Assignment #1: Snake AI – Report

By Omer Guven

Contents

Iteration 1

With this project I started off with simple pathfinding where I had the snake go get food if its size was less than 60 and it had more than 2 cells ahead of it free with no obstacles by making use of raycasting.

Then I created another filter where I had a condition to check if the tail is only reachable. This would only work if the body was greater than 60 or if it was less than 60 but had an obstacle less than or equal to 2 cells away from the head or the tail and food were not reachable once its size was greater than 60.

If only the tail was reachable then the snake would follow its tail until the food and the tail are reachable.

```
return
    new Selector(
        new Filter(
            new Condition(() => (Snake.RaycastAhead().Distance > 2 && Snake.Body.Count < 60)),
            new Action(Snake.MoveTowardsFood)
        ),
        new Filter(
            new Condition(IsTailOnlyReachable),
            new Action(() => Snake.MoveTowards(
                Snake
                .AvailableNeighbours(Snake.TailPosition)
                .FirstOrDefault(position => Snake.IsReachable(position)))
            )
        ),
        new Filter(
            new Condition(AreTailAndFoodReachable),
            new Action(Snake.MoveTowardsFood)
        ),
```

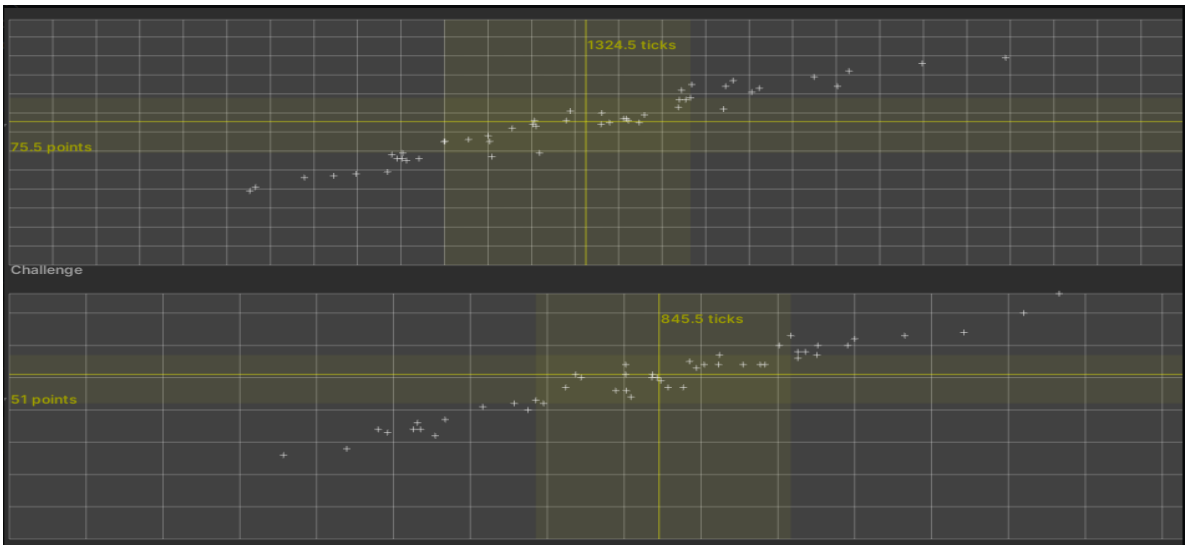
(Filters that handle basic snake movement to food and tail)

But if neither tail or food were reachable then the snake would do nothing hence, I introduced another filter which had a condition that checked if tail is not reachable and if it was not then the snake would move randomly to available places on the grid until it died or had the tail reachable.

```
    new Condition(TailNotRachable),
    new Filter(
        new Condition(() =>
        {
            List<Vector2Int> list = new List<Vector2Int>();
            for (int i = 0; i < 25; i++)
            {
                for (int j = 0; j < 25; j++)
                {
                    if (new Vector2Int(i, j) == Snake.HeadPosition)
                    {
                        continue;
                    }
                    if (Snake.IsObstacle(new Vector2Int(i, j)))
                    {
                        continue;
                    }
                    list.Add(new Vector2Int(i, j));
                }
            }

            //shuffle the list
            for (int i = 0; i < list.Count; i++)
            {
                int j = Random.Range(0, list.Count);
                Vector2Int temp = list[i];
                list[i] = list[j];
                list[j] = temp;

            }
            foreach (Vector2Int p in list)
            {
                if (Snake.IsReachable(p))
                {
                    x = p.x;
                    y = p.y;
                    return true;
                }
            }
            return false;
        }),

        new Action(() => Snake.MoveTowards(x, y))
    )
);
```

(Filter for tail is not reachable)

With this code my snake's performance from a test of 50 was:



(Results from a test of 50 in iteration 1)

Average for normal snake game was 75.5 and for the challenge was 51.

Iteration 2

During this iteration I added a few more filters with conditions and actions and changed some of the contents of existing Filters.

First, I added a filter which had a condition that checked if food and tail were not reachable. If this was true it would go into a new selector and that selector had two more filters one of the filters conditions was to raycast ahead and right and check if an obstacle is less than or equal to 2 cells away from the head, so the snake would turn left. The other filter had a condition which raycasts ahead and to the left and checks if an obstacle is less than or equal to 2 cells away from the head, so the snake would turn right.

```
new Filter(
    new Condition(FoodNotReachableAndTail),
    new Selector(
        new Filter(
            new Condition(() => Snake.RaycastAhead().Distance <= 2 && Snake.RaycastRight().Distance <= 2),
            new Action(Snake.TurnLeft)
        ),
        new Filter(
            new Condition(() => Snake.RaycastAhead().Distance <= 2 && Snake.RaycastLeft().Distance <= 2),
            new Action(Snake.TurnRight)
        )
    )
),
```

(New filter that was added)

Then I added a condition that checked if the snake could reach the food but not its tail; if so, it would move to available blocks one by one until the condition is no longer met. This is so it lives until it can reach its tail, so it has somewhere to go after getting the food.

```
//just live until tail is reachable
new Filter(
    new Condition(FoodReachableButNotTail),
        new Filter(
            new Condition(() =>
            {
                for (int i = 0; i < 25; i++)
                {
                    for (int j = 0; j < 25; j++)
                    {
                        if (new Vector2Int(i, j) == Snake.HeadPosition)
                        {
                            continue;
                        }
                        else if (Snake.IsObstacle(new Vector2Int(i, j)))
                        {
                            continue;
                        }
                        else if (Snake.IsReachable(new Vector2Int(i, j)))
                        {
                            k = i;
                            l = j;
                            return true;
                        }
                    }
                }
                return false;

            }),
            new Action(() => Snake.MoveTowards(k, l))
        )
),
```

(Filter that checks if food is reachable, but tail is not)

Finally, I changed one of my filters from iteration 1.

```
new Filter(
    new Condition(() => (Snake.RaycastAhead().Distance > 2 && Snake.Body.Count < 60)),
    new Action(Snake.MoveTowardsFood)
),
```

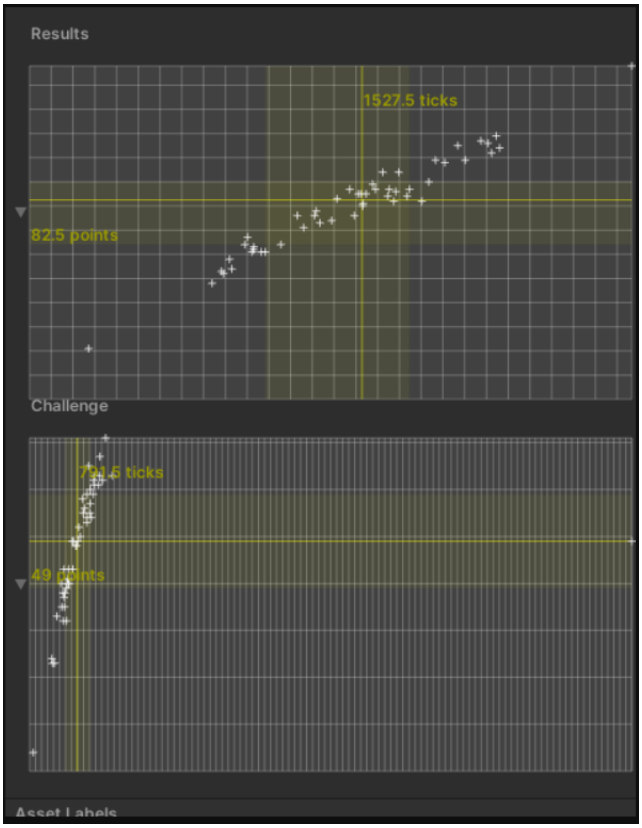
(Iteration 1 code that was altered)

```
new Filter(
    new Condition(() => Snake.Body.Count < 60 && Snake.IsFoodReachable() && Snake.RaycastAhead().Distance >= Snake.DistanceFrom(Snake.HeadPosition, Snake.FoodPosition)),
    new Action(Snake.MoveTowardsFood)
),
```

(Iteration 2 code after it was altered)

The reason behind this change was the snake was dying after it ate the food as it was causing itself to get trapped. I added is food reachable to see if the snake can get to the food without dying and raycasted ahead using the distance of the head position to the food position to make sure there were no obstacles around and if there were obstacles it would look at other filters conditions and perform their actions.

