"Chupacabra"

Introduction
My snake AI uses a combination of techniques to survive, grow and navigate the grid. In addition to the behaviour tree itself, there are two functions that track the length of the snake as well as evaluating if there is space free in the middle of the grid(13,13). These behaviours vary from moving to one of the 4 corners of the grid, eating the food, or moving towards its tail(hence the name "chupacabra" from Latin-American folklore). In this writeup I will explain each section of the behaviour tree, how I implemented this and my reasons.

Tactics
My tactic for the snake has come from my own play style (if I was to control the snake myself). It involves always moving towards the food unless a path becomes unavailable in which I will use various techniques such as moving to the corner of the grid or centre of the grid until a path to the food becomes available.

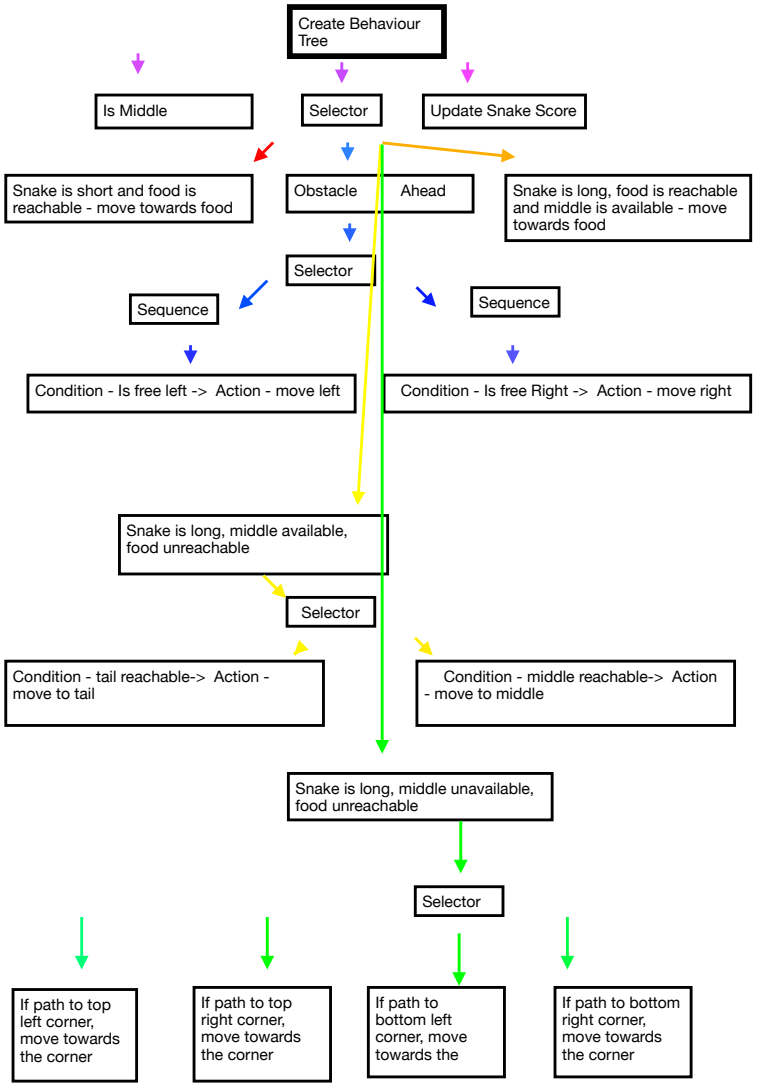
Written Functions
The code begins by calling two functions, updateSnakeScore and isMiddleReachable. updateSnakeScore processes the length of the snake. It achieves this by incrementing the snake score variable by 1 every time the position of the snakes head matches the position of the food. In retrospect, I am unsure whether the game has been made so that this will occur. I assume that within the logic of the game, when the snakes head reaches a position, it will increment the score and then place the food in a new random location. The code is written with the assumption that the script is called when the food and the snakes head are in the same position. This is because I feel that I would personally write it this way, as it seems more logical than running the script afterwards. When the score reaches 25, the tactics change, and the snake gains access to more complex behaviours as the boolean "isSnakeLong" is changed to true.

The function isMiddleReachable, checks if the centre of the grid is reachable. I could have easily implemented this code without writing my own function however I feel that it would save me the time of having to put the condition multiple times, as opposed to having a simple boolean that changes. I chose the middle of the grid as I feel that it is the most likely area to be free at a given time. The middle is defined by a 2x2 grid in the centre of the main grid. The function checks if any of these are free. Another reason for choosing this location is I feel that it is less likely for the snake to be boxed in by itself and it gives the snake a good position to move to food at any area of the grid.

Simple Behaviours
After the written functions that use standard C#, the AI begins with a selector that contains 4 different behaviours that have their own trees. The first behaviour is a filter that checks if the snake is short and will make it move towards the food. As I have the short threshold at 25, it is unlikely that the snake will struggle or turn into itself at this stage, therefore I have opted for speed.

