Game 6

Flocking

Flocking ~ Introduction (1)

- The main task in relation to game 6 was to implement flocking movement. This is movement which mimics that of birds seen in the natural environment.
- This type of movement utilises boids which one can view as the birds that compose the flock. Each boid has the same behaviour model however said behaviour is impacted by the movements of ones neighbours.
- The main components required for a flocking implementation are cohesion, avoidance and alignment.
- This type of movement can be quite effective in bringing to life aspects of games such as swarms of enemies, crowds and much more.

Flocking ~ Introduction (2)

- Cohesion – This is the most crucial component of flocking as it determines the overall movement of the boids. Cohesion work by moving the boid to the average position of its neighbours, in turn this simulates the idea of the boid following its neighbouring boids.
- Avoidance – This is also a pivotal component of flocking as it facilitates the avoidance of obstacles by the boids. These obstacles can either be environmental or even other boids.
- Alignment – This component serves to align the boids composing a flock in the same direction such that their forward movement is in sync.

Flocking ~ AI Explanation (1)

- The implementation of the flocking algorithm required the use of several behaviours which each implement cohesion, avoidance and alignment respectively.
- This was achieved through the creation of an abstract script FlockBehaviour which served as the basis for the other behaviours. Said script included an abstract method CalculateMove() which required three parameters these being the agent, a list of boid transforms and a flock object.
- The basis for all three behaviours is as follows, first a check is made to see if the list of transforms is empty. If this is the case than the boid has no neighbouring boids and thus returns a Vector2.zero as it doesn't have anything to base its movements on. Secondly the list of transforms is filtered such that only boids of the same type are considered.
- The following section differs based on the behaviour:
  - Cohesion – The average position of said boids is found and returned
  - Avoidance – Checks if distance between the boid and any obstacles is less than the specified radius, if this is the case an opposing force is applied to the boid to move it away from said neighbour. Said force is than returned.
  - Alignment – The average transform of said boids is found and then returned.

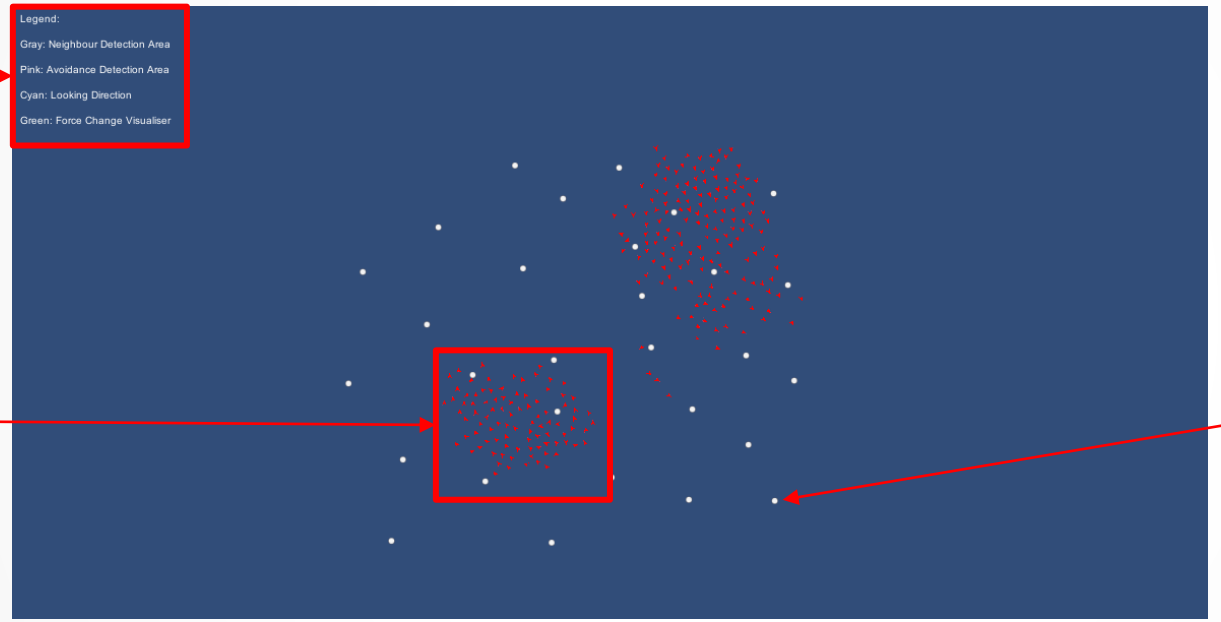
Flocking ~ AI Explanation (2)

- The values returned from the behaviours are then passed to the composite behaviour whose job is to combine the forces provided, into one singular force according to the weights assigned to each sub-behaviour.
- The force calculated by the composite behaviour method is than applied to the boid facilitating its movement.
- This implementation lends itself well to further expansion as by creating further sub-behaviours the movement of the boids can be refined further. This can be seen with the addition of the StayInRadius behaviour which forces the boids to stay in a specific range. This was used to keep the flocks on the screen.

