Snake Game AI – Report

I implemented a snake AI sing a hybrid of behaviour trees and a pathfinding solution to come with an optimal snake AI that survives and grows quickly.

Snake AI Strategy

I realized while playing with the snake game that when the snake was small, I did not need to consider the movement of the snake as it is too small to bump into itself. Hence the use of a pathfinder algorithm seemed appropriate considering only walls seemed appropriate. For pathfinding I used the *MoveTowardsFood ()* method, which is based on the Breadth First Search algorithm. The Breadth First Search algorithm calculates the shortest path on the grid without factoring in the fact that the snake is moving. This action is called only if the snake sees a clear path to the food this is thanks to the *IsFoodReachable ()* method. However, we also must consider obstacles like walls as the snake must go into a state of survival if it cannot find a clear path to the food this was implemented using *IsObstacleAhead (), IsObstacleLeft ()* and *IsObstacleRight ()* methods. These methods helped the snake realize whether there were any Obstacles around locally and make appropriate decisions (using *TurnLeft ()* and TurnRight () methods) to survive. This strategy seemed to work fine initially but as the snake grew larger it became into a bit of a problem as the snake would start crashing randomly into itself at times as it would cluster in the centre. This is a result of the snake at times greedily eating the food following the shortest path without considering its body. Hence, I reprogrammed the snake's behaviour to spread out and follow a fixed path to avoid crowding itself in the centre and crashing into it. So, after a certain length I changed the behaviour of the snake to move along the edge of the box; initially I used ray casting to determine whether there was food to the left and make the snake follow this food, but I soon found out that was a mistake as it would sometimes crash after greedily chasing all the foods to the right. To fix this I measured the distance and made sure that the snake would only attack the food if it was close by. I accomplished this by measuring distance using *RaycastLeft.Distance* and set it to a fixed short distance. Although this was difficult to determine as sometimes, I ended up stuck as food was placed in the centre, so I had to select an optimal distance that is all inclusive but not so greedy. When testing again I noticed that the snake still dies after a certain length by moving into the body. So, I used Raycasting to predict and determine when the snake would crash into the snake and avoid it by moving in a safe direction. This process can be visualized with the behaviour tree below.

Behaviour Tree

In Figure 1 below you can see the complete behaviour tree for the Snake AI; Meanwhile Figure 2 shows the behaviour of the snake with a size below 45 and Figure 3 shows behaviour when size is greater.

