Crowd Simulation API

Version 1.0

Crowd Simulation Solution for UNITY

User's Manual

In the document followed, Crowd Simulation API and FAME (stands for Flock Animation and Modelling Environment) will be used interchangeably due to legacy naming conventions.

09 Dec 2013

CONTENTS

Release Note 4

1. Feature Overview 5

1.1 Creating flock in any desired formation shape 5

1.2 Self-Organizing Agents within Flock formation 5

1.3 Morphing of formation shape 6

1.4 Path following 7

1.5 Obstacle Avoidance 7

1.6 Simulating Environmental Forces – Vector Fields 8

1.7 Terrain 9

1.8 Configurable Crowd Behavior 10

2. FAME Modules 11

2.1 FameManager 11

2.2 FlockGroup 12

2.3 FlockMember 13

2.4 FameObstacle 15

2.5 FamePath 15

2.6 FameField 16

3. Tutorial 18

3.1 Setting up the Scene Via UnityEditor 18

Create a FAMEManager 18

Initializing terrain in FAME 19

How to create a flock group in editor? 20

Applying steering behaviors to the FlockGroup/FlockMember 22

Setting the agent properties for the FlockGroup/FlockMember 24

How to create obstacles in editor? 26

How to create vector fields in editor? 28

3.2 Using Crowd Simulation API during Runtime 29

Adding a FlockGroup to the scene 29

Populating the FlockGroup 29

Applying basic transformation to Formation Shape 30
Changing the FlockGroup and FlockMembers Attributes and Steering Behaviors 31
Deleting a FlockGroup 33
Deleting a flock member 33
Collision Flag and Field Flag 33
Adding obstacles to the scene 34
Changing obstacle parameters 35
Setting Obstacle Flag 36
Removing an obstacle 36
Adding vector field to the scene 36
Changing vector fields parameters 37
Setting Field Flag 38
Removing a vector field 38
Defining a path (FamePath) in the scene 38
Getting & Setting of FamePath Control Points 39
Performing Query in FAME 39
How can a FlockGroup changes it formation its shape? 42
How to change formation shape via control point movement 42
How to perform path following for a FlockGroup 43
Creating a new sub-formation 43
FlockMembers leave a FlockGroup 44
Joining a flockGroup 45
Spreading out FlockMembers to fill up the formation shape uniformly 45
3.3 Advance (Class Inheritance) 46
Inheriting a FlockGroup Object 46
Inheriting a FlockMember Object 46

RELEASE NOTE

09 Dec 2013, Version 1.0 - Initial Version

1. FEATURE OVERVIEW

1.1 Creating flock in any desired formation shape

Crowd Simulation API supports creation of flock formations that are formulated in the form of soft polygonal shape constraints. Thus agents belonging to a particular group shall stay within the defined polygon when possible.

The shape constraint is populated with agents by first performing a uniform sampling, to generate the same number of sample points on the polygon. The sample points are then assigned as destination position to each members of the flock. After which, the guiding steering force will attract the agents towards their individually assigned position. (See Figure 1)

FIGURE 1 EXAMPLE OF CREATING FLOCK IN ANY DESIRED FORMATION SHAPE

1.2 Self-Organizing Agents within Flock formation

It is natural that while the agents are forming the formation, those who arrived first should move inwards to occupy the front spaces while those who arrive later to fill up the rear. Here, Crowd Simulation API provides an automated approach to reorganize the agents within the formation during runtime. As the agents navigate in the virtual environment, some might encounter anomalies, such as obstacles, steep terrain and other agents of different flock groups. Thus, these agents would have to make a detour and take a longer time to move back to the formation. Other agents that manages to arrive at the formation first would move inwards, allowing agents that arrive later to fill up the formation from the rear. (See Figure 2)

FIGURE 2 EXAMPLE OF SELF-ORGANIZING AGENTS WITHIN FLOCK FORMATION

1.3 Morphing of formation shape

Crowd Simulation API allows user to change/morph the formation shape easily during runtime. Shape morphing refers to the process where a given shape transforms into another shape. In the context of shape constraint flocking, agents housed in a shape formation constraint morph or transform into another formation shape, while filling up the space inside the new formation strategically and naturally. (See Figure 3)

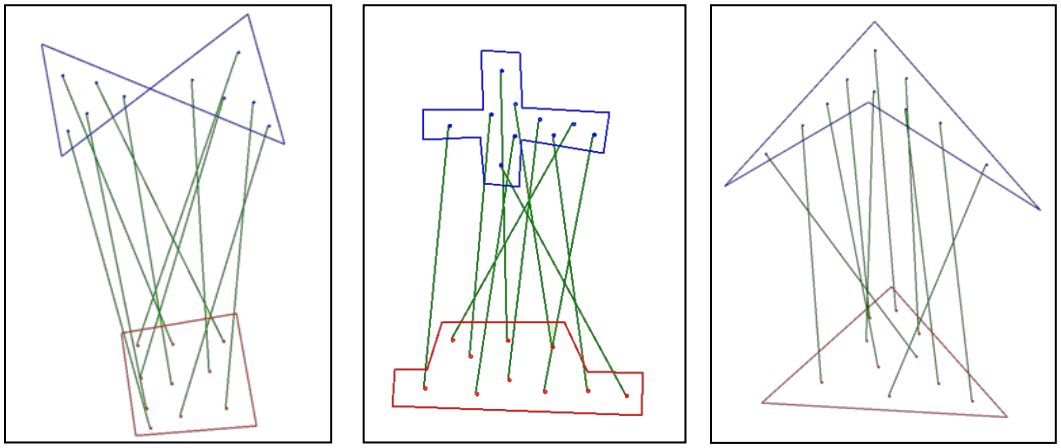

FIGURE 3 EXAMPLE OF AGENT-DESTINATION MAPPING (BOTTOM TO TOP)

1.4 Path following

The path following feature allows a group of agents to travel automatically along the defined path. The path can be defined easily in Crowd Simulation API by defining the list of points that it should pass. (See Figure 4)

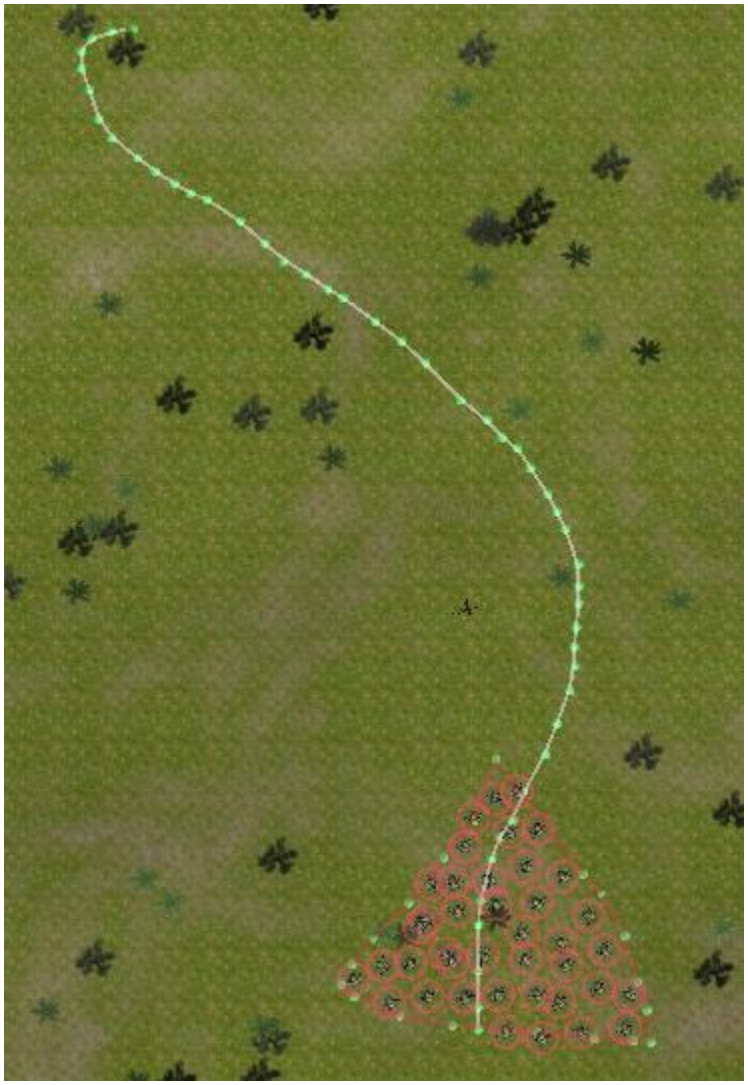

FIGURE 4 EXAMPLE OF PATH FOLLOWING

1.5 Obstacle Avoidance

Crowd Simulation API crowds are able to react and avoid collision with obstacles in the scene. Crowd Simulation API supports both static and dynamic obstacles. Another unique feature of Crowd Simulation API is the ability to define the obstacle type such that it allows certain agents to pass through while blocking the rest. (See Figure 5)