With this code my snake's performance from a test of 50 was:



(Results from a test of 50 in iteration 2)

These new changes made the code perform worse in the challenge but better in the normal snake game and the change in the challenge was not significant as it was averaging 51 before and now started to average 49 whilst living longer as well. The average for the normal snake game went up from 75.5 to 82.5 which is an increase of 7 more food being collected on average whilst living for a longer period as well compared to the results from iteration 1. So overall I believe this snake ai is performing better than the snake ai from iteration 1.

Iteration 3

During this iteration I got rid of the filter which had the condition that checked if the food and tail are not reachable.

```
//new Filter(
//    new Condition(FoodNotReachableAndTail),
//    new Selector(
//        new Filter(
//            new Condition(() => Snake.RaycastAhead().Distance <= 2 && Snake.RaycastRight().Distance <= 2),
//            new Action(Snake.TurnLeft)
//        ),
//        new Filter(
//            new Condition(() => Snake.RaycastAhead().Distance <= 2 && Snake.RaycastLeft().Distance <= 2),
//            new Action(Snake.TurnRight)
//        )
//    )
//),
```

(Removed code)

Then in the filter which checks if food is reachable, and tail is not I added ray casting ahead to the position it is going to move to, to check if it had a raycast distance that is greater than 1.

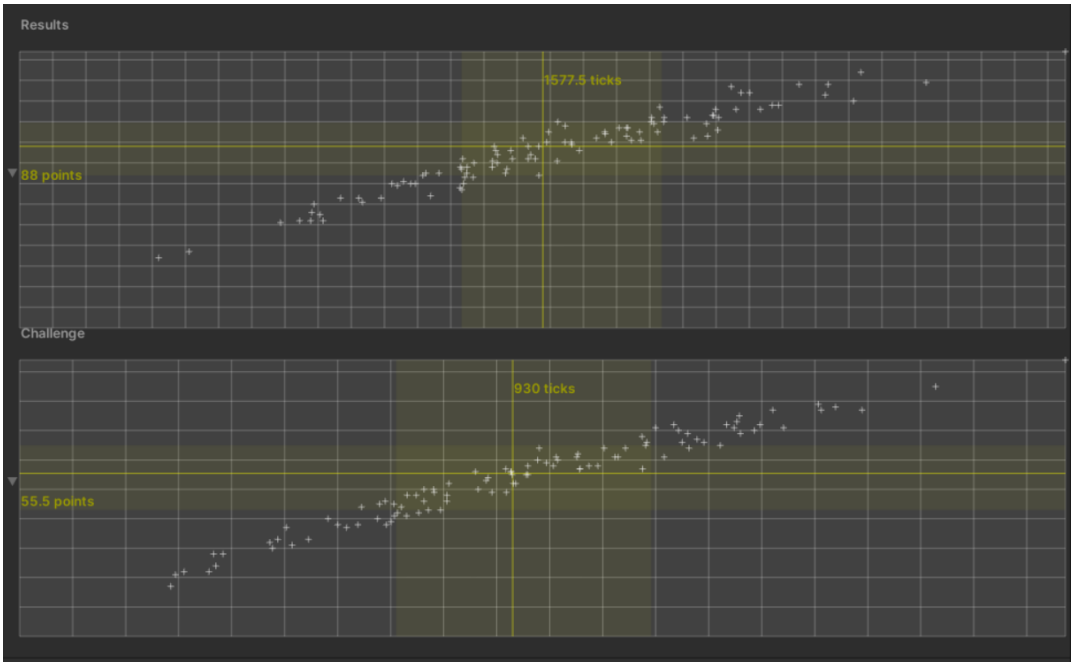
```
new Filter(
    new Condition(FoodReachableButNotTail),
        new Filter(
            new Condition(() =>
            {
                for (int i = 0; i < 25; i++)
                {
                    for (int j = 0; j < 25; j++)
                    {
                        if (new Vector2Int(i, j) == Snake.HeadPosition)
                        {
                            continue;
                        }
                        if (Snake.IsObstacle(new Vector2Int(i, j)))
                        {
                            continue;
                        }
                        if (Snake.IsReachable(new Vector2Int(i, j)) && Snake.RaycastAhead().Distance > 1)
                        {
                            k = i;
                            l = j;
                            return true;
                        }
                    }
                }
                return false;
            }),
            new Action(() => Snake.MoveTowards(k, l))
        )
),
```

(Raycast added)

Then I increased the number of foods the snake eats whilst it does pathfinding and checks if food is reachable and does not care if tail is reachable to less than 65 from 60 to see if it performed better.

```
new Filter(
    new Condition(() => Snake.Body.Count < 65 && Snake.IsFoodReachable()),
    new Action(Snake.MoveTowardsFood)
),
```

With these changes to my code my snake's performance from a test of 100 was:



(Results from a test of 100 in iteration 3)

This output shows an increase in the normal snake game, which was from an average score of 82.5 to 88 which is an increase of 5.5 and there was a slight increase in its ticks meaning that it is living longer on average as well. The average for the challenge has also increased meaning that this code is more optimal than the code from iteration 2. The Challenge increased from an average of 49 points in iteration 2 to an average of 55.5 points in iteration 3 meaning that the changes in iteration 3 helped with the overall performance of the snake ai.

Iteration 4

During this iteration I added more conditions to my filter that controlled if the snake could move to the tail. This is so the snake has more opportunities to move to the tail instead of taking food which can lead to its death.

```
new Filter(
    new Condition(() => IsTailOnlyReachable() || (Snake.IsObstacleAhead() && AreTailAndFoodReachable())),
    new Action(() => Snake.MoveTowards(
        Snake
        .AvailableNeighbours(Snake.TailPosition)
        .FirstOrDefault(position => Snake.IsReachable(position)))
    )
),
```

(Adding more conditions to give snake more opportunities to follow tail to live longer)

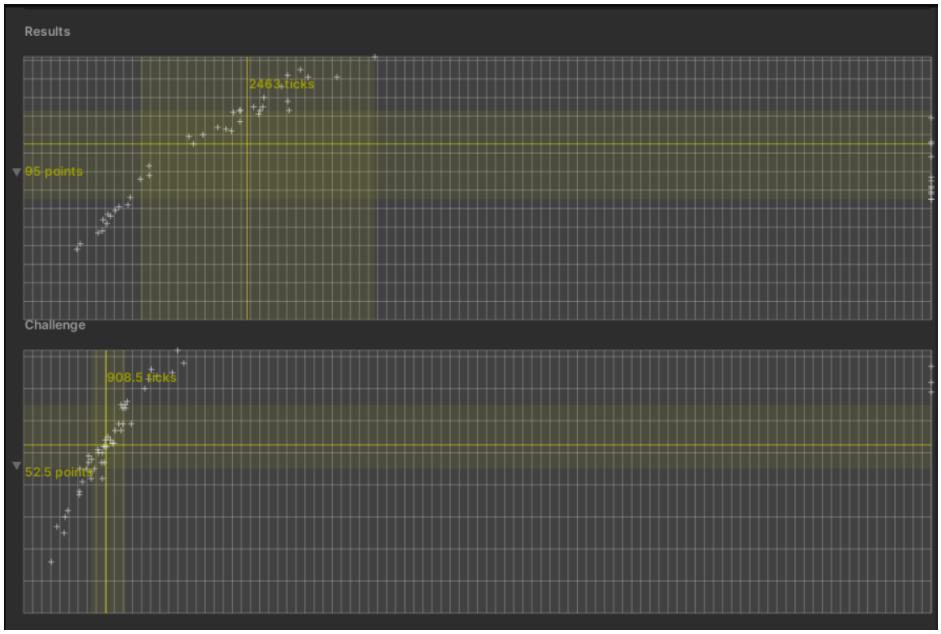
I added an or to the condition and then checked if there is an obstacle ahead and if the tail and food are reachable. Now if there is an obstacle ahead and the snake has access to both tail and food it should move to the tail allowing it to live longer.