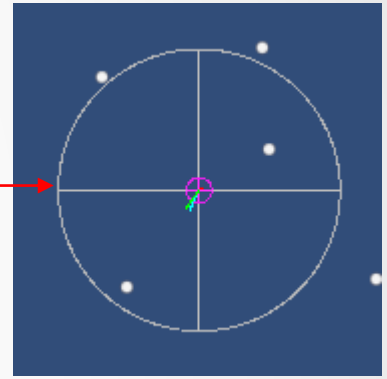
Flocking ~ Mini-Game Implementation (1)

Playable Area:

Gizmos legend



Gizmos



Flock

Obstacle

Flocking ~ Mini-Game Implementation (2)

- The flocking implementation was inspired from [1]. The scripts found in the Scripts sub-directory include the:
  - ContextFilter – This script serves as the abstract class for the implementation of any filter script.
  - Flock – This script contains the crucial methods for the managing of the flock. The start method initialises the flock agents and repeatedly calls the MoveToPlace functionality if its set to active. The Update method calls the necessary methods to move the individual boids. The GetNearbyObjects() methods retrieves all nearby collider2D be they from obstacles or other boids. The MoveToPlace() method changes the location to which the flock is forced to move given that the method is activated. Finally the last two methods display the gizmos and GUI respectively.
  - FlockAgent – This script facilitates the movement of the individual boid.
  - FlockBehaviour – This script serves as the abstract class for the implementation of any behaviour script.

Flocking ~ Mini-Game Implementation (3)

- The scripts found in the BehaviorScripts sub-directory include the:

  The basis for all three behaviours is as follows, first a check is made to see that the required data is present. If this is not the case than a Vector2.zero is returned. Secondly the list of transforms is filtered such that only boids of the same type are considered.

  The following section differs based on the behaviour:

  - CompositeBehaviour – The CalculateMove() method is called for all specified behaviours, the result is multiplied by the specified weight and added to the other results to form a singular movement vector which is than applied to the boid.
  - StayInRadiusBehaviour – A check is carried out to see if the boid is within the specified area. If this is not the case a vector in the direction of the specified area is generated and returned.
  - SteerCohesionBehaviour – The list of transforms is filtered and the average position of said boids is than found, passed through a SmoothDamp() function and then returned.
  - AvoidanceBehaviour – The list of transforms is filtered and the square distance between the boid and any neighbour is checked to see if it is less than the specified radius, if this is the case an opposing force is applied to the boid to move it away from said neighbour. Said force is than returned.
  - AlignmentBehaviour – The list of transforms is filtered and the average transform of said boids is than found and returned.

Flocking ~ Mini-Game Implementation (4)

- The scripts found in the FilterScripts sub-directory include the:
  - SameFlockFilter – This script implements the ContextFilter class to allow the boid to avoid other boids in the same flock. This is achieved by filtering the boids according to whether they belong to the same flock or not. This function is used in the behaviours so as to only consider boids of the same flock.
  - PhysicsLayerFilter – This script implements the ContextFilter class to allow the boid to avoid obstacles present in the indicated layer. This is achieved by filtering the obstacles which are in different layers and then returning said filtered list to be used in the behaviour objects.

Flocking ~ Exercise (1)

Now it's your turn to Code ! – Let's implement the pattern movement algorithm :)

Open the Behaviour Scripts sub-directory your task is to implement the following behaviour components:

- CohesionBehaviour
- StayInRadiusBehaviour
- AvoidanceBehaviour
- AlignmentBehaviour

Flocking ~ Exercise (2)

Note: This method is used to force the agents to converge onto the same point (Cohesion Behaviour).

1. Access the context list which holds a series of transform information and work out its count (context is passed to the CalculateMove method). If said count is 0 return Vector2.zero
   - (Hint: Use .Count to get the count)
2. Create a Vector2 variable named cohesionMove and store in it Vector2.zero
3. Create a list of transforms named filteredContext and filter all the transforms in the context list such that only transforms from the same flock are used.
   - (Hint: If filter is null then the entire context list is used)
   - (Hint: Use filter.Filter(agent, context) to filter the context list according to the agent flock)

Flocking ~ Exercise (3)

4. For each transform in the filteredContext list add its position to the cohesionMove variable and work out its averge. Store the average in cohesionMove.
   - (Hint: Use .position to get the objects position)
5. Pass the cohesionMove variable through the Vector2.SmoothDamp() function and store the result in cohesionMove
   - (Hint: Use Vector2.SmoothDamp(agent.transform.up, cohesionMove, ref currentVelocity, agentSmoothTime))
6. Return cohesionMove
7. Go into the BehaviourObjects sub directory right click and choose create/Flock/Behaviour/SteerCohesion, this will create a behaviour object.
8. By clicking on said object you can edit the agent smooth time to smoothen the agent movement
9. Select the Composite behaviour and add the newly created behaviour to the compsite one. Specify a weighting for said behaviour, this affects the importance that the behaviour is given in relation to other behaviours.
   - (Hint: A weight of 4 can be set)