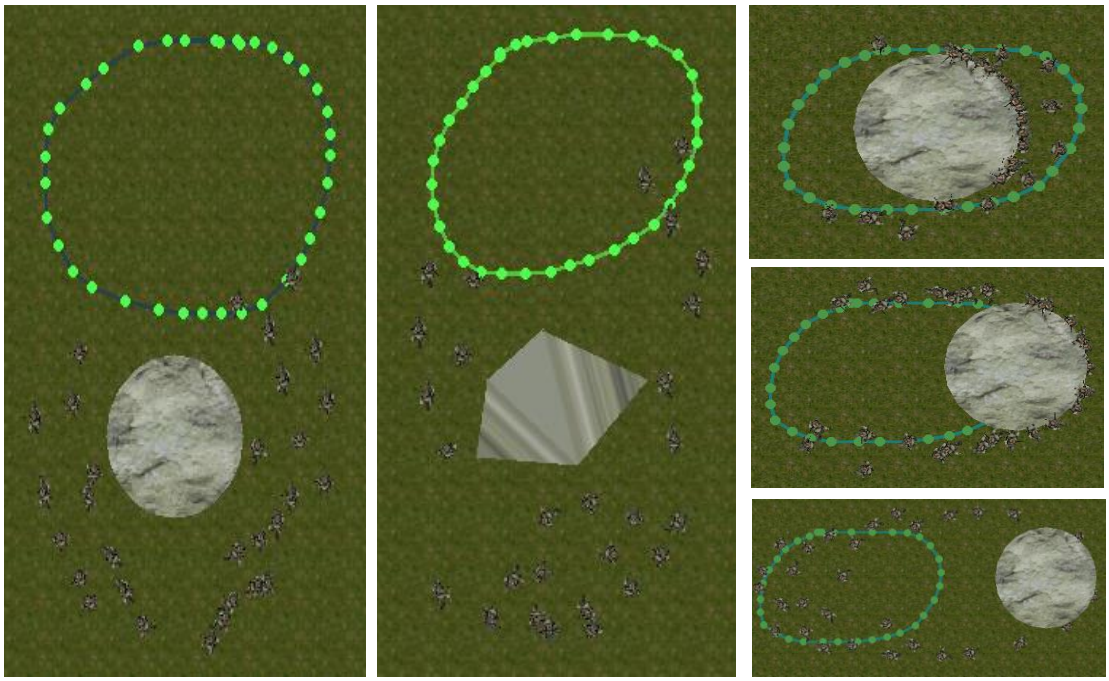

FIGURE 5 EXAMPLE OF OBSTACLE AVOIDANCE (LEFT: ROUND OBSTACLE, MID: POLYGONAL OBSTACLE, RIGHT: MOVING OBSTACLE

1.6 Simulating Environmental Forces – Vector Fields

Vector field is represented by an array of vectors defining the forces in a region. Each vector denotes, for example, a force that is asserted onto the agents as they travel across the region. Crowd Simulation API provides four different templates of potential fields for ease of use in defining and placing environmental forces within the virtual environment. These include the 1) directional field, 2) circular field 3) attraction and 4) repulsion field. Fusions of the templates can be easily achieved to arrive at different desired effects. (See Figure 6)

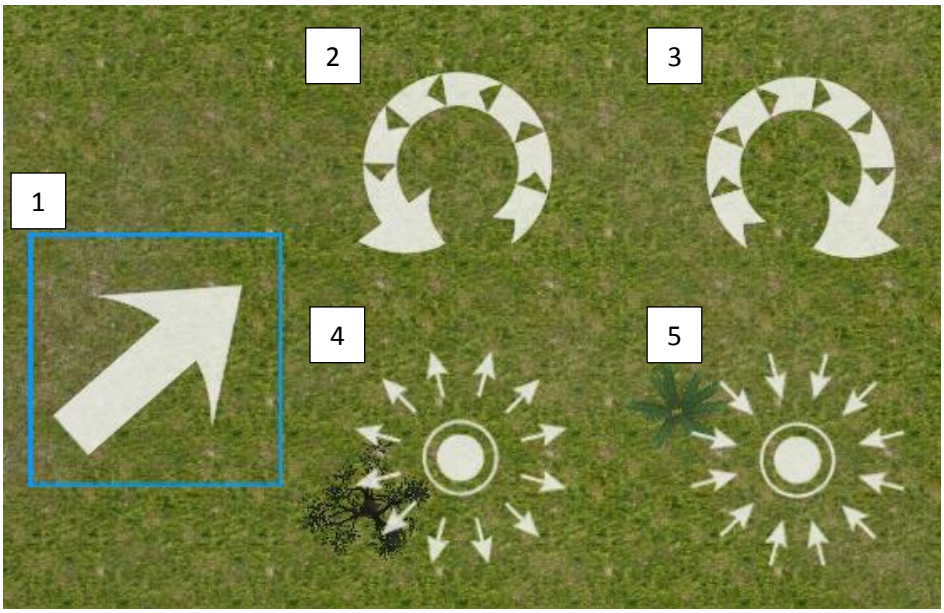

FIGURE 6 EXAMPLE OF VECTOR FIELD - 1: DIRECTIONAL FIELD; 2: CIRCULAR FIELD (ANTI-CLOCKWISE); 3: CIRCULAR FIELD (CLOCKWISE) 4: REPULSION FIELD; 5: ATTRACTION FIELD

1.7 Terrain

Modelling of various forces of nature such as the flow of the water current or the invisible but powerful forces of wind fronts are important to make a virtual environment more realistic. Crowd Simulation API takes into considerations the terrain effect on the movement behavior of flock agents. Naturally, it is more difficult to climb uphill than walking on flat ground. Similarly, it is a breeze to travel down slope due to help of the gravitational pull. Crowd Simulation API models this simple phenomenon by adjusting the maximum speed of the agent dynamically based on the slope of the terrain where the agent is located (See Figure 7).

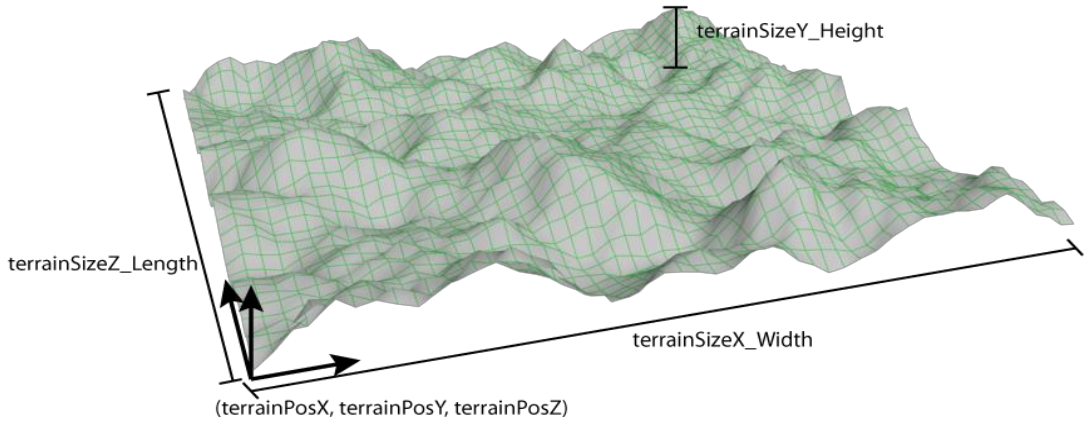

FIGURE 7 EXAMPLE OF HEIGHT MAP OF TERRAIN

1.8 Configurable Crowd Behavior

The movement of the agents is based on a modified version of the "Boid" model. Crowd Simulation API allows you to easily configure the steering behavior for each group of agents easily. These parameters include the radius of effect and the weights of each force. (See Figure 8)

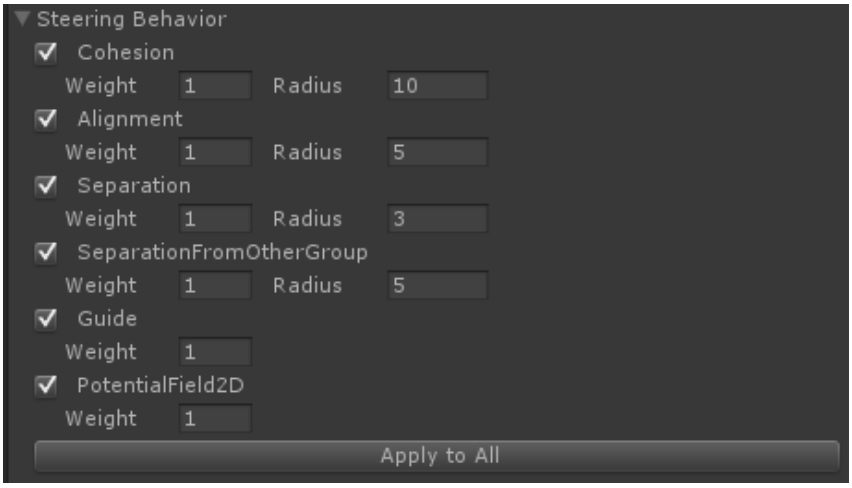

FIGURE 8 EXAMPLE OF STEERING BEHAVIOR SETTING IN INSPECTOR PANEL OF UNITY EDITOR

2. CROWD SIMULATION API MODULES

2.1 FameManager

The FameManager is the main point of interaction between the Crowd Simulation API library and your game. It handles the creation and deletion of all Crowd Simulation API related components.