Then I added on to the filter that lets the snake go to the food. To the condition of the filter, I added an and to check if there are no obstacles ahead and if it returns true then it should go for the food. This makes sure that it is safe to go for the food.

```
new Filter(
    new Condition(() => AreTailAndFoodReachable() && !Snake.IsObstacleAhead()),
    new Action(Snake.MoveTowardsFood)
),
```

(Adding to the condition to check if food is safe)

With these changes to my code my snake's performance from a test of 50 was:



(Results from a test of 50 in iteration 4)

My snake's performance in the normal snake game had an increase in its average score compared to the results from iteration 3 as it went up by 7 points on average and it lived for a lot longer than the snake ai in iteration 3. However, there was a slight decrease in the challenge since it went down by 3 points compared to the results in iteration 3. Overall, I believe this snake ai is better than the snake ai in iteration 3 since it has had a better impact on the snake's performance as it had an increase of 7 points in the normal snake game but a small decrease in the challenge by 3 points which is not that significant. However, in the results from iteration 4 we can see that it is going into infinite loops causing it to die once the game hits max ticks which is a problem that I need to solve.

Iteration 5

During this iteration I made some changes to existing filters and added an action to occur if no filter's condition is satisfied.

First, within the filter that has the condition that checks if the food is reachable, but tail is not I removed the raycast ahead in the if statement. This is because when I was testing my code it seemed like it was causing the snake to die more than helping the snake to live.

```
if (Snake.IsReachable(new Vector2Int(i, j)) && Snake.RaycastAhead().Distance > 1)
```

(Code before the change)

```
if (Snake.IsReachable(new Vector2Int(i, j)))
```

(Code after the change)

Then I lowered the amount of food the snake eats whilst it does pathfinding if the food is reachable and the snake does not care if its tail is reachable. I lowered it to less than or equal to 55 from 65 to see if it performed better. As it can start to perform the code that lets it live longer sooner rather than later since it can reach the required size faster.

```
new Filter(
    new Condition(() => Snake.Body.Count <= 55 && Snake.IsFoodReachable()),
    new Action(Snake.MoveTowardsFood)
),
```

(Changed the snake body count size)

Then to stop the infinite loop I added on to the condition which lets the snake move to its tail since I believe the loop occurs once the snake starts to follow its tail and can't stop following its tail.

```
new Filter(
    new Condition(() => (IsTailOnlyReachable() || Snake.IsObstacleAhead() && AreTailAndFoodReachable() && Vector2Int.Distance(Snake.TailPosition, Snake.HeadPosition) > 3)),

    new Action(() => Snake.MoveTowards(
        Snake
        .AvailableNeighbours(Snake.TailPosition)
        .FirstOrDefault(position => Snake.IsReachable(position)))
    )
),
```

(Code added to stop infinite loop)

The code Vector2Int.Distance(Snake.TailPosition, Snake.HeadPosition) > 3) should check if the tails distance to the head is greater than 3 if it is less than 3 it means that the tail and the head are following each other which can cause an infinite loop so it would stop the head from following the tail.

Then I added a size checker condition to the filter which allows the snake to go to the food. So, if the tail and food are available and there are no obstacles, and the size is greater than 50 then the snake would move towards the food.

```
new Filter(
    new Condition(() => AreTailAndFoodReachable() && Snake.Body.Count > 55 && !Snake.IsObstacleAhead()),
    new Action(Snake.MoveTowardsFood)
),
```

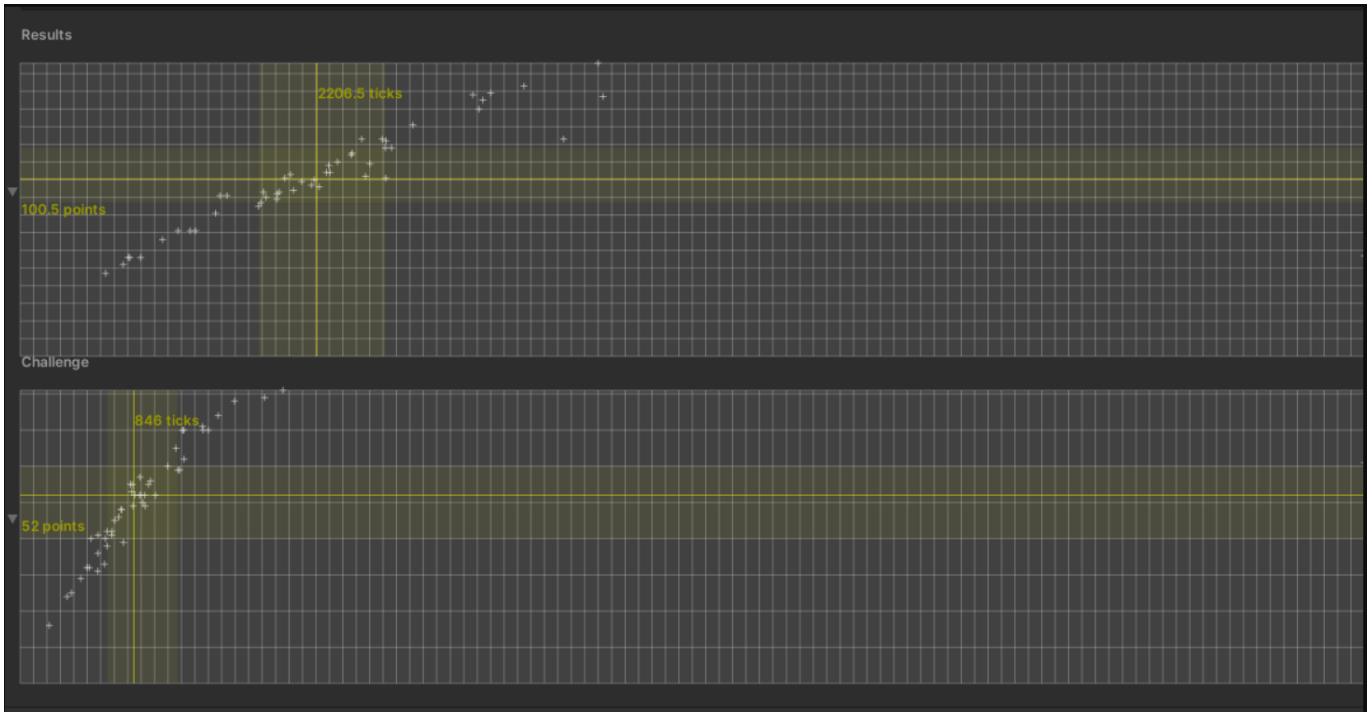
(Changes made to second condition that lets snake move to food)

Finally in the selector I added an action which would be performed if none of the filter's conditions are met the action lets the snake follow its body and since this can cause a loop of the snake following its own body, I added a random number generator that generates type float between 0-1 and if it is greater than 0.6 then it would stop moving towards its body and would go to an available corner on the grid.

```
new Action(() =>
{
    var current = Snake.Body.Last;
    while (current != null)
    {
        foreach (var b in Snake.AvailableNeighbours(current.Value))
        {
            if (Snake.IsReachable(b) && Random.Range(0f, 1f) < 0.6)
            {
                Snake.MoveTowards(b);
                return;
            }
            else if (Snake.IsReachable(new Vector2Int(1, 1)))
            {
                Snake.MoveTowards(1, 1);
                return;
            }
            else if (Snake.IsReachable(new Vector2Int(1, 24)))
            {
                Snake.MoveTowards(1, 24);
                return;
            }
            else if (Snake.IsReachable(new Vector2Int(24, 1)))
            {
                Snake.MoveTowards(24, 1);
                return;
            }
            else if (Snake.IsReachable(new Vector2Int(24, 24)))
            {
                Snake.MoveTowards(24, 24);
                return;
            }
        }
        current = current.Previous;
    }
})
```

(Action to be performed by snake if no filter's condition is met)

With these changes to my code my snake's performance from a test of 50 was:



(Results from a test of 50 in iteration 5)

My snake's performance in the normal snake game went up to 100.5 which is an increase of 5.5 from iteration 4 which shows that it is performing better. However, the snake ai in the challenge has had a decrease of 0.5 compared to iteration 4 which is not a significant drop showing that the changes made were more beneficial meaning that the snake ai from this iteration is better than the snake ai from iteration 4.

