Hugh Mulligan 15471618, Scott Weir 18728509

Software design and development Project

# Introduction

Our aim was to make a game based around interactions in between the player and enemy players in a maze like arena. We decided the multi agent nature of net logo would prove very useful in the making of interactions between the agents we wanted such as the player the enemy players and the monster guarding the goal which in this case would be a flag.

One of our main goals was to make the enemy players to act realistically and provide a challenge to the user and the main problem in this was giving them realistic path finding. In our design we wanted the AI to control the movement of agents in the created world but be made in a way that upon placing the agents into a new level or world that they would act the same without any changes needed. The AI would define a path through the virtual world given a certain set of constraints. Which in our case would be to find the shortest path to take an agent from its starting position to the goal position. This starting position and goal position can change. This pathfinding AI system would use the virtual world created as its search space.

# Studies and Background Knowledge

The first task to this project was the familiarisation with the net logo environment. It is a multiagent based environment. Agent based modelling is modelled off autonomous decision-making entities called agents. At its basic form an agent based system is all about the different agents and their relationships and interactions with each other. Each agent can be given unique attributes that define how that agent acts.

In net logo there are four types of agents turtles, patches, links and the observer. Turtles are the agents that move around the world. The world itself is a two dimensional one that is a grid that is made up of patches. Every patch is a square piece of ground in the world. The observer doesn’t have a location in the model and is mainly used as the creator of new turtles and patches as a way on interacting with the model for its generation. It observes the world that is made up of turtles and patches. When net logo starts up there are no turtles but these can be created by the observer. (Anon., n.d.)

For our approach to the project we divided up the work based off of our respective backgrounds. Scott had previous experience with net logo so was in charge of making the levels creating the agents and designing the ideas of how the agents would interact. Hugh had previous experience with graph theory and modelling so he was put in charge of the path finding algorithm to be chosen and how to implement it.

# Design

One of our first challenges was translating our idea into a set of requirements that would be implemented. We wanted to give ourselves a starting point that would help us identify how to use the tools supplied by net logo to create this game environment. We did this by exploring models supplied by the software as learning tools one of which was a simple rendition of Pac-Man.

## Requirements for the system

1. Making a prototype of the game environment and development tools.

* Take the pac man game environment supplied by net logo and strip it down and rebuild it to facilitate our objectives.
* Implementation of a map maker to allow us to easily create more levels. Once again using pac man as a base implement new turtles environments and sliders.
* Making buttons to control your character and interact with the agents.

1. Implementation of A\* search algorithm to replace current basic path finding methods, to allow the AI to dynamically search for an objective within the Maze.

* Create pseudo code for the functionality of the algorithm.
* Remove existing methods/code that allows the AI to navigate.
* Implement algorithm and test.

1. Implementation of Capture the Flag objectives, includes new turtles with unique designs and the modifying of existing code base to facilitate the flag capturing operations.

* Create visualization of capture the flag components and interaction.
* Create required turtles and designs.
* Implement new methods and variables to facilitate functionality for players and AI.

1. Modify the A\* search algorithm to allow the AI players to actively participate and compete in the capture the flag objective.

* Implement code modifications to existing algorithm and test.