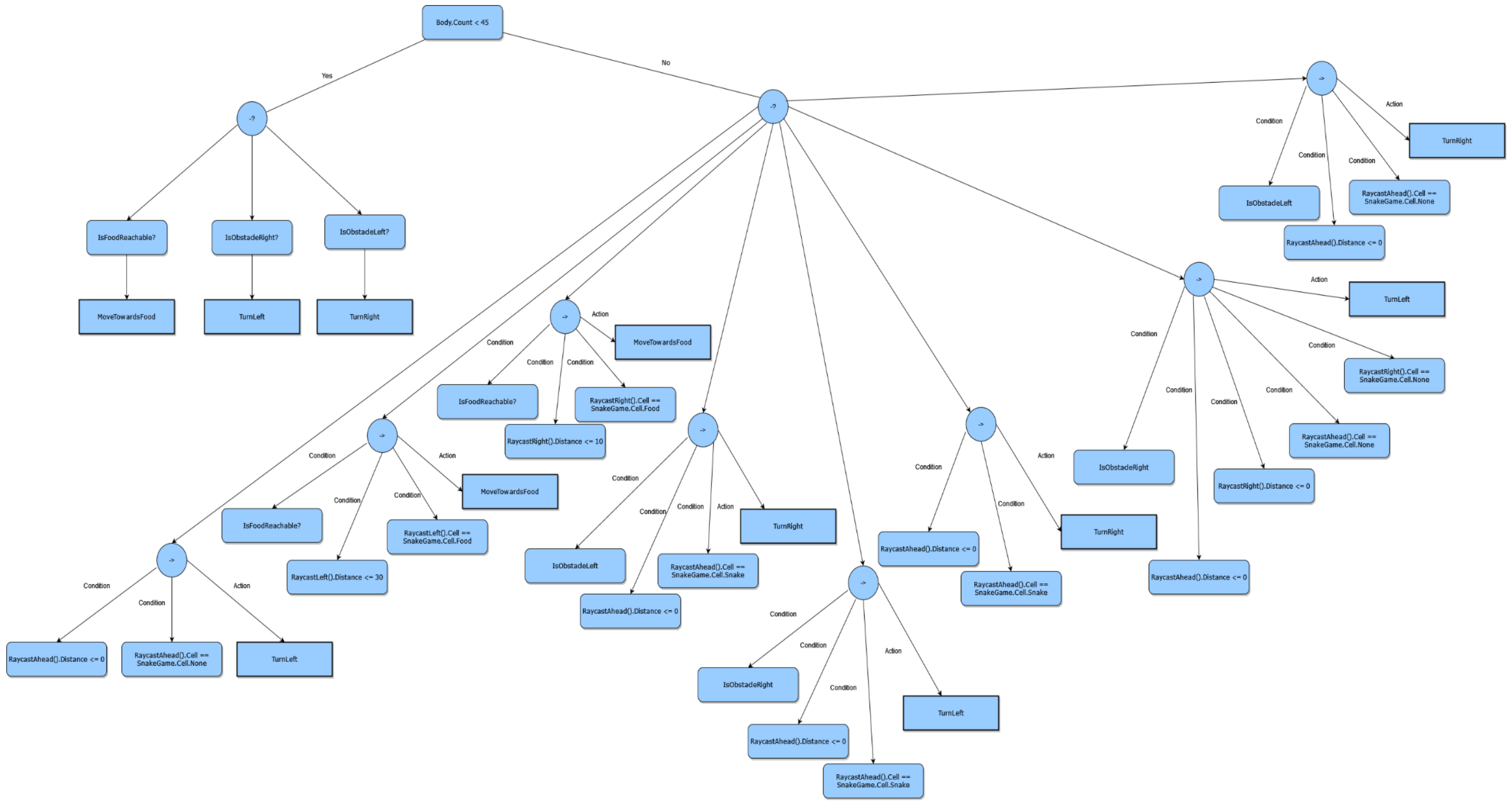

Body.Count < 45
Yes
No
-?
IsFoodReachable?
MoveTowardsFood
IsObstacleRight?
TurnLeft
IsObstacleLeft?
TurnRight
->
Condition
Action
RaycastAhead().Distance <= 0
RaycastAhead().Cell == SnakeGame.Cell.None
TurnLeft
IsFoodReachable?
RaycastLeft().Distance <= 30
RaycastLeft().Cell == SnakeGame.Cell.Food
MoveTowardsFood
IsFoodReachable?
RaycastRight().Distance <= 10
RaycastRight().Cell == SnakeGame.Cell.Food
MoveTowardsFood
IsObstacleLeft
RaycastAhead().Distance <= 0
RaycastAhead().Cell == SnakeGame.Cell.Snake
TurnRight
IsObstacleRight
RaycastAhead().Distance <= 0
RaycastAhead().Cell == SnakeGame.Cell.Snake
TurnLeft
RaycastAhead().Distance <= 0
RaycastAhead().Cell == SnakeGame.Cell.Snake
TurnRight
IsObstacleRight
RaycastAhead().Distance <= 0
RaycastRight().Distance <= 0
RaycastAhead().Cell == SnakeGame.Cell.None
RaycastRight().Cell == SnakeGame.Cell.None
TurnLeft
IsObstacleLeft
RaycastAhead().Distance <= 0
RaycastAhead().Cell == SnakeGame.Cell.None
TurnRight