Iteration 6

During this iteration I made some changes to existing filters.

I first made a Boolean function which checked if the food was safe to get to by checking the obstacles around the head and if the obstacles were greater than 2 that meant that the food is not safe and the Boolean would return false.

```
bool FOODSAFE()
{
    int unReachable = 0;
    foreach (var p in Snake.AvailableNeighbours(Snake.HeadPosition))
    {
        if (Snake.IsObstacle(p))
        {
            unReachable++;
        }
    }
    return unReachable <= 2;
}
```

(Boolean function for FOODSAFE())

Then I added to the conditions that control if the snake can go get the food by adding FOODSAFE(). This is so it only goes to the food if the head has less than or equal to 2 obstacles around it. I also decided to lower the size which the snake can eat without worrying about if the tail and the food are reachable at the same time to 50. I made this change as the snake was still dying early and was unable to meet the conditions in the other filters as it did not have the required points so by lowering it, it can potentially perform better as it can get to the size that is required to perform other actions as it would meet the requirements for their conditions sooner.

```
new Filter(
    new Condition(() => Snake.Body.Count <= 50 && Snake.IsFoodReachable() && FOODSAFE()),
    new Action(Snake.MoveTowardsFood)
),
```

(FOODSAFE() Added to condition)

```
new Filter(
    new Condition(() => AreTailAndFoodReachable() && Snake.Body.Count > 50 && FOODSAFE()),
    new Action(Snake.MoveTowardsFood)
),
```

(FOODSAFE() Added to condition)

Next, I added the condition that lets the snake follow the tail. To the condition I added an or that checks if food is not safe by calling !FOODSAFE() and it also checks if tail and food are

reachable and that the tail is more than 3 spaces away from the head this is, so it goes to the tail if food is not safe but does not get in an infinite loop of following its tail.

```
new Filter(
    new Condition(() => (IsTailOnlyReachable() ||
    Snake.IsObstacleAhead() && AreTailAndFoodReachable() && Vector2Int.Distance(Snake.TailPosition, Snake.HeadPosition) > 3) ||
    !FOODSAFE() && AreTailAndFoodReachable() && Vector2Int.Distance(Snake.TailPosition, Snake.HeadPosition) > 3),

    new Action(() => Snake.MoveTowards(
        Snake
        .AvailableNeighbours(Snake.TailPosition)
        .FirstOrDefault(position => Snake.IsReachable(position)))
    )
),
```

(Changes made to the condition that controls if snake movement to the tail)

Finally in this iteration I realised the action that the snake performs if it did not satisfy any filter's conditions was not performing as I wanted it to. This is because once the snake stops following its body it would automatically go to position 1,1 on the grid and would not check the other corners. So, I decided to change it so it would go to the corner that is farthest and reachable from its head position. I did this by getting the distance from the head to each corner on the grid and checking if they are reachable and the one that has the largest distance would be the area that the snake would move to. This is because that area is most likely to have most space since it is the farthest area from the head.

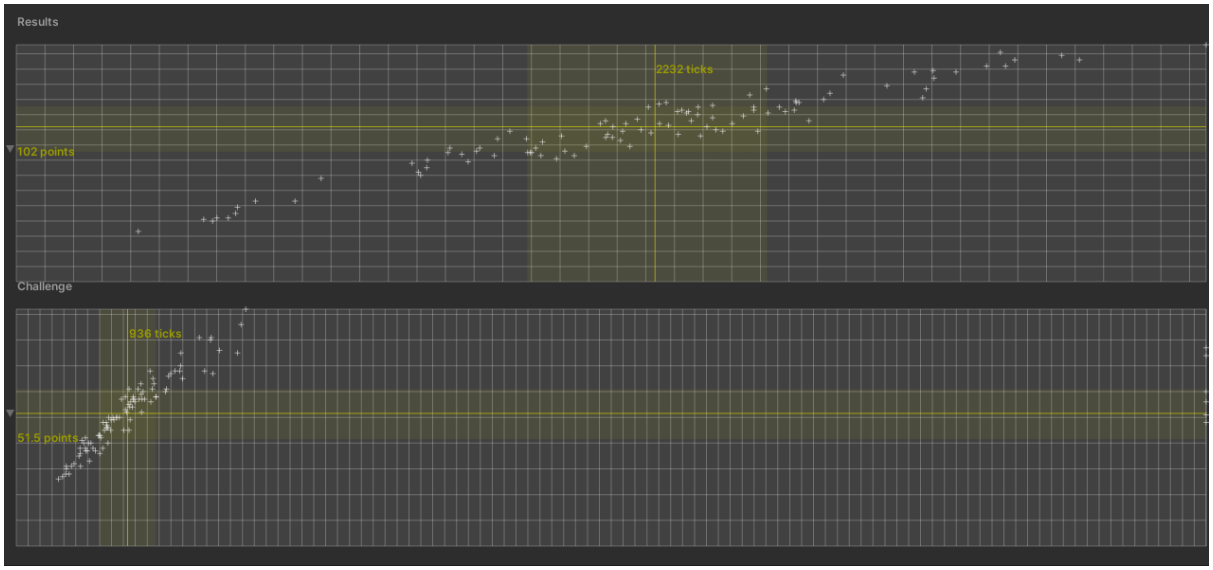
```
new Action(() =>
{
    int HEADTOBOTTOMLEFT = Snake.DistanceFrom(Snake.HeadPosition, new Vector2Int(1, 1));
    int HEADTOBOTTOMRIGHT = Snake.DistanceFrom(Snake.HeadPosition, new Vector2Int(1, 24));
    int HEADTOTOPRIGHT = Snake.DistanceFrom(Snake.HeadPosition, new Vector2Int(24, 24));
    int HEADTOTOPLEFT = Snake.DistanceFrom(Snake.HeadPosition, new Vector2Int(24, 1));

    if (!Snake.IsReachable(new Vector2Int(1, 1)))
    {
        HEADTOBOTTOMLEFT = 0;
    }
    if (!Snake.IsReachable(new Vector2Int(1, 24)))
    {
        HEADTOBOTTOMRIGHT = 0;
    }
    if (!Snake.IsReachable(new Vector2Int(24, 24)))
    {
        HEADTOTOPRIGHT = 0;
    }
    if (!Snake.IsReachable(new Vector2Int(24, 1)))
    {
        HEADTOTOPLEFT = 0;
    }

    if (max < HEADTOBOTTOMLEFT)
    {

        max = HEADTOBOTTOMLEFT;
    }
    else if (max < HEADTOBOTTOMRIGHT)
    {
        max = HEADTOBOTTOMRIGHT;
    }
    else if (max < HEADTOTOPRIGHT)
    {
        max = HEADTOTOPRIGHT;
    }
    else if (max < HEADTOTOPLEFT)
    {
        max = HEADTOTOPLEFT;
    }
```

(Updated code for action if no condition is met)

```
    var current = Snake.Body.Last;
    while (current != null)
    {
        foreach (var b in Snake.AvailableNeighbours(current.Value))
        {
            if (Snake.IsReachable(b) && Random.Range(0f, 1f) < 0.6)
            {
                Snake.MoveTowards(b);
                return;
            }
        }
        if (max == HEADTOBOTTOMLEFT)
        {
            Snake.MoveTowards(new Vector2Int(1, 1));
        }
        else if (max == HEADTOBOTTOMRIGHT)
        {
            Snake.MoveTowards(new Vector2Int(1, 24));
        }
        else if (max == HEADTOTOPRIGHT)
        {
            Snake.MoveTowards(new Vector2Int(24, 24));
        }
        else if (max == HEADTOTOPLEFT)
        {
            Snake.MoveTowards(new Vector2Int(24, 1));
        }
        current = current.Previous;
    }
})
```

(Updated code for action if no condition is met)

With these changes to my code my snake's performance from a test of 50 was:



(Results from a test of 50 in iteration 5)

My snake's performance has increased in the normal snake game as it is collecting more food on average by 1.5 and it is living for longer periods than the snake ai in iteration 5 by 25.5 more ticks on average showing that the changes in this iteration are beneficial. However there has been a 0.5 decrease in the average score of the challenge but with this new snake ai the snake is living for a longer period than the snake ai in iteration 5 by 90 more ticks on average. However, this may be due to the infinite loops which are visible in

the output of the challenge graph. Next, I would try to find a solution to stop these infinite loops and find a way to make the challenge perform better without lowering my average for the normal snake game.