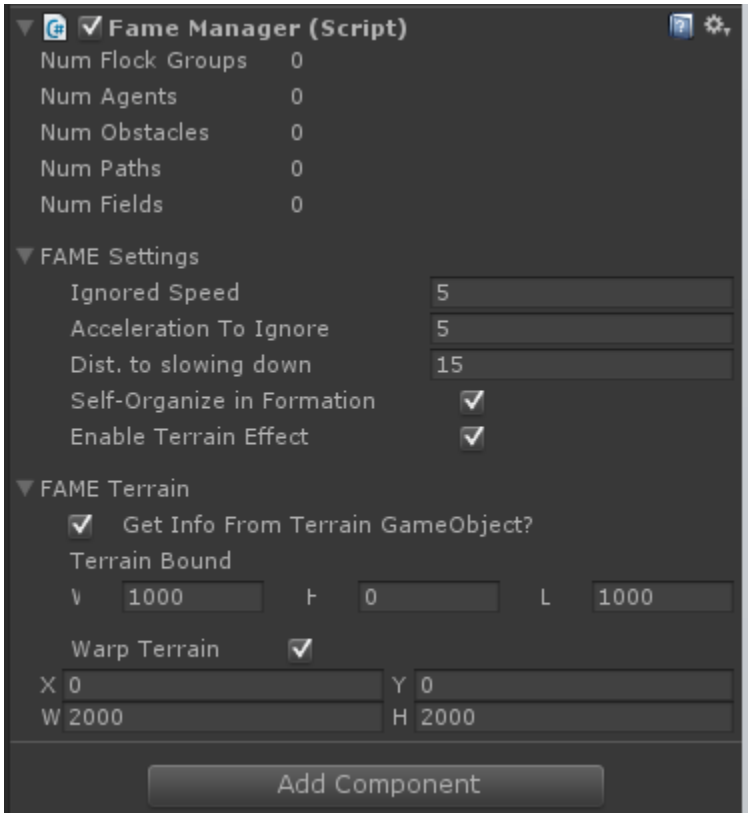

FIGURE 9 EXAMPLE OF FAMEMANAGER IN INSPECTOR PANEL OF UNITY EDITOR

Parameters	
Ignored Speed	The minimum speed that the agent will start to move
Acceleration To Ignore	The minimum magnitude of the acceleration that will be applied to the agent. Any acceleration lower than the specified value will be ignored.
Dist. to slowing down	The distance between the agent and its destination that the agent will start to slow down.
Self-Organize in Formation	Whether the agents within the same FlockGroup will exchange their destination position to reduce the overall distance travelled.
Enable Terrain Effect	Whether the gradient of the terrain will affect the movement speed of ground moving FlockMembers.
Warp Terrain	Whether FlockMembers exiting the specified boundary will get

	teleported to the opposite end of the map.

2.2 FlockGroup

A FlockGroup is a class to represents the group of agents. It contains information regarding the shape of the formation and the list of agents that belongs to the group. Each flockGroup has a unique integer ID to identity the FlockGroup.

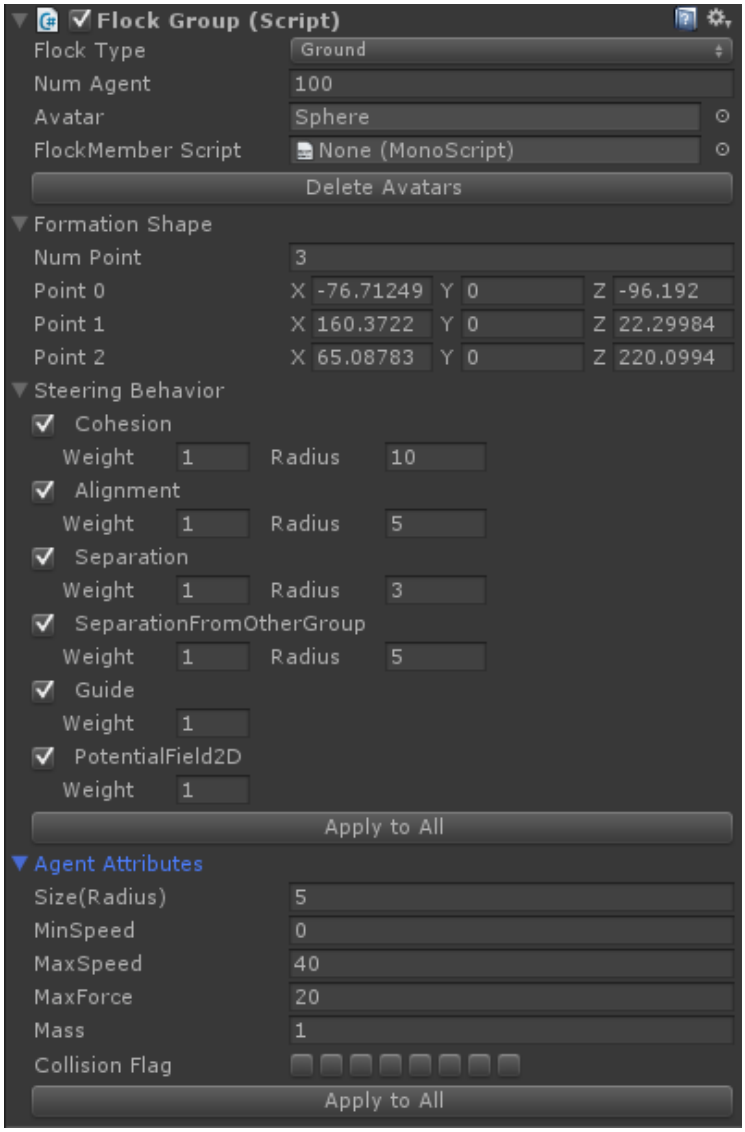

FIGURE 10 EXAMPLE OF FAMEGROUP IN INSPECTOR PANEL OF UNITY EDITOR

Parameters	

FlockID	The unique id that represents the flockGroup
FlockType	The type of unit that this group represents. It represents either a Ground or Flying type. The ground flock moves in the XZ axes while the Flying flock will move in all the three axes (XYZ). Another significant difference is that the Ground flock will take the terrain gradient into consideration when calculating its movement while the Air flock ignores the terrain gradient completely
CurrentPosition	The current position of the flock.
shapeConstraint	The set of points that the polygon shape which represents the spaces that flockGroup occupy. Agents belonging to a particular group shall stay within the defined polygon when possible.
MeshHeight	The meshHeight represents the height of the shapeConstraints. This param is used only for the creation of Air type flock. The agents will populated within the 3D mesh instead of the 2D polygon when creating the Air type units.
AgentRadius	Radius of the circular space that the agent will occupy. Param is assigned to each FlockMember belonging to the group upon initialization.
AgentMinSpeed	Minimum speed that the agent will travel. Param is assigned to each FlockMember belonging to the group upon initialization.
AgentMaxSpeed	Maximum speed that the agent will travel. Param is assigned to each FlockMember belonging to the group upon initialization.
AgentMass	Mass of the agent. Param is assigned to each FlockMember belonging to the group upon initialization.
AgentMaxForce	The maximum force that is applied on each agent in each updates Param is assigned to each FlockMember belonging to the group upon initialization.
CollisionFlag	A 8bit flags that is used to determine whether the agents can travel through the obstacles. Param is assigned to each FlockMember belonging to the group upon initialization.
FieldFlag	A 8bit flags that is used to determine whether the different types of vectors field in the map will affect the movement behavior of the agents in the group. Param is assigned to each FlockMember belonging to the group upon initialization.

2.3 **FlockMember**

A FlockMember is a class to represent each individual agent in the game. Each agent has a unique integer ID to identity the FlockMember.

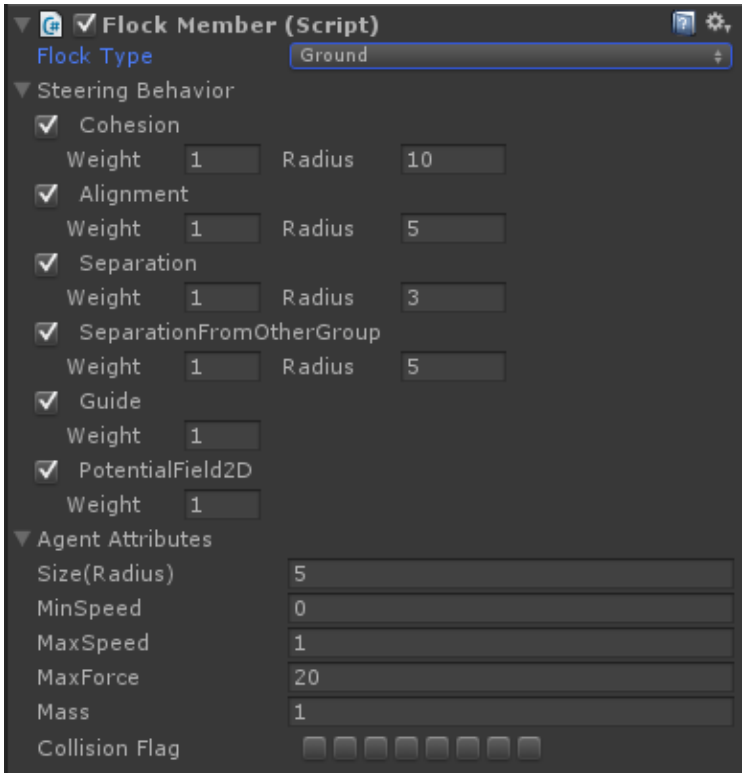

FIGURE 11 EXAMPLE OF FAMEMEMBER IN INSPECTOR PANEL OF UNITY EDITOR

Parameters	
`AgentID`	The unique id that represents the FlockMember
`FlockType`	The type of unit that this FlockMember. It represents either a Ground or Flying type. The ground flock moves in the XZ axes while the Flying flock will move in all the three axes (XYZ). Another significant difference is that the Ground flock will take the terrain gradient into consideration when calculating its movement while the Air flock ignores the terrain gradient completely
`CurrentPosition`	The current position of the agent.
`Destination`	The assigned destination of the agent. The agent will move towards this destination position if the Guide steering behavior is applied to the agent.
`AgentRadius`	Radius of the circular space that the agent will occupy.
`AgentMinSpeed`	Minimum speed that the agent will travel.
`AgentMaxSpeed`	Maximum speed that the agent will travel.
`AgentMass`	Mass of the agent.
`AgentMaxForce`	The maximum force that is applied on each agent in each updates

CollisionFlag	A 8bit flags that is used to determine whether the agents can travel through the obstacles.
FieldFlag	A 8bit flags that is used to determine whether the different types of vectors field in the map will affect the movement behavior of the agent.