Figure 1: Complete Behaviour Tree

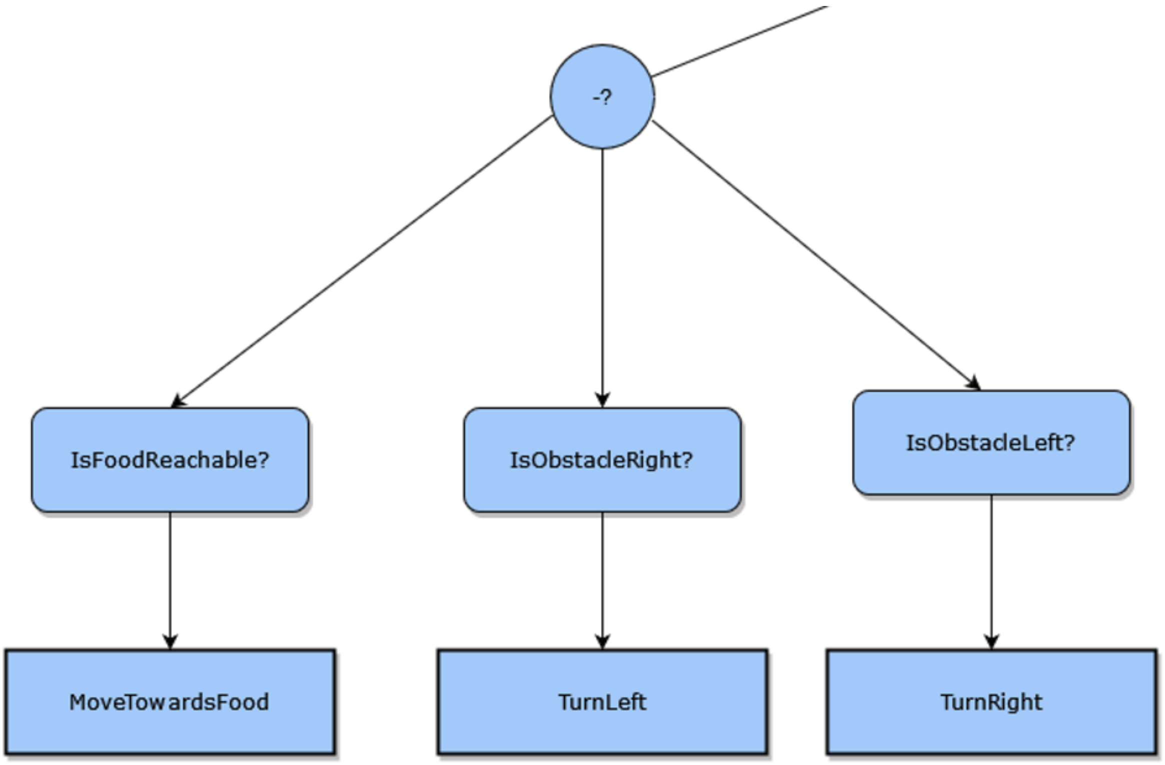


Figure 2: Snake AI behaviour tree if Snake less than 45

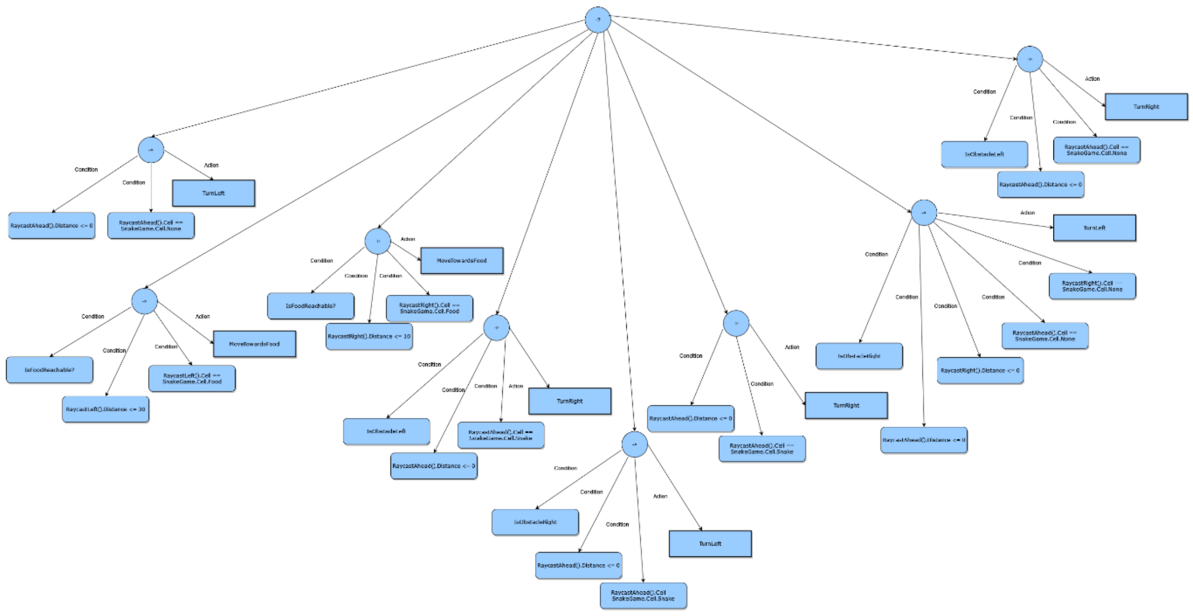
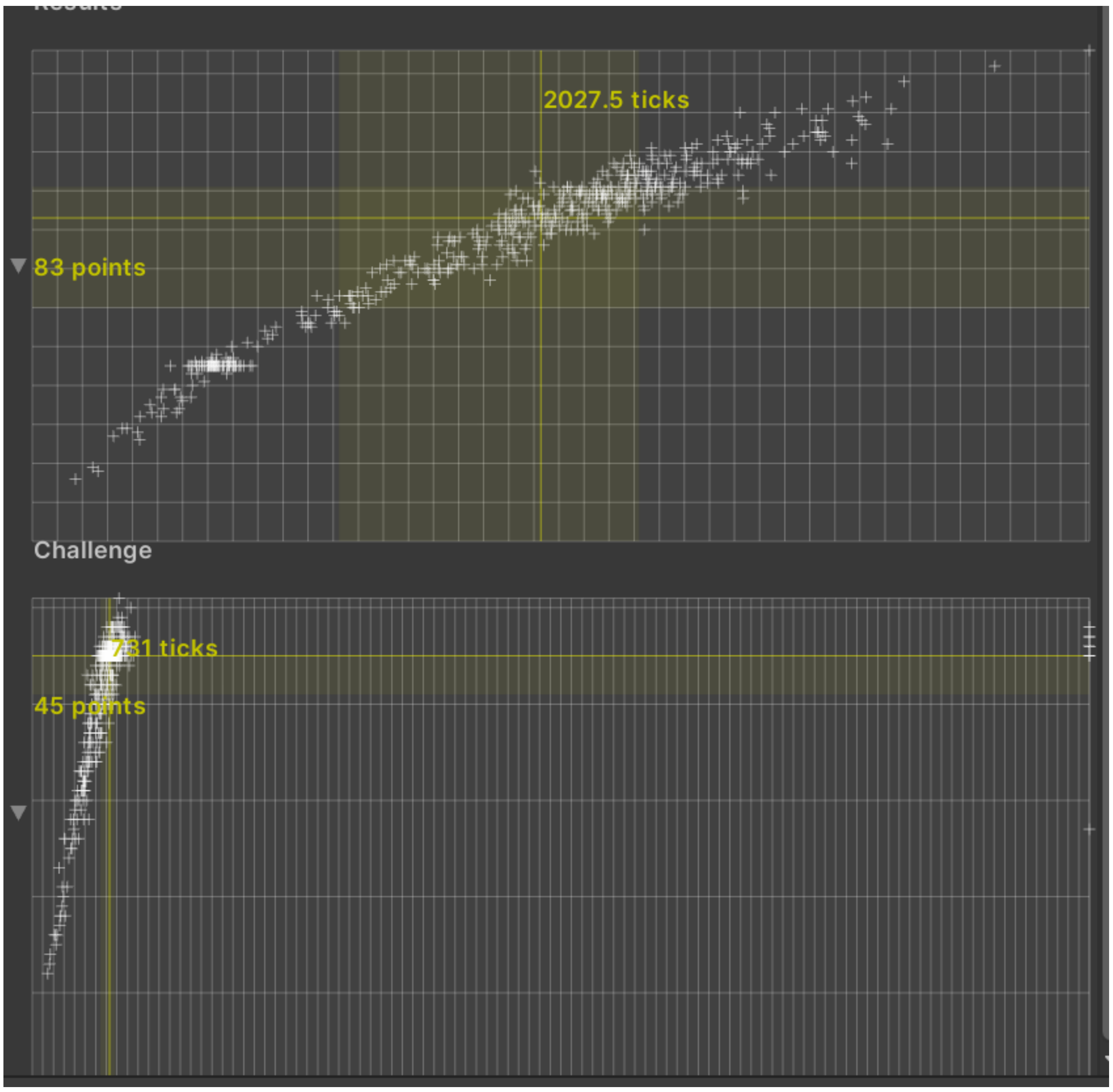