Final Iteration

During my final iteration I made some changes to existing filters and added a new filter to my behaviour tree code.

First, I decided to change my FoodSafe() function, so it checks around the food position instead. So, it only goes to the food if the food is in a safe position, and I also changed it so it checks if there are 2 or more obstacles around and if there were it would return false.

```
//checks if food is safe by comparing number of obstacles around food and if it is greater than or equal to 2 then it returns false
bool FoodSafe()
{
    int unReachable = 0;
    foreach (var p in Snake.AvailableNeighbours(Snake.FoodPosition))
    {
        if (Snake.IsObstacle(p))
        {
            unReachable++;
        }
    }
    return unReachable < 2;
}
```

(Boolean FoodSafe() changed)

Then I decided to create a new Boolean function that checked if the head position is safe, I done this by using a similar code to the one I used for FoodSafe(), and it checks if there are more than one obstacle around the head if there is more than one obstacle around the head it would return false.

```
//checks if head is safe by comparing number of obstacles around head and if it is greater than 2 then it returns false
bool HeadSafe()
{
    int unReachable = 0;
    foreach (var p in Snake.AvailableNeighbours(Snake.HeadPosition))
    {
        if (Snake.IsObstacle(p))
        {
            unReachable++;
        }
    }
    return unReachable <= 1;
}
```

(New Boolean function called HeadSafe() was made)

I also added a random.range(0f,1f) < 0.9 to my filter that let the snake follow its tail since it was still going into infinite loops.

```
new Filter(
    //condition checks if:
    //tail is reachable but food is not or
    //Obstacle is ahead and food and tail are reachable and tail and head have a distance greater than 3 or
    //food is not safe and tail and food are reachable and tail and head have a distacne greater than 3
    //random range to make condition false so it stops infinte loops.
    new Condition(() => ((IsTailOnlyReachable() ||
    Snake.IsObstacleAhead() && AreTailAndFoodReachable() && Vector2Int.Distance(Snake.TailPosition, Snake.HeadPosition) > 3) ||
    !FoodSafe() && AreTailAndFoodReachable() && Vector2Int.Distance(Snake.TailPosition, Snake.HeadPosition) > 3) && Random.Range(0f, 1f) < 0.9),

    //if any of the conditions in the condition is met then the snake would move towards its tail
    new Action(() => Snake.MoveTowards(
        Snake
        .AvailableNeighbours(Snake.TailPosition)
        .FirstOrDefault(position => Snake.IsReachable(position)))
    )
),
```

(random.range(0f,1f) < 0.9 added to condition)

Then I created a new filter that has a condition which checks if the head is not safe. If the head is not safe it would calculate the distance to the closest obstacle from the right of the head to the left and would save them to an int variable. Then I compare the distances and take the one that has the largest distance away from the obstacle and check if the obstacle is reachable then it would turn to the side that has the farthest obstacle. For example, if the obstacle on the right of the head was further than the left of the head and the obstacle to the right is reachable then the snake would turn right.

```
new Filter(
    //condition checks if snake head is not safe
    new Condition(() => !HeadSafe()),
    new Action(() =>
    {
        //finds distance of the closest obstacle to the left of the snake from its head position and saves it to HeadToLeft
        int HeadToLeft = Snake.DistanceFrom(Snake.HeadPosition, Snake.RaycastLeft().Position);
        // finds distance of the closest obstacle to the right of the snake from its head position and saves it to HeadToRight
        int HeadToRight = Snake.DistanceFrom(Snake.HeadPosition, Snake.RaycastRight().Position);

        //checks if left obstacle is further away from right obstacle and checks if the snake can reach the left obstacle
        if (HeadToLeft > HeadToRight && Snake.IsReachable(Snake.RaycastLeft().Position))
        {
            //snake turns towards the left obstacle
            Snake.TurnLeft();
        }
        //checks if right obstacle is further away from left obstacle and checks if the snake can reach the right obstacle
        else if (HeadToLeft < HeadToRight && Snake.IsReachable(Snake.RaycastRight().Position))
        {
            //snake turns towards the right obstacle
            Snake.TurnRight();
        }

    })
),
```

(New Filter that has condition if head is not safe with an action has been added)

Then I changed the filter that handles random movement and linear movement (move cell by cell). I changed the condition in the random movement, so it only happens when the snake has a size less than 60 and no food or tail is reachable. This is because once the snake was too large the random movement would cause it to die when the condition for it was met.

```
new Filter(
    new Condition(() =>
    {
        //checks if food and tail are not reachable and checks that the snake size is less than 60
        if (FoodNotReachableAndTail() && Snake.Body.Count < 60)
        {
            //creates new list with type vector2Int
            List<Vector2Int> list = new List<Vector2Int>();
            //for loop from 0 to 25 (not including)
            for (int i = 0; i < 25; i++)
            {
                //for loop from 0 to 25 (not including)
                for (int j = 0; j < 25; j++)
                {
                    //checks if i,j is equal to the head position
                    if (new Vector2Int(i, j) == Snake.HeadPosition)
                    {
                        continue;
                    }
                    //checks if i,j is an obstacle on the grid
                    if (Snake.IsObstacle(new Vector2Int(i, j)))
                    {
                        continue;
                    }
                    //adds i,j to the list
                    list.Add(new Vector2Int(i, j));
                }
            }

            //shuffle the list
            for (int i = 0; i < list.Count; i++)
            {
                //int j is a random point within the list
                int j = Random.Range(0, list.Count);
                //swaps elements in index i with elements in index j of the list
                Vector2Int temp = list[i];
                list[i] = list[j];
                list[j] = temp;

            }
            foreach (Vector2Int p in list)
            {
                //checks if snake can reach p
                if (Snake.IsReachable(p))
                {
                    //sets x to p.x
                    x = p.x;
                    //sets y to p.y
                    y = p.y;
                    return true;
                }
            }
        }
        return false;
    }),
    new Action(() =>
    {
        //if condition returns true then snake moves to x,y
        Snake.MoveTowards(x, y);
    })
),
```

(Condition changed so it only works when snake size is less than 60)

Then I changed the condition in the filter that handles linear movement (move cell by cell), so it only happens if food is reachable, and tail is not, and head is safe or neither tail and food are reachable, and the snake size is greater than or equal to 60 and the head is safe.

```
//just live until tail is reachable
new Filter(
    //condition checks if
    //food is reachable and tail is not or
    //food and tail are not reachable and snake size greater than or equal to 60 and head is safe
    new Condition(() => (FoodReachableButNotTail() || FoodNotReachableAndTail() && Snake.Body.Count >= 60) && HeadSafe()),
    new Filter(
        new Condition(() =>
        {
            //for loop goes from 0 to 25 (not including)
            for (int i = 0; i < 25; i++)
            {
                //for loop goes from 0 to 25 (not including)
                for (int j = 0; j < 25; j++)
                {
                    //checks if i,j is equal to the head position
                    if (new Vector2Int(i, j) == Snake.HeadPosition)
                    {
                        continue;
                    }
                    //checks if i,j is an obstacle on the grid
                    if (Snake.IsObstacle(new Vector2Int(i, j)))
                    {
                        continue;
                    }
                    //checks if snake can reach i,j
                    if (Snake.IsReachable(new Vector2Int(i, j)))
                    {
                        //set k to i
                        k = i;
                        //set l to j
                        l = j;
                        return true;

                    }
                }
            }
            return false;

        }),
        new Action(() => {
            //if condition returns true then snake moves to k,l
            Snake.MoveTowards(k, l);
        })
    )
),
```