## Implementation of Game Interactions

1. Patches
   1. Patch Colours
      1. Black: Generic path tile.
      2. Gray: Wall tile.
      3. Brown: Gate tile.
      4. Green: Marsh tile.
      5. Orange: Lava tile.
      6. Blue: Water tile.
   2. Patch Interactions
      1. Black: Safe for all turtles.
      2. Gray: Barrier to all turtles.
      3. Brown:
         1. Safe to player and enemy turtles.
         2. Barrier to monster turtle.
      4. Green:
         1. Safe to monster turtle.
         2. Obstacle to player and enemy turtles.
      5. Orange:
         1. Safe to monster turtle.
         2. Obstacle to player and enemy turtles.
      6. Blue:
         1. Safe to player and enemy turtles.
         2. Barrier to monster turtle.
2. Players
   1. Player Shape:
      1. Player shape starts out as ‘player’.
      2. If flag is picked up, the player shape will change to ‘playerflag’.
      3. If the player dies, the shape is changed back to ‘player’.
   2. Player Movement:
      1. Players can move along the 2D axis of x and y coordinates.
      2. Movement can be input via built in GUI.
      3. Movement can be input via assigned physical key.
   3. Player Direction:
      1. Player home position is assigned as the starting x and y coordinates.
      2. Player will automatically move forward one tile every 0.6 ticks.
      3. Player direction is true if the tile ahead is not a wall.
   4. Player Tile Interactions:
      1. Safe
         1. Player can move along any black tile.
         2. Player can move across water tile to break monster chase.
         3. Player home position is blocked by a gate, stopping access to the monster.
      2. Obstacles
         1. If the player moves onto a marsh tile, a movement debuff is applied for a counter of 10 per players current movement speed.
         2. If player moves onto a lava tile, the player loses a life and is forced to respawn back at home position.
   5. Player Turtle Interactions:
      1. Flag
         1. If the player contacts the flag, the player receives a debuff to speed, move forward 1 tile every 0.8 ticks.
      2. Monster
         1. If the player contacts the monster on the same tile, the player will die and will respawn back at home position.
      3. Enemys
         1. Player can safely interact with enemy players if neither currently have the flag in their possession.
         2. If player currently has the flag and contacts an enemy, the player will die and be respawned back at home position.
3. Enemys
   1. Enemy Shape:
      1. Enemy shape starts out as ‘enemy’.
      2. If flag is picked up, the player shape will change to ‘enemyflag’.
      3. If the enemy dies, the shape is changed back to ‘enemy’.
   2. Enemy Movement:
      1. Enemy can move along the 2D axis of x and y coordinates.
      2. Movement input is automatically controlled by computer.
   3. Enemy Direction:
      1. Enemy home position is assigned as the starting x and y coordinates.
      2. Enemy will automatically move forward one tile every 0.6 ticks.
      3. Enemy direction is true if the tile ahead is not a wall.
   4. Enemy Tile Interactions:
      1. Safe
         1. Enemy can move along any black tile.
         2. Enemy can move across water tile to break monster chase.
         3. Enemy home position is blocked by a gate, stopping access to the monster.
      2. Obstacles
         1. If the Enemy moves onto a marsh tile, a movement debuff is applied for a counter of 10 per players current movement speed.
         2. If Enemy moves onto a lava tile, the player loses a life and is forced to respawn back at home position.
   5. Enemy Turtle Interactions:
      1. Flag
         1. If the enemy contacts the flag, the player receives a debuff to speed, move forward 1 tile every 0.8 ticks.
      2. Monster
         1. If the Enemy contacts the monster on the same tile, the enemy will die and will respawn back at home position.
      3. Player
         1. Enemy can safely interact with players if neither currently have the flag in their possession.
         2. If enemy currently has the flag and contacts the player, the enemy will die and be respawned back at home position.
      4. Enemys
         1. Player can safely interact with enemy players if neither currently have the flag in their possession.
         2. If enemy currently has the flag and contacts another enemy, the enemy with the flag will die and be respawned back at home position.
4. Monster
   1. Monster Shape:
      1. Monster shape is static and doesn’t change.
   2. Monster Movement:
      1. Monster can move along the 2D axis of x and y coordinates.
      2. Movement input is automatically controlled by computer.
   3. Monster Direction:
      1. Monster home position is assigned as the starting x and y coordinates.
      2. Monster will automatically move forward one tile every 1 ticks.
      3. Monster direction is true if the tile ahead is not a wall or gate.
   4. Monster Tile Interactions:
      1. Safe
         1. Monster can move along any black tile.
         2. Monster can move across lava tile.
         3. Monster can move across marsh tile.
      2. Obstacles
         1. Monster cannot cross water tile.
         2. Monster cannot cross gate tile.
   5. Monster Turtle Interactions:
      1. Flag
         1. Monster cannot interact with the flag.
      2. Enemys
         1. If monster sees an enemy, speed is increased for the duration of current visibility, move forward one tile every 0.6 ticks.
         2. If monster contacts an enemy, the enemy will die.
      3. Players
         1. If monster sees the player, speed is increased for the duration of current visibility, move forward one tile every 0.6 ticks.
         2. If monster contacts the player, the enemy will die.
5. Flag
   1. Flag Shape:
      1. Flag shape is static and doesn’t change.
   2. Flag Movement:
      1. Flag is a static object and doesn’t move freely.
      2. Flag position is controlled by Turtle interactions.
   3. Flag Direction:
      1. Flag home position is assigned as the starting x and y coordinates.
      2. Flag drop position is assigned as the x and y coordinates that a player or enemy dies to the monster.
   4. Flag Tile Interactions:
      1. Flag, if dropped, will remain on any tile that is safe to players or enemy turtles.
   5. Flag Turtle Interactions:
      1. Enemys
         1. If enemy contacts the flag, the flag turtle will be set to hidden and a variable called visible will be set to false to ensure that only one flag object is in play.
         2. If the enemy manages to return to their home position to capture the flag, the flag is then set to respawn at its home position with flag turtle set to show and visible set to true.
      2. Players
         1. If player contacts the flag, the flag turtle will be set to hidden and a variable called visible will be set to false to ensure that only one flag object is in play.
         2. If the player manages to return to their home position to capture the flag, the flag is then set to respawn at its home position with flag turtle set to show and visible set to true.
6. Game Conditions
   1. Win
      1. If player first to reach 3 captures.
      2. If player is the last man standing.
   2. Loss
      1. If player runs out of lives.
      2. If enemy reaches 3 captures.

