criteria C document fully written with the analysis of enginuity of techniques used and code relating to it.	Film the Criteria D video	21/02/2023	C
/02/2023	Filmed Criteria D video	Criteria D video of product	Begin Criteria E document	24/02/2023	D
24/02/2023	Critera E document started	Talked to client with final product in hand and discussed the pros and cons with the product as well as overall satisfaction.	Finish Test plan evaluation table	01/03/2023	E
03/03/2023	Criteria E improvements and extensions + Test plan evaluation table finished	Finish Criteria E document essentially having made comments on extensibility for improvements and whether the product was successful.	Finish Criteria E document	05/03/2023	E
05/03/2023	Criteria E document finish	Final changes made to Criteria E (word count measured, checked grammar, etc.)	-	-	E

Criterion C:

Techniques used:

1. Cast ray to the position of the mouse in real-time

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class MoveTargetPositi : MonoBehaviour
{
    public LayerMask hitlayers;
    void Update()
    {
     Vector3 mouse = Input.mousePosition;//will take the mouses position
     Ray pointPlacement = Camera.main.ScreenPointToRay(mouse); //This will then cast a "ray" to the position of the mouse
     RaycastHit hit;
     if (Physics.Raycast(pointPlacement, out hit, Mathf.Infinity ,hitlayers))
     {
         this.transform.position = hit.point; //move the target to the mouse position, allowing the program to run in real-time
     }
    }
}
```

In order for the program to calculate the real-time location of the end node, the function casts a "ray" to the location of the mouse placement. This was done using the c# unity operation Ray which takes the input of the current mouse position and casts the ray. A function such as this was one of the reasons why Unity3D was chosen for the modelling software. [2]

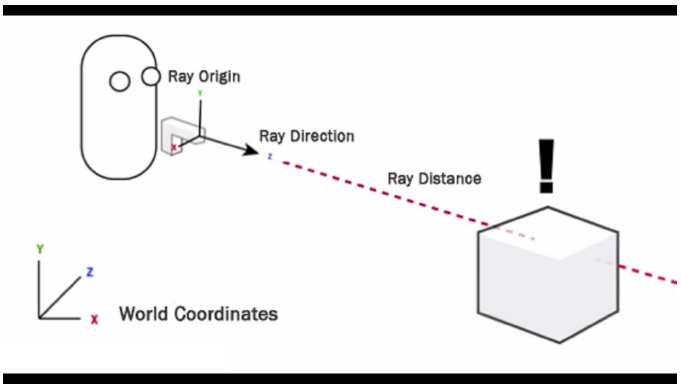


Figure: Image of how raycast function works[3]

[2] French, Leonard. "Raycasts in Unity, made easy." *Game Dev Beginner*, 18 June 2021, https://gamedevbeginner.com/raycasts-in-unity-made-easy/. Accessed 15 January 2023.

[3] "Was (Not Was) - Somewhere In America There's A Street Named After My Dad." *YouTube*, 10 May 2017, https://i.ytimg.com/vi/aOlgXDe6x3M/maxresdefault.jpg. Accessed 31 March 2023.

2. Pathfinding

Finding the path to traverse:

```
void FindPath(Vector3 a_StartPos, Vector3 a_TargetPos)
{
    Node StartNode = referencingofGrid.NodeFromWorldPoint(a_StartPos);//Ge
    Node TargetNode = referencingofGrid.NodeFromWorldPoint(a_TargetPos);//
```

To find the path the function takes two parameters (start position and target position) and fetches the closest in-array nodes to both positions using NodeFromWorldPoint which takes the Vector3 position with reference to the grid size.

```
List<Node> OpenList = new List<Node>();//A list of the nodes in the open list
HashSet<Node> ClosedList = new HashSet<Node>();//Hashset of nodes for the closed list

OpenList.Add(StartNode);//Add the starting node to the open list
```

The ClosedList is kept to track the nodes that have already been evaluated and their costs calculated. This list does not need to be updated and assessed as frequently as the OpenList which is why the use between List<Node> and HashSet<Node>. Openlist.add(StartNode) adds the starting node to the open list. This is an efficient use of operators to store & manage the nodes during the search.[4]

```
while(OpenList.Count > 0)// a loop that runs the following program as long as there is something in the open list.