2.4 FameObstacle

The FameObstacles represents the area which the agent should avoid. There are two different kinds of obstacles in Crowd Simulation API: Round Obstacle and Polygonal Obstacle.

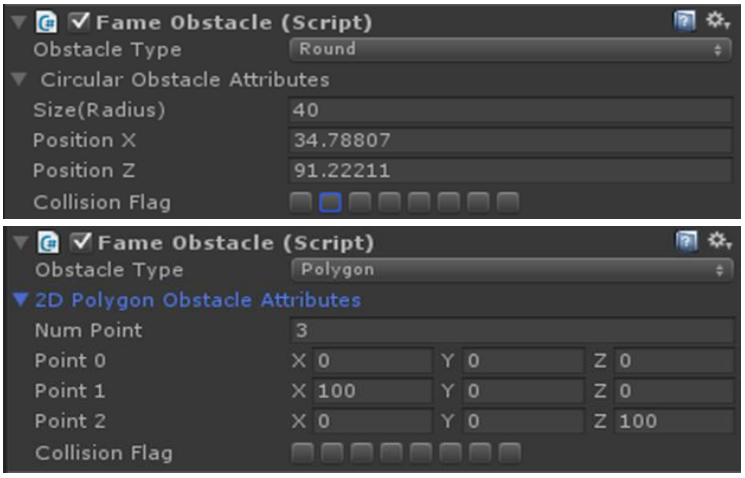

FIGURE 12 EXAMPLE OF FAMEOBSTACLE (TOP: ROUND OBSTACLE, BOTTOM: POLYGON OBSTACLE) IN INSPECTOR PANEL

Parameters	
ObstacleID	The unique id that represents the FameObstacle
obstacleType	The type of obstacle: Round or Polygon
obstacleRadius	Radius of obstacle if the obstacleType is Round
obstaclePoints	The list of point that defines the polygon shape of the obstacle
collisionFlag	A 8Bit flag which denote obstacle type. It is used to check whether the FlockMember will be able to pass through.
Speed	The speed which the obstacle moves
waypoints	An array of Vector point which the obstacle will move.

2.5 FamePath

The FamePath represents a spline path in the scene. It is used by the FlockGroup to perform path following.

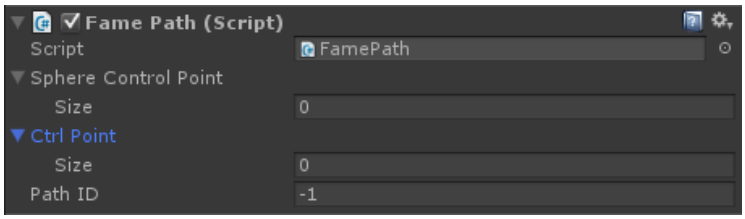

FIGURE 13 EXAMPLE OF FAMEPATH IN INSPECTOR PANEL OF UNITY EDITOR

Parameters	
`PathID`	The unique id that represents the FamePath

2.6 FameField

The FameField represents a vector field which is placed in the scene. It is an array of vectors defining the forces in a region.

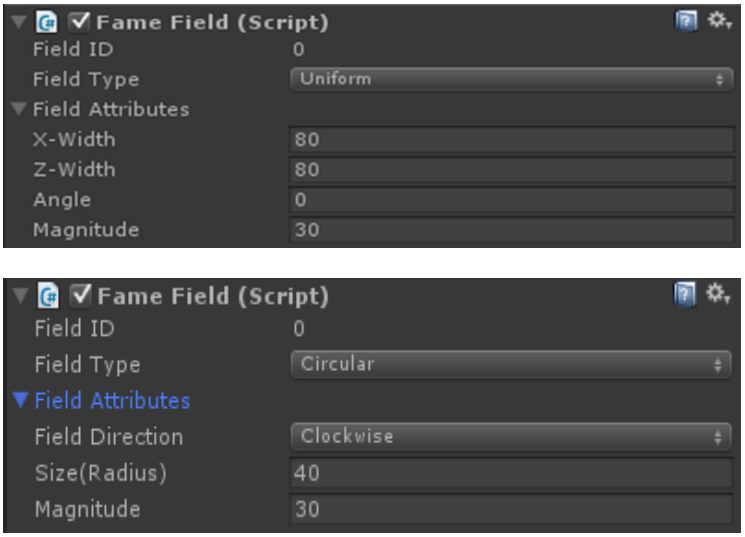

FIGURE 14 EXAMPLE OF FAMEFIELD (TOP: UNIFORM FIELD, BOTTOM: CIRCULAR FIELD) IN INSPECTOR PANEL

Parameters	
`FieldID`	The unique id that represents the FameField
`FameFieldType`	The field type. Possible values are: Uniform and Circular.
`FameCircularDirection`	The direction of the vector field for Circular type FameField. Possible values are: Clockwise, Anticlockwise, Attract and Repel
`fieldWidthX`	The width of the Uniform field (length measured in the x-axis).
`fieldWidthZ`	The height of the Uniform field (length measured in the z-axis).
`fieldAngleRad`	The angle representing the direction which the uniform field is

	pointing, where 0 represents a unit force in the X direction, i.e. Vector3(1,0,0);
`fieldRadius`	The radius of the Circular field.

3. TUTORIAL

3.1 Setting up the Scene Via UnityEditor

Create a FAMEManager

"FameManager" is a singleton that initializes and creates the environments of Crowd Simulation API library. The user need to create the Fame Manager first when the user want to use the library. It is recommended that the Fame Manager be created in the unity editor. The steps to create the Fame Manager from the unity editor are shown below:

Step 1,

Select the menu item "Create Empty" under the menu "GameObject" (or the user can use the shortcut key "Ctrl+Shift+N") to create an empty game object (See Figure 15).

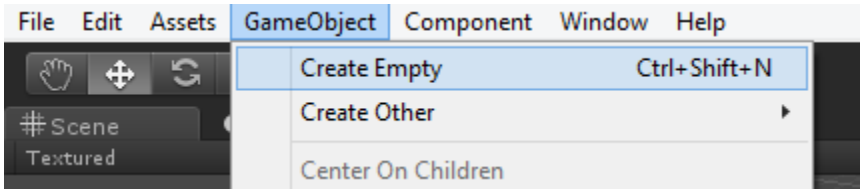

FIGURE 15 CREATE EMPTY OBJECT

Then, the user can find the empty object in the Hierarchy panel. The user can change its name to our own preference. In this case, its name is changed to "FameMgr".

Step 2,

Drag the "FameManager" script onto the empty object created.

Step 3,

Select the object "FameMgr" created in Step 1 from Hierarchy panel. The Inspector panel will now looks like Figure 16.

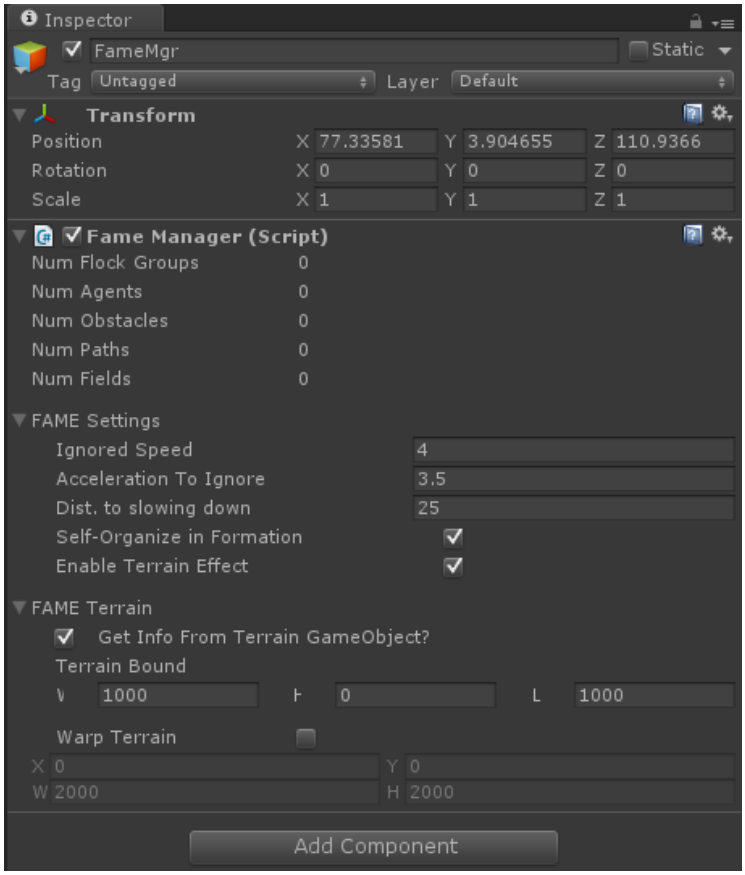

FIGURE 16 INSPECTOR PANEL OF FAMEMGR

Initializing terrain in Crowd Simulation API

Crowd Simulation API allows the user to specify the terrain profile which constrains agents' motions based on terrain topology. For example, with terrain profile, agents will slow down while travelling uphill and speed up when moving downhill. The terrain parameters can be found inside the FAMEManager component.