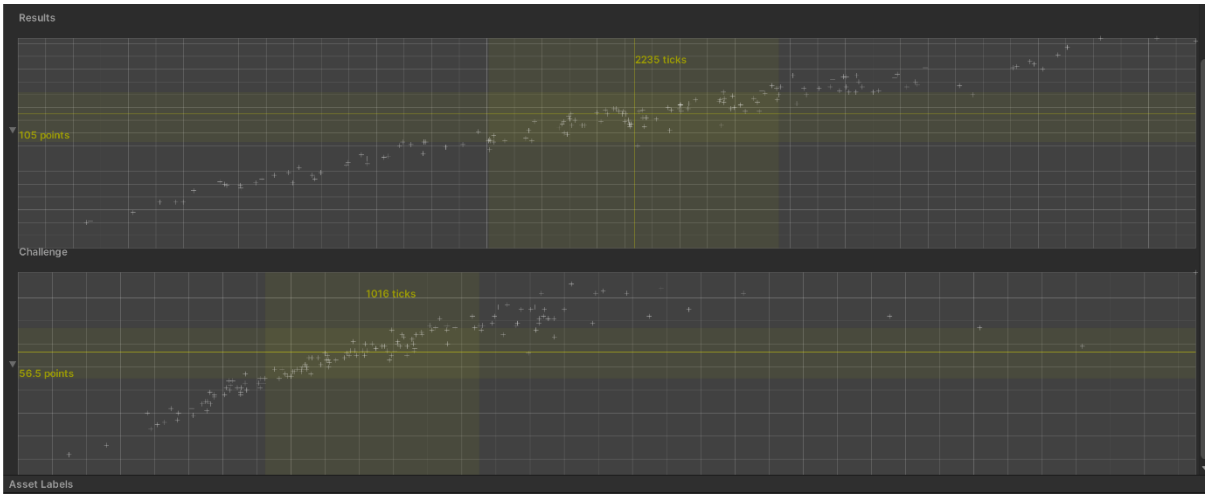
(Condition Changed for linear movement)

Finally, I had an error in my action that occurs if no condition is met where the snake would still go to position 1,1 this was because I needed to use if else statements instead of only if statements when checking if the snake was able to reach a corner.

```
//checks if the snake can not reach the bottom left corner
if (!Snake.IsReachable(new Vector2Int(1, 1)))
{
    //if snake can not reach bottom left it sets HeadToBottomLeft to 0;
    HeadToBottomLeft = 0;
}
//checks if the snake can not reach the bottom right corner
else if (!Snake.IsReachable(new Vector2Int(1, 24)))
{
    //if snake can not reach bottom left it sets HeadToBottomRight to 0;
    HeadToBottomRight = 0;
}
//checks if the snake can not reach the top right corner
else if (!Snake.IsReachable(new Vector2Int(24, 24)))
{
    //if snake can not reach top right it sets HeadToTopRight to 0;
    HeadToTopRight = 0;
}
//checks if the snake can not reach the top Left corner
else if (!Snake.IsReachable(new Vector2Int(24, 1)))
{
    //if snake can not reach top right it sets HeadToTopLeft to 0;
    HeadToTopLeft = 0;
}
```

(Change made to stop snake from only going to one corner)

With these changes to my code my snake's performance from a test of 150 was:

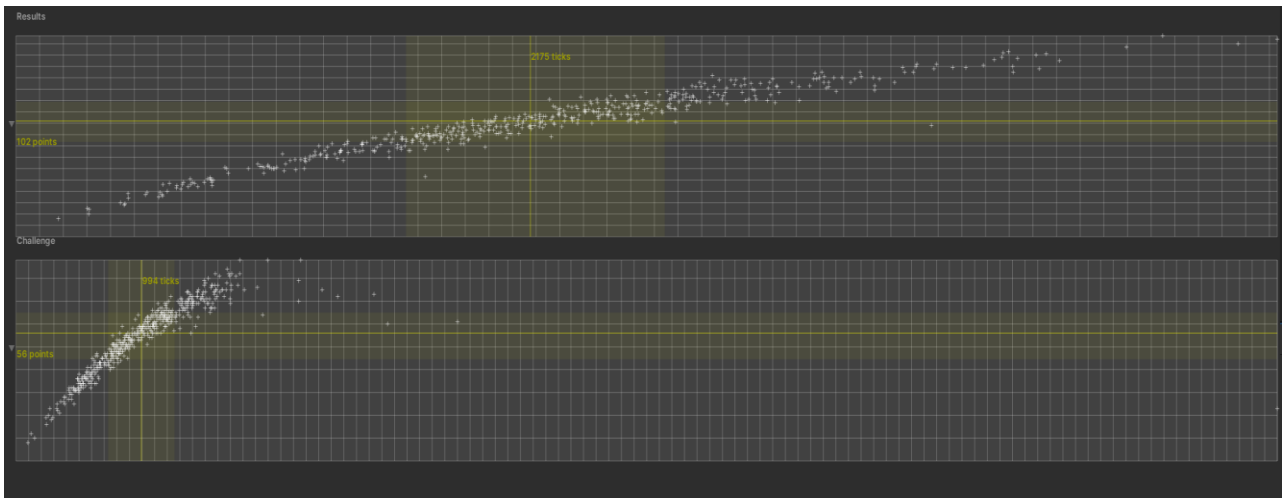


(Results from a test of 150 in final iteration)

This output shows that my snake was performing better in the normal snake game as it averaged 105 points in 150 tests. The results show that the snake is more consistent since the test was a larger sample making this output more reliable. Also, the snake performed better in the challenge since it had an average of 56.5 points which is 5 higher than the snake ai in iteration 6. Also, we can see no infinite loops have occurred during the 150 tests meaning that the problem of getting infinite loops has been solved. So overall this snake ai is performing better than the snake ai in iteration 6.

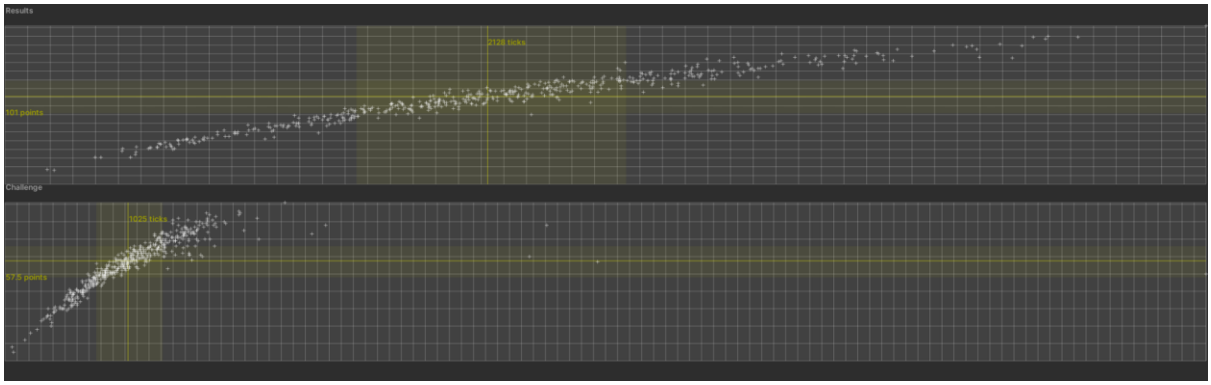
Final Iteration - Test Of 500

First test of 500 for final snake ai:



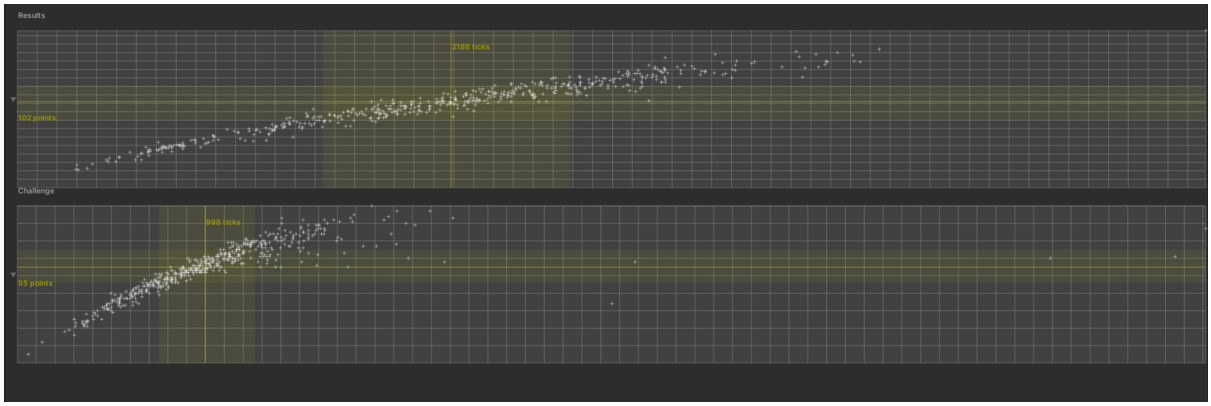
(Results from a test of 500 in final iteration)

Second Test of 500 for final snake ai:



(Results from a test of 500 in final iteration)