{
    Node CurrentNode = OpenList[0];//Create a node and set it to the first item in the open list which in this case is the starting no
    for(int i = 1; i < OpenList.Count; i++)//This loops through the open list
    {
        if (OpenList[i].FCost < CurrentNode.FCost || OpenList[i].FCost == CurrentNode.FCost && OpenList[i].hCost < CurrentNode.hCost)/
        {
            CurrentNode = OpenList[i];//This will set the current node to the object (the node iterated through by the loop.)
        }
    }
    OpenList.Remove(CurrentNode);//Remove the current node from the open list
    ClosedList.Add(CurrentNode);//Add the current node to the closed list
```

A for loop is then used to run the program that follows as long as there is something in the OpenList. Inside the while loop, the CurrentNode is set to be the first node in the OpenList which is the start node. The function then loops through the remaining nodes in the OpenList to find the node with the lowest FCost value. If there were to exist multiple nodes with the same FCost, the one with the lowest hCost value is chosen. The current node is then removed from the open list and added to the closed list, so that it doesn't get reassessed.

```
if (CurrentNode == TargetNode)//Check if the current node is equal to the afore set target node, to check if program is finished.
{
    GetFinalPath(StartNode, TargetNode);// if the current node is equal to the target node, then call the GetFinalPath function t
}

foreach (Node NeighbouringNode in referencingofGrid.GetNeighboringNodes(CurrentNode))//Loop through each neighbour of the current
{
    if (!NeighbouringNode.IsWall || ClosedList.Contains(NeighbouringNode))//If the neighbour that is checked happens to be a wall
    {
        continue;
    }
    int MoveCost = CurrentNode.gCost + GetManhattenDistance(CurrentNode, NeighbouringNode);//Get  F cost of the neighbour which c
```

The function then checks if the CurrentNode is the target node. If it is then it calls the GetFinalPath() function to calculate the path. However, if the CurrentNode is not equal to the target node then it loops through each neighbouring node of the CurrentNode. If the node is not in the open set (is a wall/not

---

[4] Sabnis, Mahesh. "Understanding HashSet in C# with Examples." *DotNetCurry.com*, 28 April 2017, https://www.dotnetcurry.com/csharp/1362/hashset-csharp-with-examples. Accessed 10 Febuary 2023.

traversable) it is skipped and the program continues. Else the program calculates the movecost by adding the gCost and the Manhattan distance (the hCost).

```
    if (MoveCost < NeighbouringNode.gCost || !OpenList.Contains(NeighbouringNode))// If the movecost is greater than the neighbou
    {
        NeighbouringNode.gCost = MoveCost;// the g cost is the new f cost
        NeighbouringNode.hCost = GetManhattenDistance(NeighbouringNode, TargetNode);//Set the h cost
        NeighbouringNode.ParentNode = CurrentNode;//Set the parent of the node for retracing steps so that you can run program ba

        if(!OpenList.Contains(NeighbouringNode))//If the neighbouringnode is not in the openlist then add it to the list
        {
            OpenList.Add(NeighbouringNode);
        }
    }
}
```

If the movecost is less than the each of the neighbouring nodes then the movecost (FCost) is set to the new gCost, the manhattan distance is set as the hCost and the CurrentNode is set as the parent node which allows the algorithm to trace back the steps once path is found. The crucial factor about how the code is written is that it is highly readable and modifiable which is essential in order for me to rewrite the code adhering to the client's feedback. As well as it delegates the client to be able to choose the "settings" of the labyrinth (start & end nodes, walls, and grid size) and caters to those changes. [5]

Get final path function:

```
void GetFinalPath(Node a_StartingNode, Node a_EndNode)
{
    List<Node> FinalPath = new List<Node>();//holds the nodes that are in the final path and need to be highlighted
    Node CurrentNode = a_EndNode;//stores the current node of iteration

    while(CurrentNode != a_StartingNode)//this while loop works through each node that has been selected as current node throughout the
    {
        FinalPath.Add(CurrentNode);//add to final path
        CurrentNode = CurrentNode.ParentNode;//continue
    }

    FinalPath.Reverse();//reverse the path so it moves from start to finish instead of the other way around

    referencingofGrid.FinalPath = FinalPath;

}
```

The function GetFinalPath calls upon the starting node and target node. Then creates a list of nodes which were created in order to hold the nodes in the path. Then the CurrentNode (set as end node because it works backwards) is checked and passed as an argument through a while loop which searches through each node for its parent node. Once it finds the parent it adds the CurrentNode to the FinalPath and sets the CurrentNode to the ParentNode. The path is then reversed in order to run the path from start to finish. [6]

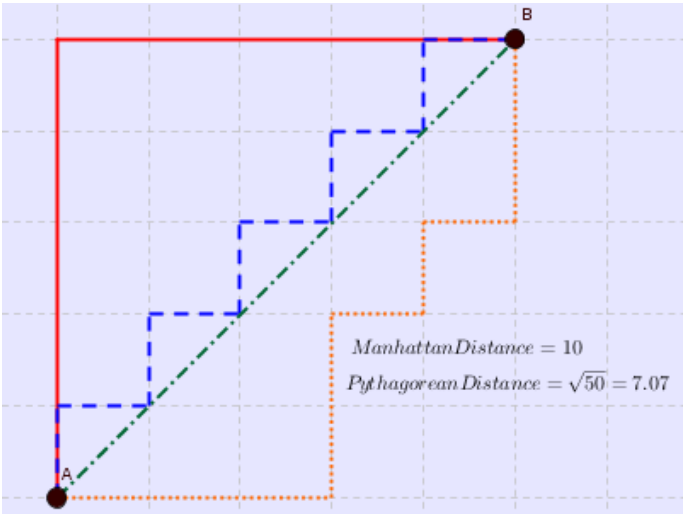
Manhattan distance:

The Manhattan distance is a distance measure that is calculated by taking the sum of distances between the *x* and *y* coordinates. This is the method used in my program and was chosen due it not taking into account the various features of the nodes. Whereas, calculating using Pythagorean distance indicates a faster method to get from point to point this assumes that all nodes are perfectly square which may not be the case in the labyrinth for the client.

---

[5] Daniel. "Unity - A Star Pathfinding Tutorial." *YouTube*, DanielUnity, 10 January 2018, https://www.youtube.com/watch?v=AKKpPmxx07w. Accessed 12 December 2022.

[6] Daniel. "Unity - A Star Pathfinding Tutorial." *YouTube*, DanielUnity, 10 January 2018, https://www.youtube.com/watch?v=AKKpPmxx07w. Accessed 12 December 2022.



Figure

The manhattan distance can be calculated using the following formula:[8]

$$Distance(x, y) = \sum_{i=1}^{n} |x_i - y_i| = |x_1 - y_1| + |x_2 - y_2| + ... + |x_n - y_n|$$

```
int GetManhattenDistance(Node a_nodeA, Node a_nodeB) or the h cost
{
    int ix = Mathf.Abs(a_nodeA.xgrid - a_nodeB.xgrid);//x1-x2
    int iy = Mathf.Abs(a_nodeA.ygrid - a_nodeB.ygrid);//y1-y2

    return ix + iy;//Return the sum of the two giving the h cost
}
```

This was the code that was used to implement the manhattan distance calculation in which a function calls upon two points, that being the current node and end node and based on the x and y coordinates of both nodes will return the sum (the manhattan distance).

3. Grid

Creating the grid:

[7] Sadhu, Koulick. "Maximum Manhattan distance between a distinct pair from N coordinates." *GeeksforGeeks*, 4 January 2023, https://www.geeksforgeeks.org/maximum-manhattan-distance-between-a-distinct-pair-from-n-coordinates/. Accessed 10 Febuary 2023.

[8] Sadhu, Koulick. "Maximum Manhattan distance between a distinct pair from N coordinates." *GeeksforGeeks*, 4 January 2023, https://www.geeksforgeeks.org/maximum-manhattan-distance-between-a-distinct-pair-from-n-coordinates/. Accessed 10 Febuary 2023.