There are two ways to define a terrain:

1. Get the height information from the created terrain object in scene. To do this, the user only needs to check the box "Get Info From Terrain GameObject?", as shown in Figure 17

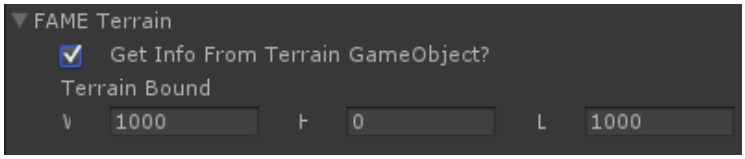

FIGURE 17 GET HEIGHT INFORMATION FROM TERRAIN OBJECT

2. Set the height information manually. To do this, the user needs to uncheck the box "Get Info From Terrain GameObject?", then load an Height Map asset and set its resolution, as shown in Figure 18

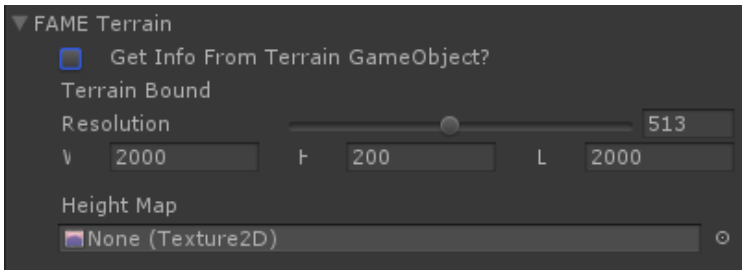

FIGURE 18 SET HEIGHT INFORMATION MANUALLY

How to create a flock group in editor?

"`FlockGroup`" is a script used to initialize and create flocks and agents.

Step 1,

Create an empty object and change its name as described earlier.

Step 2,

Drag the "`FlockGroup`" script onto the empty object created.

Step 3,

Set the configurations of flock group. In this case, we will set the "Flock Type" to Ground, the "Num Agent" to 100, "Avatar" to a normal sphere, and "FlockMember Script" as FlockMember

script. Next, the user needs to set the formation shape for flock group. For this, the user can modify the formation shape as he pleases. In this case, we set the shape to a convex quadrilateral, and keep other options as default. At this point, the `FlockGroup` Component should look like Figure 19.

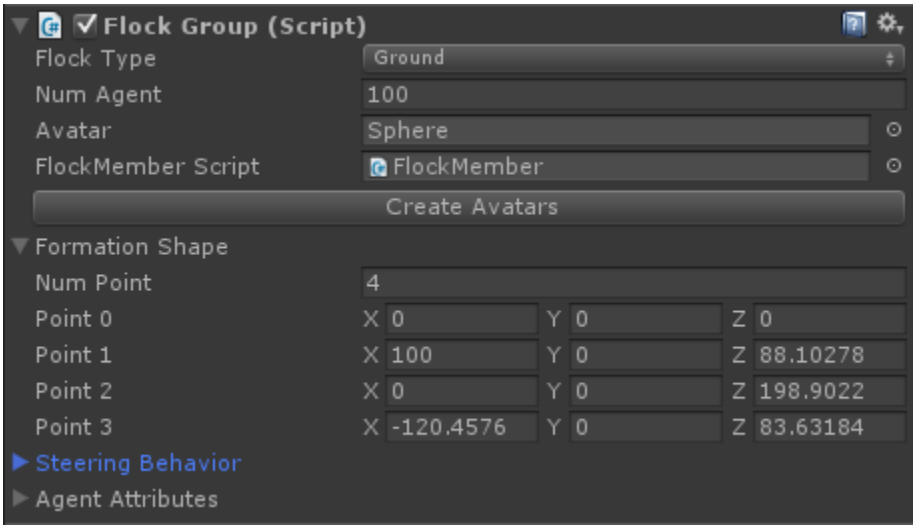

FIGURE 19 CONFIGURATIONS OF FLOCK GROUP

Step4,

Click the button "Create Avatars". The script will automatically generate the flock group and agents, as shown as in Figure 20.

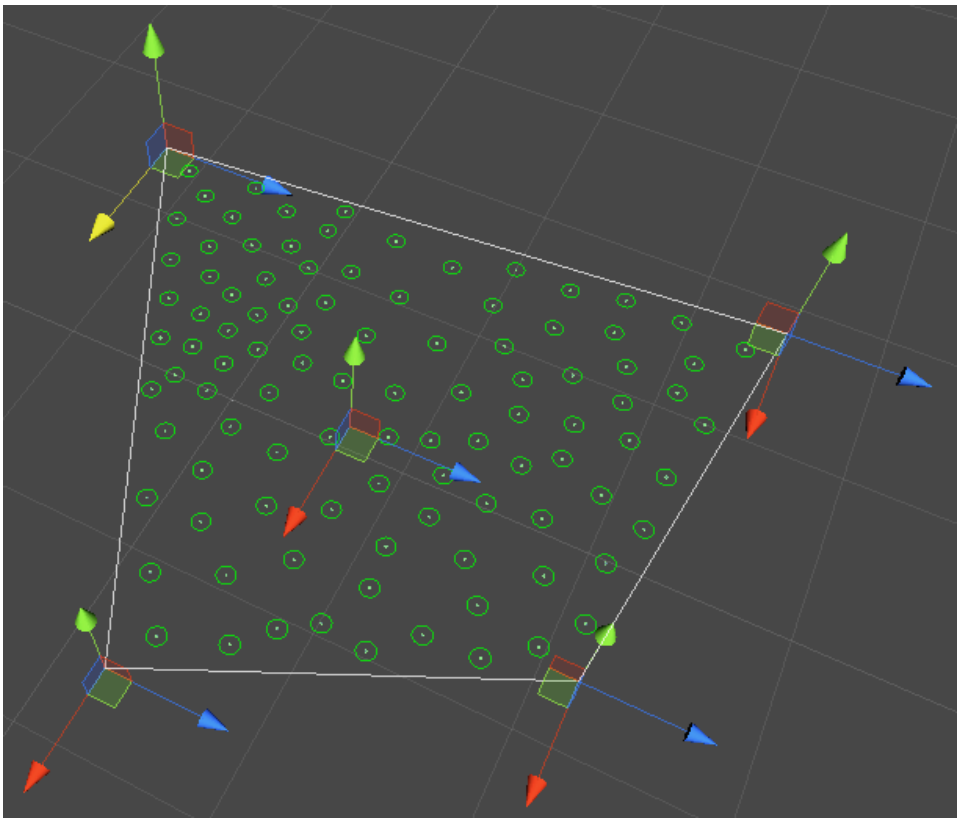

FIGURE 20 CREATED FLOCK AND AGENTS

Applying steering behaviors to the FlockGroup/FlockMember

Crowd Simulation API allows you to easily customize the steering behavior of each FlockMembers via the editor. After populating the FlockGroup with agents from the previous step, click on the Steering Behavior foldout to see the list of steering behaviors that can be assigned to the agents (See Figure 21). There are 6 different behaviors provided, namely:

- **Cohesion**: The cohesion behavior is a force that guides the agent to the center of nearby neighbors.
- **Alignment**: The alignment behavior endows the agent with the ability to align its direction and velocity with other nearby moving agents.
- **Separation:** The separation behavior gives an agent the ability to maintain appropriate distance from its neighbors that belongs to the same group.
- **Separation from other group**: The separation from other group behavior gives an agent the ability to maintain appropriate distance from its neighbors that does not belong to the same group.
- **Guide**: The guide force is a force that guides the agents to its destination.
- **Vector fields**: The vector field is the external forces that are applied to the agent if it travels across the vector fields.