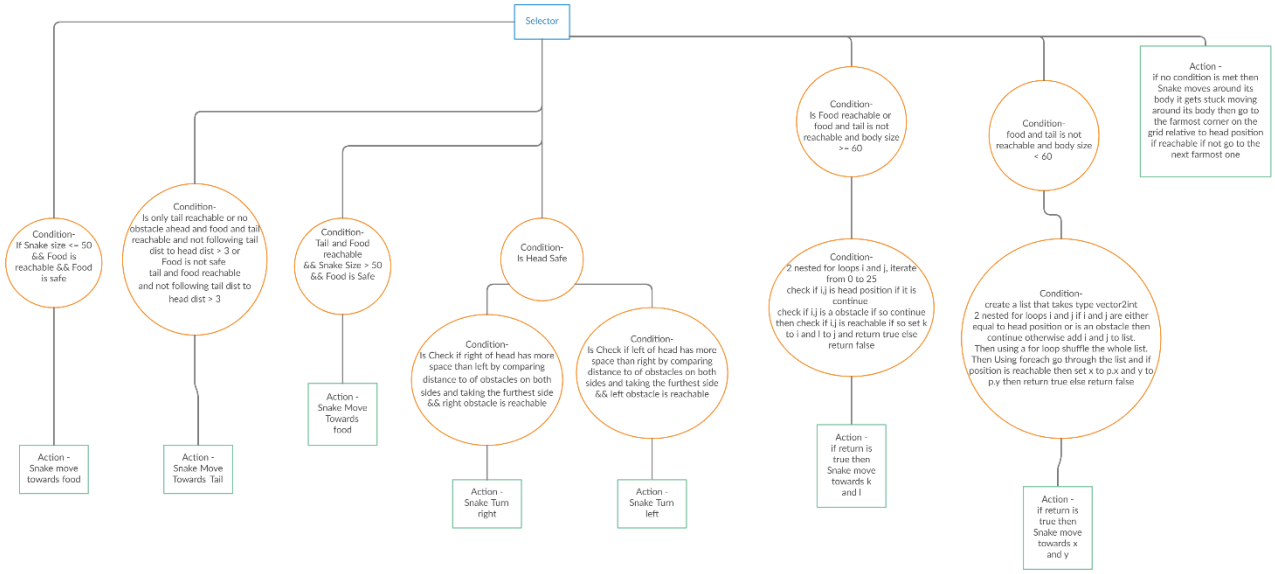
Third test of 500 for final snake ai:



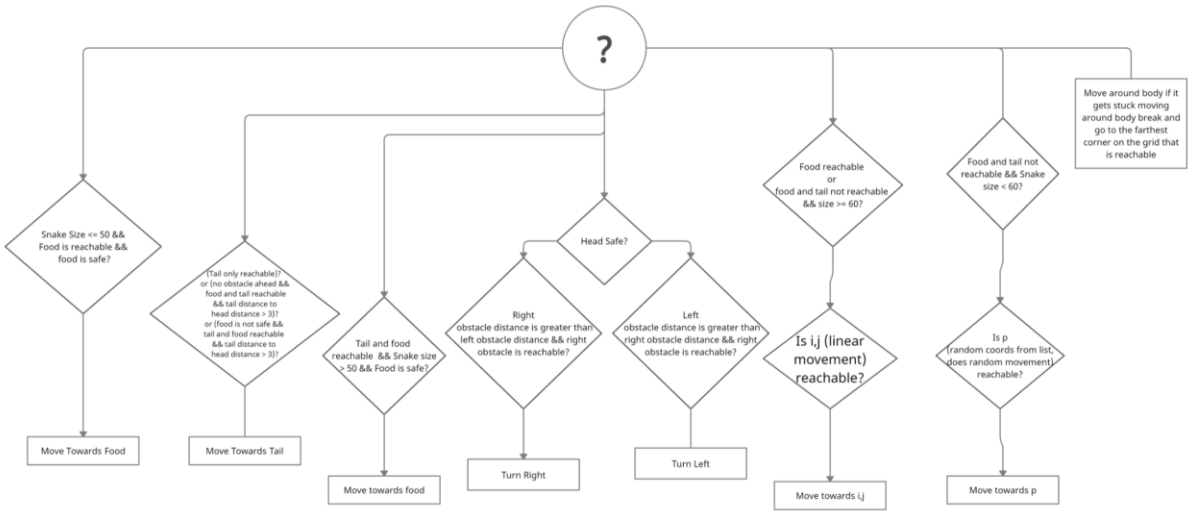
(Results from a test of 500 in final iteration)

Behaviour Tree Diagrams

Behaviour tree of the code for final iteration of snake ai



(First Behaviour Tree for final iteration of snake ai)



(Second Behaviour Tree for final iteration of snake ai)