```
void CreateGrid()// this function draws the grid
{
    NodeArray = new Node[GridSizeX, GridSizeY];//This declares the array of nodes.
    Vector3 bottomLeft = transform.position - Vector3.right * vGridWorldSize.x / 2 - Vector3.forward * vGridWorldSize.y / 2;


    for (int x = 0; x < GridSizeX; x++)//This loop, loops through the array of nodes which are declared by the client. Loops through the x coordinates
    {
        for (int y = 0; y < GridSizeY; y++)//Loop through the array of nodes. Loops through the y coordinates.
        {
            Vector3 worldPoint = bottomLeft + Vector3.right * (x * fNodeDiameter + fNodeRadius) + Vector3.forward * (y * fNodeDiameter + fNodeRadius);
            bool Wall = true;//Make the node a wall (therefore, by default every node is a wall)

            //if the node isn't obstructed then the if statement changes the return to false.
            if (Physics.CheckSphere(worldPoint, fNodeRadius, WallMask))
            {
                Wall = false;//The node is not a wall
            }

            NodeArray[x, y] = new Node(Wall, worldPoint, x, y);//Create a new node in the array and is a list of all traversable nodes.
        }
    }
}
```

NodeArray is a variable that is declared as a 2D array of nodes based on the GridSizeX and GridSizeY, which is the array that stores the nodes that the grid is made of. After looping through every node to determine if it is traversable or not. Then it is updated only to include the traversable nodes. Recalling the same variable is efficient as it would otherwise be redundant. The function uses a nested for loop to iterate through each column and row to create a new array with the position of each node which is an efficient way to initialise every node properly.

where:

x_1 is the client input size of the grid along the x-axis

x_2 is the client input size of the grid along the z-axis

and n is the client-chosen size of the nodes

Index	0	…	x_1 / n
0	Node (x,y)	Node (x,y)	Node (x,y)
…	Node (x,y)	Node (x,y)	Node (x,y)
x_2/n	Node (x,y)	Node (x,y)	Node (x,y)

Checking if a wall obstructs path or not:

```
    //if the node isn't obstructed then the if statement changes the return to false.
    if (Physics.CheckSphere(worldPoint, fNodeRadius, WallMask))
    {
        Wall = false;//The node is not a wall
    }