Flocking ~ Exercise (4)

Note: This method is used to keep the agent within a specified area, mainly so that none of the agents go off screen (Stay in Radius Behaviour)

1. Calculate the offset of the agent from the center, resulting in a vector pointing in the opposite direction
   - (Hint: Use center variable)
2. Divide the centerOffset magnitude variable created in 1 by the radius of the circle and store it in a variable temp. The value in this variable will denote if the individual agent is within the specified radius or not 0 - at the center | < 1 within the radius | > 1 beyond the radius
   - (Hint: Use centerOffset.magnitude)
3. Check if the agent is within the radius, if this is the case return a Vector2.zero otherwise return the centerOffset multiplied by temp squared. This force will be used in the composite behavior method which is provided, to apply a force to the agent thus keeping it within the specified area.
4. Go into the BehaviourObjects sub directory right click and choose create/Flock/Behaviour/StayInRadius, this will create a behaviour object.
5. By clicking on said object you can edit the circle center and radius, change these values until you are satisifed
6. Select the Composite behaviour and add the newly created behaviour to the compsite one. Specify a weighting for said behaviour, this affects the importance that the behaviour is given in relation to other behaviours.
   - (Hint: A weight of 0.1 can be set)

Flocking ~ Exercise (5)

Note: This method is used to indicate to the agents what they should avoid and how to avoid it (Avoidance Behaviour)

1. Access the context list which holds a series of transform information and work out its count (context is passed to the CalculateMove method). If said count is 0 return Vector2.zero
   - (Hint: Use .Count to get the count)
2. Create a Vector2 variable named avoidanceMove and store in it Vector2.zero
3. Create an integer variable named numAvoid and set it to 0
4. Create a list of transforms named filteredContext and filter all the transforms in the context list such that only transforms from the same flock are used
   - (Hint: If filter is null then the entire context list is used)
   - (Hint: Use filter.Filter(agent, context) to filter the context list according to the agent flock)

Flocking ~ Exercise (6)

5. For each transform in the filteredContext list check if the square distance between the agent and its neighbour is less than the specified radius. If this is the case increment numAvoid and work out the difference between the agent and the object to avoid. Add this value to avoidanceMove and then work out its avergae, store the result in avoidanceMove.
    - (Hint: Use .position to get the objects position)
    - (Hint: Use numAvoid to work out the average)
6. Return avoidanceMove
7. Go into the BehaviourObjects sub directory right click and choose create/Flock/Behaviour/Avoidance, this will create a behaviour object.
8. By clicking on said object you can edit the filter used. In this case enter the FilterObject directory and drag the SameFlockFilter into the flock area.
9. Select the Composite behaviour and add the newly created behaviour to the compsite one. Specify a weighting for said behaviour, this affects the importance that the behaviour is given in relation to other behaviours.
    - (Hint: A weight of 10 can be set)

Flocking ~ Exercise (7)

Note: This method is used to align the agents to one another so that they face the same direction whilst moving (Alignment Behaviour)

1. Check if the context count is 0, if this is the case return the agent.transform.up. This makes sure that the agent keeps facing whichever direction it was facing.
2. Create a Vector2 variable named alignmentMove and store in it Vector2.zero
3. Create a list of transforms named filteredContext and filter all the transforms in the context list such that only transforms from the same flock are used.
    - (Hint: If filter is null then the entire context list is used)
    - (Hint: Use filter.Filter(agent, context) to filter the context list according to the agent flock)
4. For each transform in the filteredContext list add the transform.up of each transform to the alignmentMove and work out its average. Store the result in alignmentMove.

Flocking ~ Exercise (8)

5. Return alignmentMove
6. Go into the BehaviourObjects sub directory right click and choose create/Flock/Behaviour/Alignment, this will create a behaviour object.
7. By clicking on said object you can edit the filter used. In this case enter the FilterObject directory and drag the SameFlockFilter into the flock area.
8. Select the Composite behaviour and add the newly created behaviour to the compsite one. Specify a weighting for said behaviour, this affects the importance that the behaviour is given in relation to other behaviours.
    - (Hint: A weight of 1 can be set)

Flocking ~ Conclusion

- In conclusion flocking movement has the benefit of being useful to implement independent agents which navigate according to their own perception of the environment. It also lends itself well to further development when it comes to more specific behaviour.

Flocking ~References

[1] – Prof. A. Dingli, ICS2211: "LEVEL 2 MOVEMENT" [Online]. Available: https://www.um.edu.mt/vle/pluginfile.php/1103257/mod_resource/content/1/Level2_Movement.pdf [Accessed: 18-Mar-2023]