Behaviour Tree Code

```
using UnityEngine;
using AlanZucconi.AI.BT;
using AlanZucconi.Snake;
using System.Linq;
using System.Collections.Generic;

[CreateAssetMenu
(
    fileName = "SnakeAI_oguve001",
    menuName = "Snake/2021-22/SnakeAI_oguve001"
)]
public class SnakeAI_oguve001 : SnakeAI
{

    //intialise variables
    int x;
    int y;

    int k;
    int l;

    int max;



    public override Node CreateBehaviourTree(SnakeGame Snake)
    {
        //checks if food is reachable but tail is not
        bool FoodReachableButNotTail()
        {
```

```
    return !Snake
        .AvailableNeighbours(Snake.TailPosition)
        .Any(position => Snake.IsReachable(position))
        && Snake.IsFoodReachable();
}
//checks if tail and food are reachable
bool AreTailAndFoodReachable()
{
    return Snake
        .AvailableNeighbours(Snake.TailPosition)
        .Any(position => Snake.IsReachable(position))
        && Snake.IsFoodReachable();
}
//checks if tail is only reachable and food is not
bool IsTailOnlyReachable()
{
    return Snake
        .AvailableNeighbours(Snake.TailPosition)
        .Any(position => Snake.IsReachable(position))
        && !Snake.IsFoodReachable();
}

//checks if food and tail are not reachable
bool FoodNotReachableAndTail()
{
    return (!Snake
        .AvailableNeighbours(Snake.TailPosition)
        .Any(position => Snake.IsReachable(position)))
        && (!Snake.IsFoodReachable());
}
```

```
    //checks if food is safe by comparing number of obstacles around food and if it is greater than or equal to 2 then it returns false
    bool FoodSafe()
    {
       int unReachable = 0;
       foreach (var p in Snake.AvailableNeighbours(Snake.FoodPosition))
       {
          if (Snake.IsObstacle(p))
          {
             unReachable++;
          }
       }
       return unReachable < 2;
    }

    //checks if head is safe by comparing number of obstacles around head and if it is greater than 2 then it returns false
    bool HeadSafe()
    {
       int unReachable = 0;
       foreach (var p in Snake.AvailableNeighbours(Snake.HeadPosition))
       {
          if (Snake.IsObstacle(p))
          {
             unReachable++;
          }
       }
       return unReachable <= 1;
    }
```

```
    return
      //selector initialised
      new Selector(
        new Filter(
          //condition checks if snake size less than or equal to 50 and if food is reachable and safe
          new Condition(() => Snake.Body.Count <= 50 && Snake.IsFoodReachable() &&
FoodSafe()),
          //if condition is met then the snake moves towards food
          new Action(Snake.MoveTowardsFood)
        ),
        new Filter(
          //condition checks if:
          //tail is reachable but food is not or
          //Obstacle is ahead and food and tail are reachable and tail and head have a distance
greater than 3 or
          //food is not safe and tail and food are reachable and tail and head have a distacne
greater than 3
          //random range to make condition false so it stops infinte loops.
          new Condition(() => ((IsTailOnlyReachable() ||
          Snake.IsObstacleAhead() && AreTailAndFoodReachable() &&
Vector2Int.Distance(Snake.TailPosition, Snake.HeadPosition) > 3) ||
          !FoodSafe() && AreTailAndFoodReachable() && Vector2Int.Distance(Snake.TailPosition,
Snake.HeadPosition) > 3) && Random.Range(0f, 1f) < 0.9),

          //if any of the conditions in the condition is met then the snake would move towards its
tail
          new Action(() => Snake.MoveTowards(
            Snake
            .AvailableNeighbours(Snake.TailPosition)
            .FirstOrDefault(position => Snake.IsReachable(position)))
          )
        ),
```

```
        new Filter(
            //condtion checks if tail and food are reachable and snake size is greater than 50 and food is in a safe position
            new Condition(() => AreTailAndFoodReachable() && Snake.Body.Count > 50 && FoodSafe()),
            //if condition met then the snake would move towards the food
            new Action(Snake.MoveTowardsFood)
        ),

        new Filter(
            //condition checks if snake head is not safe
            new Condition(() => !HeadSafe()),
            new Action(() =>
            {
                //finds distance of the closest obstacle to the left of the snake from its head position and saves it to HeadToLeft
                int HeadToLeft = Snake.DistanceFrom(Snake.HeadPosition, Snake.RaycastLeft().Position);
                // finds distance of the closest obstacle to the right of the snake from its head position and saves it to HeadToRight
                int HeadToRight = Snake.DistanceFrom(Snake.HeadPosition, Snake.RaycastRight().Position);

                //checks if left obstacle is further away from right obstacle and checks if the snake can reach the left obstacle
                if (HeadToLeft > HeadToRight && Snake.IsReachable(Snake.RaycastLeft().Position))
                {
                    //snake turns towards the left obstacle
                    Snake.TurnLeft();
                }
                //checks if right obstacle is further away from left obstacle and checks if the snake can reach the right obstacle
                else if (HeadToLeft < HeadToRight && Snake.IsReachable(Snake.RaycastRight().Position))
```

```
                {
                    //snake turns towards the right obstacle
                    Snake.TurnRight();
                }

            })
        ),

        //just live until tail is reachable
        new Filter(
            //condition checks if
            //food is reachable and tail is not or
            //food and tail are not reachable and snake size greater than or equal to 60 and head is safe
            new Condition(() => (FoodReachableButNotTail() || FoodNotReachableAndTail() && Snake.Body.Count >= 60) && HeadSafe()),
            new Filter(
                new Condition(() =>
                {
                    //for loop goes from 0 to 25 (not including)
                    for (int i = 0; i < 25; i++)
                    {
                        //for loop goes from 0 to 25 (not including)
                        for (int j = 0; j < 25; j++)
                        {
                            //checks if i,j is equal to the head position
                            if (new Vector2Int(i, j) == Snake.HeadPosition)
                            {
                                continue;
                            }
                            //checks if i,j is an obstacle on the grid
                            if (Snake.IsObstacle(new Vector2Int(i, j)))
```

```
                {
                    continue;
                }
                //checks if snake can reach i,j
                if (Snake.IsReachable(new Vector2Int(i, j)))
                {
                    //set k to i
                    k = i;
                    //set l to j
                    l = j;
                    return true;

                }
            }
        }
        return false;

    }),
    new Action(() => {
        //if condition returns true then snake moves to k,l
        Snake.MoveTowards(k, l);
    })


  )
),

new Filter(
  new Condition(() =>
  {
    //checks if food and tail are not reachable and checks that the snake size is less than 60
```

```
if (FoodNotReachableAndTail() && Snake.Body.Count < 60)
{
    //creates new list with type vector2Int
    List<Vector2Int> list = new List<Vector2Int>();
    //for loop from 0 to 25 (not including)
    for (int i = 0; i < 25; i++)
    {
        //for loop from 0 to 25 (not including)
        for (int j = 0; j < 25; j++)
        {
            //checks if i,j is equal to the head position
            if (new Vector2Int(i, j) == Snake.HeadPosition)
            {
                continue;
            }
            //checks if i,j is an obstacle on the grid
            if (Snake.IsObstacle(new Vector2Int(i, j)))
            {
                continue;
            }
            //adds i,j to the list
            list.Add(new Vector2Int(i, j));
        }
    }

    //shuffle the list
    for (int i = 0; i < list.Count; i++)
    {
        //int j is a random point within the list
        int j = Random.Range(0, list.Count);
        //swaps elements in index i with elements in index j of the list
```

```
            Vector2Int temp = list[i];
            list[i] = list[j];
            list[j] = temp;

        }
        foreach (Vector2Int p in list)
        {
            //checks if snake can reach p
            if (Snake.IsReachable(p))
            {
                //sets x to p.x
                x = p.x;
                //sets y to p.y
                y = p.y;
                return true;
            }
        }
      }
      return false;
   }),

   new Action(() =>
   {
      //if condition returns true then snake moves to x,y
      Snake.MoveTowards(x, y);
   })
),

//action is none of the conditions are met
new Action(() =>
{
```

```
        //gets distance of the head to the bottom left corner of the grid and saves it to
HeadToBottomLeft
        int HeadToBottomLeft = Snake.DistanceFrom(Snake.HeadPosition, new Vector2Int(1, 1));

        //gets distance of the head to the bottom right corner of the grid and saves it to
HeadToBottomRight
        int HeadToBottomRight = Snake.DistanceFrom(Snake.HeadPosition, new Vector2Int(1,
24));

        //gets distance of the head to the Top right corner of the grid and saves it to
HeadToTopRight
        int HeadToTopRight = Snake.DistanceFrom(Snake.HeadPosition, new Vector2Int(24, 24));

        //gets distance of the head to the Top left corner of the grid and saves it to
HeadToTopLeft
        int HeadToTopLeft = Snake.DistanceFrom(Snake.HeadPosition, new Vector2Int(24, 1));

        //checks if the snake can not reach the bottom left corner
        if (!Snake.IsReachable(new Vector2Int(1, 1)))
        {
           //if snake can not reach bottom left it sets HeadToBottomLeft to 0;
           HeadToBottomLeft = 0;
        }
        //checks if the snake can not reach the bottom right corner
        else if (!Snake.IsReachable(new Vector2Int(1, 24)))
        {
           //if snake can not reach bottom left it sets HeadToBottomRight to 0;
           HeadToBottomRight = 0;
        }
        //checks if the snake can not reach the top right corner
        else if (!Snake.IsReachable(new Vector2Int(24, 24)))
        {
           //if snake can not reach top right it sets HeadToTopRight to 0;
```

```
    HeadToTopRight = 0;
}
//checks if the snake can not reach the top Left corner
else if (!Snake.IsReachable(new Vector2Int(24, 1)))
{
    //if snake can not reach top right it sets HeadToTopLeft to 0;
    HeadToTopLeft = 0;
}

//sets max to be the distance of the far most corner from the head
if (max < HeadToBottomLeft)
{
    max = HeadToBottomLeft;
}
if (max < HeadToBottomRight)
{
    max = HeadToBottomRight;
}
if (max < HeadToTopRight)
{
    max = HeadToTopRight;
}
if (max < HeadToTopLeft)
{
    max = HeadToTopLeft;
}

var current = Snake.Body.Last;
while (current != null)
{
    //b is the neightbours of current.value
```

```
            foreach (var b in Snake.AvailableNeighbours(current.Value))
            {
               //checks if b is reachable and has Random.Range(0f, 1f) < 0.6 to stop infinte loop of snake following body
               if (Snake.IsReachable(b) && Random.Range(0f, 1f) < 0.6)
               {
                  //snake moves towards b
                  Snake.MoveTowards(b);
                  return;
               }
            }
            current = current.Previous;

            //checks which distance max is equal to and would move to that corner which is the far most reachable corner.
            if (max == HeadToBottomLeft)
            {
               Snake.MoveTowards(new Vector2Int(1, 1));
            }
            else if (max == HeadToBottomRight)
            {
               Snake.MoveTowards(new Vector2Int(1, 24));
            }
            else if (max == HeadToTopRight)
            {
               Snake.MoveTowards(new Vector2Int(24, 24));
            }
            else if (max == HeadToTopLeft)
            {
               Snake.MoveTowards(new Vector2Int(24, 1));
            }
```

```
            }
          })
        );
    }
}
```

Conclusion

Overall, during this project, I have developed a well-functioning snake ai that gives consistent results and performs well. I have successfully shown the development of my snake ai which is visible from iteration 1 to the final iteration of my snake ai. Furthermore, I believe that I have a well thought out behaviour tree which has allowed me to develop my snake ai to get the results that I am getting now. Finally, I believe I optimised the score required for the snake to change behaviour so it can perform the actions in other filters which can help it live longer and not die. However, I would have liked to work more on the snake's behaviour after it hit a large score for example 115 so it can live for a longer period and eat less food and as it lives for a longer period it should eat more in the long run meaning that its score would have been higher. One way I could potentially have done this is have the snake follow its tail and then calculate the distance of the food from the head position by using snake.DistanceFrom(Snake.HeadPosition, snake.FoodPosition) and then set a limit so if the head and food are less than 5 cells away and it is safe to get the food then the snake moves towards the food and eats it.