    NodeArray[x, y] = new Node(Wall, worldPoint, x, y);//Create a new node in the array and is a list of all traversable nodes.
```

Function “if (Physics.CheckSphere(worldPoint, fNodeRadius, WallMask))” checks whether there is an object in the scene that intersects with a sphere centred at the current node's world position (worldPoint) with a radius of fNodeRadius, and that has a layer mask specified by WallMask. If there is no object with a WallMask layer that intersects with this sphere, then the current node is not considered a wall and Wall is set to false meaning the node is traversable.[9]

Words(928)

[9] Daniel. “Unity - A Star Pathfinding Tutorial.” *YouTube*, DanielUnity, 10 January 2018, https://www.youtube.com/watch?v=AKKpPmxx07w. Accessed 12 December 2022.

Criteria D
https://www.icloud.com/iclouddrive/09d3zdZ-L2fl7KzFlsj4zv93w#Screen_Recording_2023-01-10_at_19.20

Criterion E:

Evaluating success criteria:

Success Criteria	Outcome	Discussion
The user should be able to customise the grid (plane)	Achieved	The user can enter edit mode and can alter the size of the grid.
The user should be able to alter the layout of the walls in the labyrinth	Achieved	The user can change the layout of the walls in the labyrinth by clicking on every individual obstacle and dragging them to the desired position. Upon revision of the success criteria, the client can also alter the size of the walls themselves.
The user should be able to move the location of both the starting position as well as the target position.	Archived	Same as walls
The product should generate the shortest path between the start position and end position taking into account the obstacles (walls) that obstruct the path.	Achieved	Upon the user pressing the play button, the algorithm will calculate the shortest path and calling upon the finalPath() function it will highlight the path in red.

Client feedback:
The final product was given to the client before a practice session for maze running. A conversation was then held with reference to the success criteria, and it was deemed that although successful in finding the shortest path, the means of accessibility for the client made the product quite difficult to use.

Recommendations for improving the product:
Below are listed several features or applications that can be added in order to improve the product.

1. Conversion of obstacles and grid sizes
    a. Would make the maze easier to design and follow
2. Upload the product as a web app
    a. A current problem that the client may face with the product is a lack of accessibility. As the product is not published as an application and only exists on the software, the client will, therefore, need to be in my presence whenever they need to access the product or download the software themselves. If the product was created as a web or mobile app it would provide the client with more ease when trying to use the product.
3. Use labyrinth blueprint

a. If the product was able to use a blueprint or schematic for the maze as the grid it would skip the client having to build the maze themselves.

Words(374)

Final word count(1989)

References:

Abiy, Thaddeus, et al. “A* Search.” *Brilliant*, https://brilliant.org/wiki/a-star-search/. Accessed 12 December 2022.

Daniel. “Unity - A Star Pathfinding Tutorial.” *YouTube*, DanielUnity, 10 January 2018, https://www.youtube.com/watch?v=AKKpPmxx07w. Accessed 12 December 2022.

Doxygen. “A* Pathfinding Project: RaycastModifier Class Reference.” *A* Pathfinding Project*, 6 August 2019, https://arongranberg.com/astar/documentation/dev_4_3_5_58efaa48/old/class_pathfinding_1_1_raycast_modifier.php. Accessed 12 January 2023.

French, Leonard. “Raycasts in Unity, made easy.” *Game Dev Beginner*, 18 June 2021, https://gamedevbeginner.com/raycasts-in-unity-made-easy/. Accessed 15 January 2023.

Jafri, Izrum. “What is Manhattan Distance in machine learning?” *Educative*, AICPA SOC, https://www.educative.io/answers/what-is-manhattan-distance-in-machine-learning. Accessed 10 Febuary 2023.

Lague, Sebastian. “A* Pathfinding (E01: algorithm explanation).” *YouTube*, SebastianLague, 16 December 2014, https://www.youtube.com/watch?v=-L-WgKMFuhE&ab_channel=SebastianLague. Accessed 14 December 2022.

Patel, Amit. “Introduction to A*.” *Stanford CS Theory*, 20 February 2023, http://theory.stanford.edu/~amitp/GameProgramming/AStarComparison.html. Accessed 25 February 2023.

Sadhu, Koulick. “Maximum Manhattan distance between a distinct pair from N coordinates.” *GeeksforGeeks*, 4 January 2023,

https://www.geeksforgeeks.org/maximum-manhattan-distance-between-a-distinct-pair-from-n-coordinates/. Accessed 10 Febuary 2023.

Sargent, Craig, and Dave Hillyard. "OCR OCR A Level (H446) Implementing the A-star pathfinding algorithm." *YouTube*, Craig'n'Dave, 1 July 2021, https://www.youtube.com/watch?v=Q6t-TwOGlec&ab_channel=Craig%27n%27Dave. Accessed 3 January 2023.

Singh, Anmol. "Heuristic algorithms." *Cornell University Computational Optimization Open Textbook*, 21 December 2020, https://optimization.cbe.cornell.edu/index.php?title=Heuristic_algorithms. Accessed 5 October 2022.

"Was (Not Was) - Somewhere In America There's A Street Named After My Dad." *YouTube*, 10 May 2017, https://i.ytimg.com/vi/aOlgXDe6x3M/maxresdefault.jpg. Accessed 31 March 2023.