# Algorithms

We will be using algorithms for an automated method for solving a maze. Or in this case the path to the flag in the level.

Mazes without any loops are known as perfect mazes and are quite similar to a tree diagram.

Quite closely related to graph theory(A graph is a diagram consisting of a collection of vertices (dots),some pairs of which are connected by edges).

For net logo since we are dealing with tiles known as patches. Each patch can be thought as a vertex and the patches adjacent to this vertex which are walkable can be thought of as connected to it by an edge. It is also a connected graph as there is a path between all of the nodes otherwise we would have sections that are isolated by themselves. A connected graph is defined as an undirected graph is connected when there is a path between every pair of vertices. In a connected graph there are no unreachable vertices. A graph with just one vertex is said to be connected.

The degree of a vertex is defined as A non-negative integer which indicates the number of lines (arcs or edges) that enter the vertex. We can think of this as the number of available paths from our tile to another valid tile. Since we are dealing in a square space the maximum number for any vertices’ degree is 4. This detail may prove itself quite useful in developing our algorithms.

The distance between two vertices in a graph is the number of edges in a shortest path connecting them. Notice that may be more than one shortest path between two vertices. If there is no path connecting the two vertices then the distance is said to be infinite. For our purposes we will be specifically looking for the shortest distance between the current “activated” tile and the end goal or “GOD” tile. This will be our goal as we try to get our algorithm as close as possible to this d(x,y) value.

The most basic of your maze solving algorithms is that of the random mouse algorithm. This states that the simple robot or mouse follows a path until they hit a wall and then make a random decision on which direction to turn. This is very basic and inefficient. It’s quite similar to how or random AI performs at the moment and would be quite easy to implement.

The first step to improving would be to include a clause where the robot couldn’t turn back onto itself unless all other options were exhausted. Since we are dealing with a square like structure this would easy to implement just with a series of if else statements. If entered from a -y location then the wall met would be in the +y direction so first check -x and +x choosing one of them at first by a coinflip.

Our mazes will be mostly disjoint and our goal won’t always be an exit like in some traditional mazes so famous techniques such as the wall follower method won’t work quite as well.

To simply ensure that the maze is solved by the AI another popular algorithm is the trumeaux’s algorithm. Trumeaux’s algorithm requires drawing lines on the floor to mark a path and is guaranteed to work for all mazes with well-defined passages but not guaranteed to find the shortest route. A path from a junction is either unvisited marked once or marked twice.

1. Mark each path once as you follow it
2. Never enter a path that has been marked twice.
3. If you arrive at a junction with multiple unmarked paths follow an arbitrary one and mark it.

Otherwise

1. If the path you entered from has only one mark return down it and mark it again. In particular this should occur when the robot reaches a dead end.
2. If not choose arbitrarily one of the other paths with zero marks if not then one follow that path and mark it.

Our marks can simply be an array that holds previously visited tiles that is checked with each time the AI moves to another tile. The turn around and return rule effectively transforms any maze whether it has loops or not into a simple maze as loops are treated somewhat like dead ends. When the solution is reached the path to the goal is all the paths that are marked once. This is essentially a depth first search method of solving the maze.

# Dijkstra’s Algorithm

Dijkstra’s algorithm works off the idea that any sub path of the shortest path is also the shortest path between those two nodes. For example if the path A-B-C-D is the shortest path between A and D then B-C-D is the shortest path between B and D. Using this assumption we can then break down large pathfinding problems into simpler chunks for our program to check.

What drew us to this algorithm was the fact that is uses a weighted graph. This would allow us to apply differing weights to certain obstacles in the game such as an overgrown patch that has a slowdown effect being applied to the player would be considered by the algorithm with a weighted cost of visiting that patch.

Before the implementation of the algorithm we first broke down the algorithm to logical steps to be followed and then implemented these steps in the form of pseudocode.

1. The algorithm would start from the node that is currently occupied by the turtle which I shall refer to as the source node. It then examines the entire graph to determine the shortest path between the source node and all other nodes in the graph.
2. The algorithm keeps track of the shortest path from the source node to all currently searched nodes and updates this list as shorter paths are discovered.
3. Once the algorithm has determined the shortest path to a certain node is finalised it sets that nodes status to “visited”.
4. Using these visited nodes as a basis and adding new nodes to that list it expands out its solved area of the graph until all nodes are labelled as visited.
5. This way a shortest path to every node in the graph from the source node is known.

Dijkstra’s algorithm has a Big O value of O(ElogV) where V is the number of vertices and E is the number of edges.