-?
->
Condition
Action
RaycastAhead().Distance <= 0
RaycastAhead().Cell == SnakeGame.Cell.None
TurnLeft
IsFoodReachable?
RaycastLeft().Distance <= 30
RaycastLeft().Cell == SnakeGame.Cell.Food
MoveTowardsFood
RaycastRight().Distance <= 10
RaycastRight().Cell == SnakeGame.Cell.Food
IsObstacleLeft
RaycastAhead().Cell == SnakeGame.Cell.Snake
TurnRight
IsObstacleRight
RaycastRight().Distance <= 0
RaycastRight().Cell == SnakeGame.Cell.None

Figure 3: Snake AI behaviour tree if Snake more than 45

Results

When testing the AI, I noticed that on average the snake got an average of 83 points in the classic mode and 45 points in the challenge mode. The result is shown below in the plot graph below which was obtained after executing the automation test.



Evaluation

I noticed that in some scenarios the snake would greedily follow a path to food which would eventually end up in closing in on itself. I also realized somewhat my snake would lead into some impossible routes with no survival path.

Future Improvements

The use of Raycasting to predict and determine when the snake would crash is somewhat still hard to implement perfectly as it sometimes must make impossible decisions like whether go into the snake or the wall however I am not sure how to program a behaviour tree to completely avoid that scenario. One way I saw was to follow an algorithm like the Hamiltonian Path which visits each cell on the grid in a loop this is more efficient than the solution I implemented however I only discovered algorithm close to the deadline. Hence, I did not have enough time to experiment with this algorithm. Another method to solve this problem is to use reinforcement Learning on the snake so it can learn all the wrong paths and make better decisions based on previous routes and mistakes. However, now this is out of my expertise as I am only aware on how to do Reinforcement Learning using Python and is outside the scope of this project.