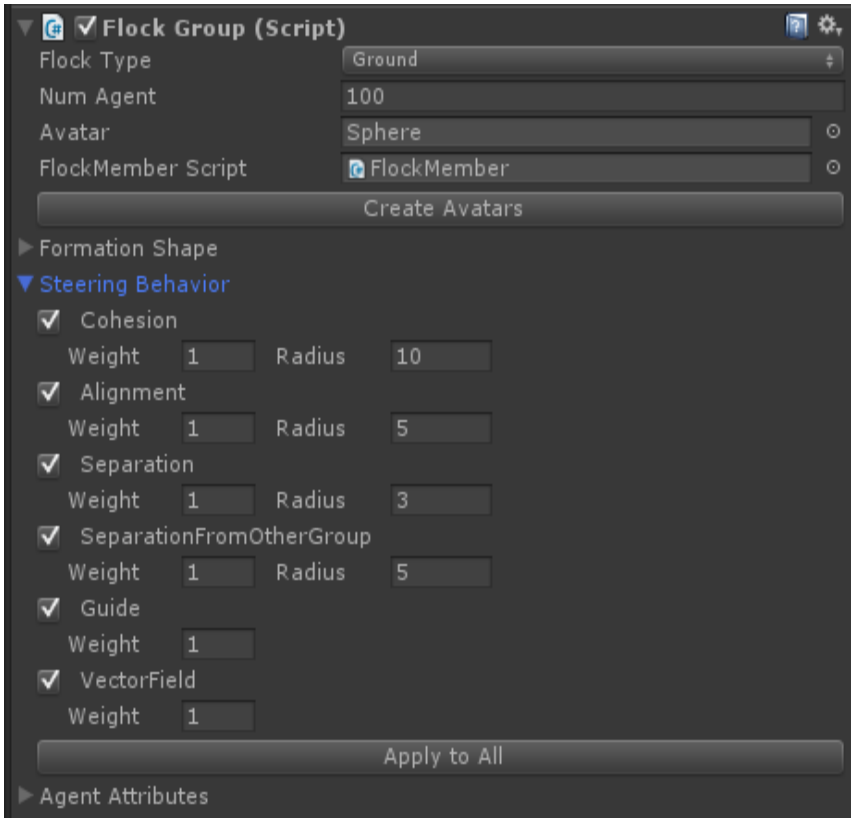

FIGURE 21 CUSTOMIZING STEERING BEHAVIORS OF EACH FLOCKGROUP

You can easily enable or disable each steering behavior by clicking on the checkbox of each respective steering behavior. You can also set the weights and the radius of effect of each steering behavior as well. You may visualize the radius of effect for the active steering behavior on the scene panel (See Figure 22).

After configuring the steering behaviours, click on the "Apply to All" button to assign the behavior to all the FlockMember belonging to the FlockGroup.

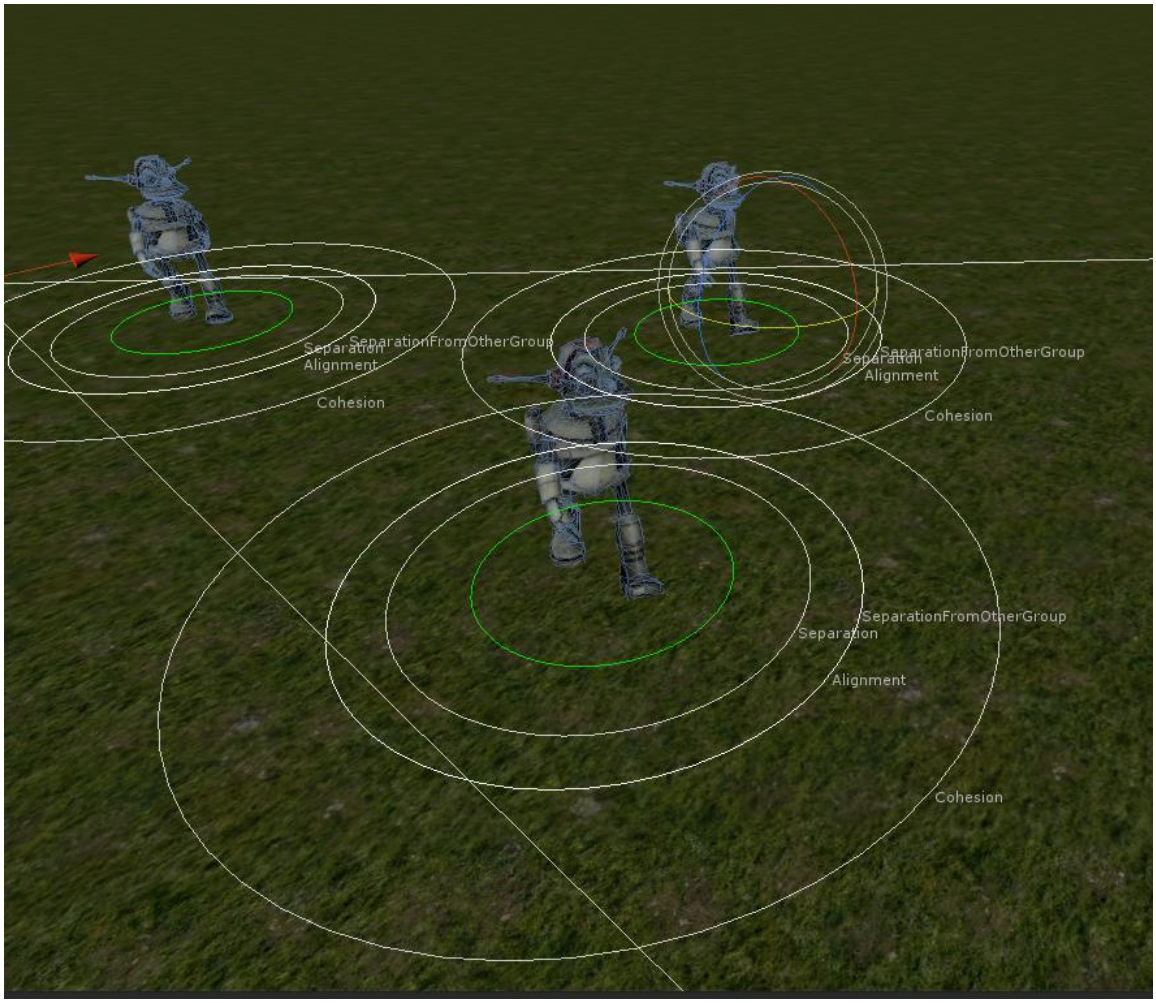

FIGURE 22 SIZE OF ACTIVE STEERING BEHAVIORS IN THE SCENE PANEL

Setting the agent properties for the FlockGroup/FlockMember

Similar to the how you apply the steering behaviors; you may set the attributes of the agent by opening the Agent Attribute foldout (See Figure 23). The meaning of each attribute is as follows:

Parameters	
`AgentRadius`	Radius of the circular space that the agent will occupy.
`AgentMinSpeed`	Minimum speed that the agent will travel.
`AgentMaxSpeed`	Maximum speed that the agent will travel.
`AgentMass`	Mass of the agent.
`AgentMaxForce`	The maximum force that is applied on each agent in each updates

`CollisionFlag`	A 8bit flags that is used to determine whether the agents can travel through the obstacles.
`FieldFlag`	A 8bit flags that is used to determine whether the different types of vectors field in the map will affect the movement behavior of the agent.

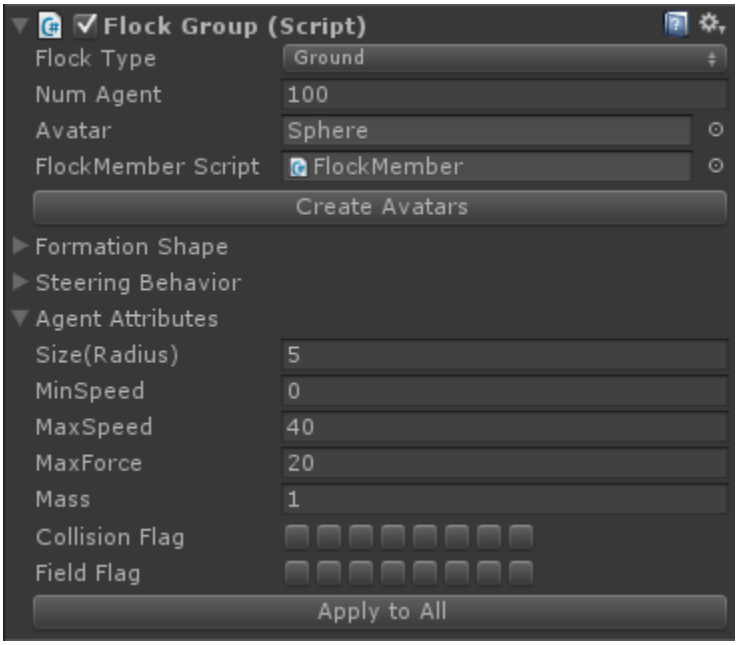

FIGURE 23 CUSTOMIZING AGENT ATTRIBUTE FOR EACH FLOCKGROUP

Similarly, you may visualize the radius, min speed and max speed from the scene panel (See Figure 24).