## Pseudocode

Dijkstra(Graph G, source)

FOR EACH vertex v in G

distance[v] = infinity

previous[v] = undefined

distance[source] = 0

nodes = the set of all nodes in the Graph G

WHILE nodes is not empty

x = the node in nodes with the smallest distance[]

remove x from nodes

FOR EACH neighbour y of x

dist = distance[y] + distance between[x+y]

IF dist < distance[y]

Distance[y] = dist

Previous[x] = y

RETURN previous[]

(Thomas Grossmann, 2016)

# A\* Search Algorithm

The a\* (a star) search algorithm is a logical follow up to the Dijkstra’s algorithm improving the efficiency of the algorithm by setting the search in a certain direction towards the goal through the use of a heuristic. This is done with he aim of removing the necessity to calculate the shortest distance between every node on the graph to the source node and instead shorten the scope of the calculations needed to be done.

(GeeksforGeeks, n.d.)how we would like to perform our level design. At this stage of development the goal is in the centre which we can denote as (0,0). If we stick with this as a goal point then A search algorithm could be quite effective. However it doesn’t represent a real player versus ai experience as for a search multiple paths are traversed at once. This is however quite useful elsewhere in the gaming industry as you run this simulation to say that the ai character needs to get to say a door quickly and the a search algorithm supply’s it with the shortest path. This method will find the shortest path and do it quickly.

So far starting idea for a difficulty setting is that A search being the final level of difficulty and Trumeaux as a starting one.

Implement these two algorithms as a starting point

# Pseudocode

Very similar to Dijkstra’s algorithm just with an added heuristic for efficiency.

Let f(n) = g(n) + h(n)

Where

F(n) = total estimated cost of node through path n

G(n) = cost so far through path n

H(n) = estimated cost from n to goal. (heuristic)

make an openlist containing only the starting node

make an empty closed list

WHILE (the goal 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 neighbour’s

for (each neighbour of the current node):

IF (neighbour has lower g value than current and is in the closed list) :

replace the neighbour with the new, lower, g value

current node is now the neighbour's parent

ELSE IF (current g value is lower and this neighbour is in the open list ) :

replace the neighbour with the new, lower, g value

change the neighbour's parent to our current node

ELSE IF (this neighbour is not in both lists):

add it to the open list and set its g

(GeeksforGeeks, n.d.)

# Testing

For the testing of the different algorithm implementations we started with a unit testing approach by testing the path finding aspect first by returning the output to the console in net logo. Since we were working with a limited area of a 10 by 10 grid we were able to calculate what the correct result would be manually and check it against the output.

We also used unit testing to ensure that the different agents interacted with each other as intended. Such as in the use case where an enemy

Based on our requirements we used unit testing on each interaction between the agents as they were implemented independently. After a new interaction passed our unit testing we then added it to the level and used integration testing to ensure that it didn’t break any existing interaction. We followed this waterfall like approach to adding in each of our desired features and then testing that newly feature before moving onto the next feature to be added.

The next step was to test how it integrated with our movement functions and see if the path being feed to it would allow our in game turtles to move along the defined path. We did this by simply starting up the game with a simple level where the goal was at the co ordinates (0,0) on the map and observed if the non-player controlled turtles could reach the goal and manually check if the path they travelled to the goal was along the shortest path.

The next step to our integration testing was to integrate a moving goal such as the player controlled turtle having the goal and moving it back to their starting point. The non-player controlled characters would then have to be able to follow the player along the shortest path to the player. As a result of this testing we discovered a problem where if both turtles were travelling along the shortest paths to their respective goals one being the turtle holding the flags starting position and the other being the first turtle to steal the flag from them that in some chases these two shortest paths would converge and the turtle with the flag wouldn’t be able to be caught as they would be both travelling at the same speed along the same path. This we decided wasn’t an entertaining outcome for the user so we applied a speed penalty to any turtle that is currently holding the goal.

Finally we tested how the finished user experience would play out as a whole and discussed any further improvements we could make given the time. One such thing we thought of would be to allow the players to interact with the monster agent in a more meaningful way such as by hiding a separate objective such as a sword in the maze that would allow them to kill the monster. We would also like to make his behaviours more complex.

# References

Anon., n.d. *Ccl.northwestern.edu.* [Online]   
Available at: https://ccl.northwestern.edu/netlogo/docs/programming.html#:~:text=In%20NetLogo%2C%20there%20are%20four,%2C%20links%2C%20and%20the%20observer  
[Accessed 12 04 2022].

GeeksforGeeks, n.d. *GeeksforGeeks.* [Online]   
Available at: https://www.geeksforgeeks.org/a-search-algorithm/  
[Accessed 21 03 2022].

Thomas Grossmann, H. F., 2016. *www.gitta.info.* [Online]   
Available at: http://www.gitta.info/Accessibiliti/en/html/Dijkstra\_learningObject1.html  
[Accessed 15 04 2022].