AI ToDo

* Week 3
  + Evade
    - Editor Weight
  + Arrive
    - Min speed
    - You do need to come to a complete stop, make sure you are reducing your speed to 0 through the weight that is applied.
    - FIXED: Changed deceleration value from 0.5 to 1.0.
  + Post week 3 tasks to forum with explanations
* Week 4
  + Wander
    - Circle moves behind enemy
    - Debug lines are outside of circle
      * Point on circle is never used
    - Circle distance changes over time
    - Maths Magnitude 0 check
    - Wander, I assume you are using your position or velocity (not normalised) to dictate your forward vector. The fast you go the further that circle will move if you keep the distance in.
  + Collision Avoidance
    - Current code not making a difference
    - Increased the weighting
  + Group Movement
    - Separation
    - Alignment
      * Seems to be working – check with Davin
    - Cohesion
      * Not working ocrrectly
* Week 5
  + Decision making
    - Ent.TakeDamage(m\_AttackPower);
    - Trigger detection not working for player weapon (the collision box pushes the enemy)
    - SOLUTION: set the player weapon collider to a trigger
    - Find closest health pickup
    - Make health pickup heal player and enemy
    - Evade if no health pickup
    - Move all if statement stuff into the switches
    - Make a FSM graph
    - Post FSM graph forum
    - Post Current enemy to forum
    - Go to last position
    - Can get stuck on wall when seeking
      * Switch from seek to A star
        + GetClosestPointOnPath – if the entity is moved from the path (i.e. player hits them), get the closets point on the path instead of going back to the current node
      * Player and AI moves outer walls
    - Make FSM clearer to read
      * Go to -> moving towards
      * “Last seen” Position reached
      * Pursue and attack player (maybe player in attack range)
      * Player not visible -> not in line of sight
    - Post new FSM
    - Post A star (seek health, last seen)
* Week 6
  + PathFinding
    - Manhattan distance
    - euclidean distance
  + Player and AI moves outer walls
  + A\* - Standard
    - Path reaches target but goes over colliders
      * Added check if walkable
    - Path avoided collider but cannot go back to previous nodes on the open list when going the wrong way.
    - Added a check to see if the node is in the list (instead of nodeinfo as the parent would change)
      * If the lowest f cost node is not in the open or closed list, it is the next node
    - Works better but cuts some corners
      * Only check non-corner neighbours i++ -> i + 2
    - Path line, enemy keeps moving back and fourth
      * Path node positions need to be added backwards to make the start position the first position in the list
    - Increased weighting to make enemy stick to path more.
    - Works correctly most of the time but switch man and euc around as euc ignores obstacles between the start and end
      * Tried this but doesn’t fix the problem
  + A\* Diagonal
  + A\* No Cut Corners
  + Consider fixing neighbour/ closed list as it is very large when pathfinding

# Group Movements

The update functions of all the group behaviours begin with the following:

Firstly, I used the Physics2D.OverlapCircleAll function to get an array of colliders within the specified range of the entity. It then loops through each collider and checks if it is another valid entity using the seperation behaviour script and it does not belong to the same entity using this instance of the script.

If it is a valid entity, another check is performed to see if the dot product of the two entities is greater than whatever the FOV is set to. If it is, then the unique group behaviour code is executed.

**With No Group Behaviours**

## Separation

In the loop, the force to add (calculated by dividing the normalised vector to the entity by the absolute vector to the entity) is added to the entity’s accumulated separation force.

Once the loop ends, the entity’s normalised accumulated separation force is multiplied by its weight and is returned.

**With Separation**

The entities now keep a distance away from each other, however, when next to the player, they begin to group up and no longer keep a distance.

Part of the issue was that my accumulated force was set to zero outside of the update loop, meaning that it kept getting larger each time the update loop was called. I also adjusted the weighting and separation range to more suitable values.

## Alignment

Adds the neighbour's forward vector to the entity’s accumulated heading force.

Once the loop ends, the alignment force is calculated by first dividing the cumulated heading force by the number of neighbours within range and then subtracting entity’s normalised velocity.

The normalised alignment force multiplied by it’s weight is then returned.

**With Alignment**

## Cohesion

The neighbour’s position is added to the accumulated position.

Once the loop ends, the target position is calculated by dividing the accumulated position by the number of neighbours in range. The seek behaviour is then used to seek the target position.

**With Cohesion**

Now that I have implemented all three of the group behaviours, I can combine them

**With All Behaviours**

# A Star

I have managed to fix my previous issue where shortest path is not always found when the end node is to the right of the start node. The problem was that when adding a neighbour node to the open list, the gCost was being set to the distance between the current node and the neighbour node instead of the distance plus the accumulated gCost.

Now the A star pathfinding algorithm is working correctly.

Currently the algorithm is using the Euclidean distance for finding the gCost and the Manhattan distance for finding the hCost. I tried switching these around, however, I think that the AI looked less believable, so I switched it back.

# A Star – Diagonal

Before looping through all of the current node’s neighbour nodes, I created a new variable called neighbourIncrement. This value is added to the index of the aforementioned loop.

If m\_AllowDiagonal = true, the increment is set to 1 so that all the neighbour nodes are checked in the loop.  
If m\_AllowDiagonal = false, the increment is set to 2 so that the corner nodes are skipped in the loop.

# A Star – No Cutting Corners

Before looping through all of the current node’s neighbour nodes, I added another for loop that checks if the current node is next to a wall node. If the neighbour node is a wall, it then checks the wall's neighbours to see how many walls it is connected to. If the wall is connected to 3 or less other walls, then it is a corner node. If the neighbour node is a corner wall, then neighbourIncrement is set to 2 so that it doesn’t get the next corner neighbour node (to prevent cutting the corner).