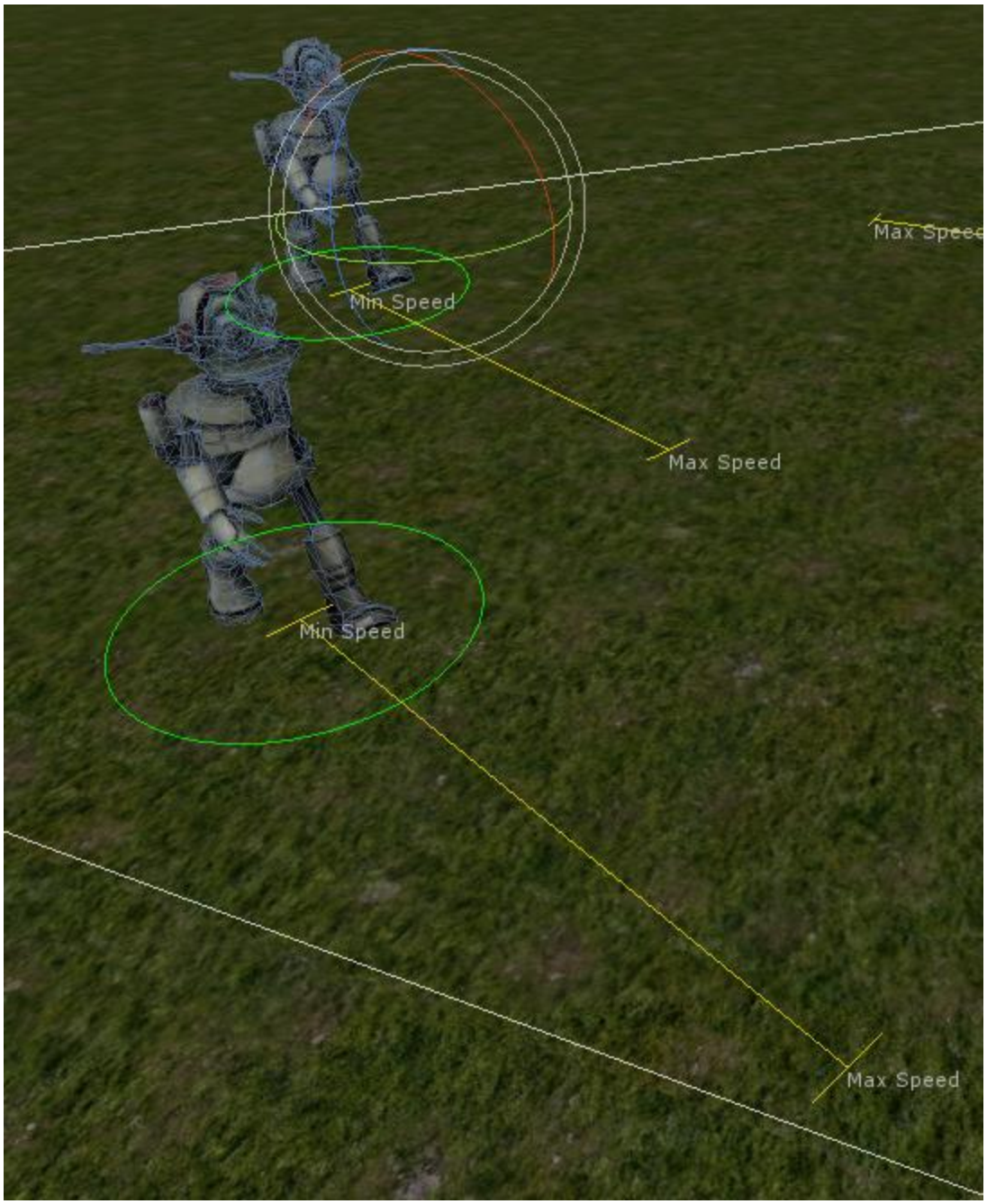

FIGURE 24 VISUALIZING THE AGENT ATTRIBUTE IN THE SCENE PANEL

How to create obstacles in editor?

"`FameObstacle`" is a script used to initialize and create obstacles.

Step 1,

Create an empty object and change its name as described previously.

Step 2,

Drag the "`FameObstacle`" script onto the empty object created.

Step 3,

Set the configurations of the created Obstacle.

The "Collision Flag" is an 8-bit flag parameter used to identify whether a specific agent can pass the specific obstacle. Each agent and obstacle has its own "Collision Flag" parameter. The "collision Flag" parameter in agent is used to indicate which kinds of obstacle it can go through. As for the "Collision Flag" parameter in obstacle, It is used to indicate which kinds of agent can pass it. The agent can pass this obstacle only when the "and (&)" operator of these two parameters is a non-zero value.

As an example, if the collision flag of an obstacle is checked at 0 bit, 5 bit and 7 bit, as shown in Figure 25, it means this obstacle only allows the agents whose collision flag is checked at 0 bit or 5 bit or 7 bit, as shown in Table 1.

FIGURE 25 COLLISION FLAG OF OBSTACLE (CHECKED AT 0, 5, AND 7)

	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0	& Result
Obstacle	1	0	1	0	0	0	0	1	
Agent 1	0	1	0	1	1	1	1	0	0000 0000 (Cannot Pass)
Agent 2	1	0	0	0	0	0	0	0	1000 0000 (Can Pass)
Agent 3	0	0	0	0	0	0	0	1	0000 0001 (Can Pass)
Agent 4	1	1	0	0	1	1	0	0	1000 0000 (Can Pass)
Agent 5	0	1	1	0	0	0	0	1	0010 0001 (Can Pass)
Agent 6	0	1	0	1	0	0	0	0	0000 0000 (Cannot Pass)

TABLE 1 RESULT OF DIFFERENT AGENT COLLISION FLAG

How to create vector fields in editor?

A vector field is an array of vector forces that influences any agent inside the field. It is usually used to represent the effects of some forces in the region, such as river, wind zone, cyclone, etc.

"`FameField`" is a script that used to initialize and create the vector field.

Step 1,

Create an empty object and change its name as described previously.

Step 2,

Drag the "FameField" script onto the empty object created.

Step 3,

Set the configurations of the created fame field. There are two types of fame field. "Uniform" is a polygon field which would add a vector force to each agent in the field, could be used to represent environmental forces such as river, wind zone, etc. "Circular" is a circle field which would add a rotating force or attraction/repulsion force to each agent in the field.

3.2 Using Crowd Simulation API during Runtime

Adding a FlockGroup to the scene

In Crowd Simulation API, flock groups can be created by defining the shape constraints for each group to take up. The shape constraint takes the form of a polygonal shape and is defined by a set of three or more points in the Euclidean space. It can be implemented by calling the `FlockGroup CreateFlock(Vector3[] shape, FlockType flockType, Type flockComponentType, int[] agents = null)` within FameManager class. This method will return a reference of the FlockGroup created.

Each flock group created would be assigned a unique id. Retrieving the `flockgroup` can be easily done by calling `FlockGroup GetFlockGroup(int flockID)` method within the FAMEManager Class.

For example

```
int numCtrlPoints = 4;
Vector3[] points = new Vector3[numCtrlPoints];
points[0] = new Vector3(0, 0, 0);
points[1] = new Vector3(20, 0, 0);
points[2] = new Vector3(20, 0, 20);
points[3] = new Vector3(0, 0, 20);
FlockGroup group = FameManager.CreateFlock(points, FlockType.Ground,
typeof(MyFlockGroup));
FlockGroup retrieveGroup = FameManager.GetFlockGroup(group.FlockID);
```

Populating the FlockGroup

Creating a flock member requires the user to define the type that the member belongs to and the GameObject of the member. Here the type is a class which is FlockMember class or inherits from FlockMember class. This class defines the parameters of the member and provides general methods to be invoked.

It is done easily by calling the `int[] PopulateFlock(int numAgents, GameObject prefab, Type flockMemberType)` method in FlockGroup class. This method will return an array of unique member IDs to identify the created flock members.

Retrieving the flock member can easily done by calling `FlockMember GetFlockMember(int agentID)` method in `FameManager` class.

Note: The flock group that the flock member belongs to must be created before creating a flock member.

For example

```
GameObject memberPrefab = GameObject.CreatePrimitive(PrimitiveType.Cube);
int[] agents = flockGroup.PopulateFlock(1, memberPrefab, typeof(FlockMember));
```

In this code snippet, we create a Cube Prefab, and use it as avatar to create one flock member.

Applying basic transformation to Formation Shape

A FlockGroup formation shape supports three different basic transformations:

1. Translate

   The user can move the FlockGroup formation shape around the scene by specifying a movement vector by calling `void MoveGroup(Vector3 dt)` method in FlockGroup class.

   For example

```
FlockGroup group = FameManager.GetFlockGroup(1);
Vector3 delta = new Vector3(10, 0, 5);
group.MoveGroup(delta);
```

   In this code snippet, we translate the FlockGroup formation shape on a vector (10,0,5).

2. Rotate

   The user can rotate a FlockGroup formation shape on a specified anger by calling `void RotateFormation(float rad)` method in FlockGroup class.

   For example

```
FlockGroup group = FameManager.GetFlockGroup(1);
group.RotateFormation(0.1f);
```

   In this code snippet, we rotate the FlockGroup formation shape with an angle of 0.1 radian.

3. Scale

   The user can scale a FlockGroup formation shape on a specified degree by calling `void ScaleFormation(float scale)` method in FlockGroup class.

   For example

```
FlockGroup group = FameManager.GetFlockGroup(1);
group.ScaleFormation(0.5f);
```

   In this code snippet, we scale the FlockGroup formation shape to half of its current size.