The 2nd behaviour works if the food is reachable, the snake is long and the middle is available. The snake will move towards the food if this is the case. This is essentially the same behaviour, it just means that the snake will move towards the food whether it is short or long. This code may have been written in a different way, for instance by not

using the isSnakeLongBoolean. However, It is easier for me to visualise the workflow and to implement the behaviours so I have chosen to do this.



Complex Behaviours

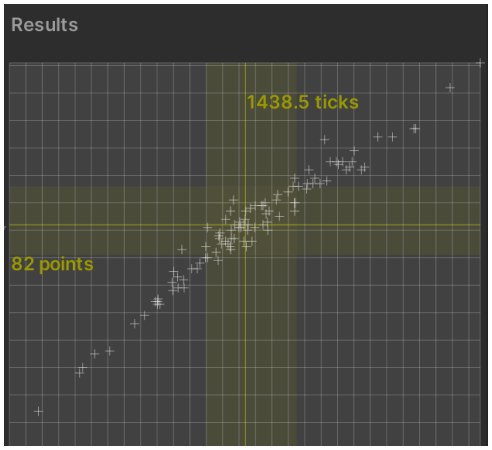
The next component of the main selector is a filter that will check if an obstacle is detected using raycasting at a specified distance ahead of it. As the snake grows longer, it becomes more cautious. This is implemented in the updateSnakeScore function that will increase the raycastDistance if the snake score is greater than 50 and then 70. In hindsight, this may have negatively affected the snake as it is not using the full scale of the grid, however I wanted the snake to reflect the way I would play and this would involve more caution past a score of 50, as I do not want to be trapped near an obstacle if I can avoid it. After a distance is detected, the snake will check if there is a free space to the left or the right of it using a random selector. If there is a free space, it will move there.

The next component is used if food is unreachable, the snake is long, and the middle is available. It will cause the snake to randomly move towards its own tail or move to the middle by using a random selector. This is so that the tail of the snake has time to free up a path to the available food. The implementation for the tail movement has been taken from the example in the assignment brief. However if the middle is available it shall move towards there instead(either one of the 4 squares located in the middle). As with many things, by placing itself in the centre of the grid, it has the option to move towards multiple directions and is strategically placed to be more likely to find a path to the food wherever it may be.

Finally if the snake is long, middle is unavailable and the food is unreachable, the snake will move to one of the 4 corners of the grid. To do this, it checks whether there is a path from the head to any of the corners and will move there. My reasons for this is that I feel that similiar to the middle, the corners are a safe bet that wherever the snake is, there will

always be a path to the corner at roughly the same length as a path to the middle. I would prefer that the snake moves to the middle which is why this scenario is the most unlikely. It is a random selector so the choice of corner will be by chance. This could be improved by working out the distance of the snakes head to any of the given corners, and then if a path is available move to the furthest corner. This would give the snake the most time for a path to the food to be free.

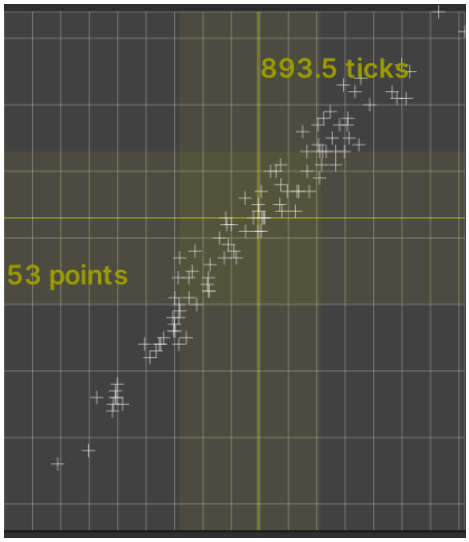
Standard Results



Results

The median result of my testing period for the standard mode was 82 points. It has a very shallow upwards curve and the results are evenly spread.

For the challenge mode I had a median of 53 points with a steep curve. What is important to note is that there are multiple failures in one spot that appears to be around the 25 mark. This may be due to the boolean change (isSnakeLong). It is unlikely to be the raycast distance as It is only changed at the 50 point mark.



Challenge Mode Results

Further Improvements

In future I would like to develop a more complex approach to raycasting. Possibly making my own functions that can check if there is space available to the left, right and diagonally from the snake. After creating a map of where these obstacles are, I can pick a point in space and move the snake towards the most suitable position.

I would also like to try an approach where I split the grid into 4 sections, and so I can create behaviours where if a route to food is unavailable, move to a different corner of the grid and use raycasting and pathfinding to decide this. I could even move to one corner of the grid and “wrap” the snake around this section by spiralling its body, this would free up space in the rest of the grid.

Furthermore, I would like to create my own pathfinding algorithm, where I can calculate the ‘risk’ of a potential route by estimating where the snakes body will be on the grid once food has been reached. If I can calculate the exact position of the snake when it reaches the food I can use this data to decide whether that route should be taken or the snake should continue moving until a safer route is available.