Changing the FlockGroup and FlockMembers Attributes and Steering Behaviors

FlockGroup and FlockMember have a number of common parameters used to define the details of the group and member. They can be categorized into two different parameter sets, 1) agent attributes and 2) steering behavior.

- Agent Attributes:

The parameters consist of the following: `Radius`, `MinSpeed`, `MaxSpeed`, `MaxForce` and `Mass`. You can change these parameters' value by easily change the parameter by calling the respective functions in the FlockGroup.

For example:

```
FlockGroup flockGroup = FameManager.GetFlockGroup(1);
flockGroup.AgentRadius = 4f;
flockGroup.AgentMinSpeed = 10f;
flockGroup.AgentMaxSpeed = 30f;
flockGroup.AgentMaxForce = 10f;
flockGroup.AgentMass = 1f;
```

Note: Once the user change a parameter's value of a FlockGroup, the corresponding parameter of all the FlockMembers belonging to the group will be updated as well.

Changing the parameters of a flock member is similar to changing the parameter of a group. You can change the parameters by setting the desired value directly to the respective parameter.

For Example:

```
FlockMember member = FameManager.GetFlockMember(1);
member.AgentRadius = 5f;
```

- Steering Behaviors:

Adding a steering behavior to a flockgroup can be done by calling the `void AddSteeringBehavior(BehaviorParam[] behaviorList)` method in FlockGroup or FlockMember class. Invoking the AddSteeringBehavior method in the flockGroup component remove all existing steering behaviors and apply new sets of steering behaviors to all its flockmember. Invoking the AddSteeringBehavior method at the flockmember component will apply the behaviour to the flockmember itself only.

For example:

```
BehaviorParam[] behaviorList = {
    new BehaviorParam(SteeringBehaviors.Cohesion,true, 40, 1),
    new BehaviorParam(SteeringBehaviors.Alignment,true, 30, 1),
    new BehaviorParam(SteeringBehaviors.Separation,true, 20, 1),
    new BehaviorParam(SteeringBehaviors.SeparationFromOtherGroup,true, 40,
1),
    new BehaviorParam(SteeringBehaviors.Guide,true, 1),
    new BehaviorParam(SteeringBehaviors.VectorField, true, 1)
    };
FlockGroup flockGroup = FameManager.GetFlockGroup(1);
flockGroup.AddSteeringBehavior(behaviorList);
//Assigning steering behavior to group

FlockMember flockMember = FameManager.GetFlockMember(1);
flockMember.AddSteeringBehaviour(behaviorList);
//Assigning steering behavior to a particular member
```

Similarly, removing a steering behaviour can be done by invoking `void RemoveSteeringBehavior(SteeringBehaviors behavior)` and specify the SteeringBehaviors to be removed.

For example:

```
FlockGroup flockGroup = FameManager.GetFlockGroup(1);
flockGroup.RemoveSteeringBehavior(SteeringBehaviors.Alignment);
flockGroup.RemoveSteeringBehavior(SteeringBehaviors.Cohesion);
flockGroup.RemoveSteeringBehavior(SteeringBehaviors.Guide);
flockGroup.RemoveSteeringBehavior(SteeringBehaviors.Separation);
flockGroup.RemoveSteeringBehavior(SteeringBehaviors.SeparationFromOtherGrou
p);
flockGroup.RemoveSteeringBehavior(SteeringBehaviors.VectorField);

// or simply this

flockGroup.RemoveAllSteeringBehavior();
```

You may also edit the parameters of existing steering behavior by calling the `void EditSteeringBehavior(SteeringBehaviors behavior, float radius, float weight)` method and specify the new values.

For example:

```
FlockGroup flockGroup = FameManager.GetFlockGroup(1);
flockGroup.EditSteeringBehavior(SteeringBehaviors.Alignment, 10,1);
flockGroup.EditSteeringBehavior(SteeringBehaviors.Cohesion, 10, 1);
flockGroup.EditSteeringBehavior(SteeringBehaviors.Separation, 10, 1);

flockGroup.EditSteeringBehavior(SteeringBehaviors.SeparationFromOtherGroup,
10, 1);
flockGroup.EditSteeringBehavior(SteeringBehaviors.Guide, 0, 1); // radius
not needed
flockGroup.EditSteeringBehavior(SteeringBehaviors.VectorField, 0, 1); //
radius not needed
```

Deleting a FlockGroup

Deleting a FlockGroup can be done by simply destroying the gameobject containing the FlockGroup component.

For example:

```
FlockGroup flockGroup = FameManager.GetFlockGroup(1);
Destroy(flockGroup.gameObject);
```

Deleting a flock member

Deleting a FlockMember can be done by destroying the gameobject containing the FlockMember component.

For example:

```
FlockMember flockMember = FameManager.GetFlockMember(1);
Destroy(flockMember.gameObject);
```

Collision Flag and Field Flag

As described previously, “Collision Flag” and “Field Flag” are 8-bit flag parameters that used to identify whether a specific agent can pass the specific obstacle and the specific vector field. Only when the “and (&)” value of the “Collision Flag” of FlockMember and obstacle is true, the agent can pass this obstacle, as well as only when the “and (&)” value of the “Field Flag” of FlockMember and vector field is true, the agent can go through the vector field without vector field’s influence.

The “Collision Flag” and “FieldFlag” of a FlockGroup can be set by assigning the flag value to the `CollisionFlag` and `FieldFlag` in the FlockGroup class respectively. Note that you will have to call the `ApplyCollisionFlag()` and `ApplyFieldFlag()` method to update the values in the Crowd Simulation API library.

For example:

```
byte collisionFlag = 45;    //00101101
byte fieldFlag = 72;        //01001000

FlockGroup flockGroup = FameManager.GetFlockGroup(1);
flockGroup.CollisionFlag = collisionFlag;
flockGroup.FieldFlag = fieldFlag;
flockGroup.ApplyCollisionFlag();
flockGroup.ApplyFieldFlag();
```

Similar to setting the “Collision Flag” and “Field Flag” in FlockGroup, you can assign the desired value to the `CollisionFlag` and `FieldFlag` of the FlockMember.

For example:

```
byte collisionFlag = 45;    //00101101
byte fieldFlag = 72;        //01001000

FlockMember flockMember = FameManager.GetFlockMember(1);
flockMember.CollisionFlag = collisionFlag;
flockMember.FieldFlag = fieldFlag;
flockMember.ApplyCollisionFlag();
flockMember.ApplyFieldFlag();
```

Adding obstacles to the scene

In Crowd Simulation API, one can add an cylindrical obstacle to the scene by calling the `GameObject CreateObstacle(Vector3 position, float radius, Type fameObstacleType)` method in FameManager class. In this method, parameter “position” defines the central position of the cylindrical obstacle; parameter “radius” defines the radius of the round obstacle; and parameter “fameObstacleType” defines the class type of the obstacle. This method will return a reference to the created round obstacle.

For example:

```
Vector3 position = new Vector3(10, 0, 10);
GameObject obj = FameManager.CreateObstacle(position, 8f,
typeof(FameObstacle));
```

Similar to adding a cylindrical obstacle, one can add a polygonal obstacle by calling the `GameObject CreateObstacle(Vector3[] points, Type fameObstacleType)` method in FameManager class. In this method, parameter "points" defines a series of points that composite the formation shape of the polygonal obstacle; and parameter "fameObstacleType" defines the class type of the obstacle. This method will return a reference to the created polygonal obstacle.

For example:

```
int numPoints = 4;
Vector3[] points = new Vector3[numPoints];
points[0] = new Vector3(0, 0, 0);
points[1] = new Vector3(20, 0, 0);
points[2] = new Vector3(20, 0, 20);
points[3] = new Vector3(0, 0, 20);
GameObject obj = FameManager.CreateObstacle(points, typeof(FameObstacle));
```

Each obstacle created would be assigned with a unique obstacle ID. Retrieving the obstacle can easily done by calling `FameObstacle GetObstacle(int obstacleID)` method within the FameManager Class.

```
FameObstacle obstacle = FameManager.GetObstacle(1);
```

Changing obstacle parameters

Changing obstacle parameters requires you to get the reference of the obstacle first. Then you can directly assign the value to any parameter of the obstacle.

For example:

```
FameObstacle obstacle = FameManager.GetObstacle(1);
obstacle.ObstacleType = ObstacleType.Round;
obstacle.ObstacleRadius = 5f;
obstacle.ObstacleType = ObstacleType.Polygon;
List<Vector3> points = new List<Vector3>();
points.Add(new Vector3(0, 0, 0));
points.Add(new Vector3(10, 0, 0));
points.Add(new Vector3(15, 0, 10));
obstacle.ObstaclePoints = points;
```

Setting Obstacle Flag

You can change "Collision Flag" of an obstacle easily by assigning the desired value to the CollisionFlag parameter and call the `void ApplyCollisionFlag()` method in FameObstacle class to update the changes in the Crowd Simulation API library.

For example:

```
byte collisionFlag = 45;    //00101101
FameObstacle obstacle = FameManager.GetObstacle(1);
obstacle.CollisionFlag = flag;
obstacle.ApplyCollisionFlag();
```

Removing an obstacle

Removing an obstacle requires you to get the obstacle ID of the obstacle. You can call the `int PointInObstacle(float x, float z)` method in FameManager class to get the obstacle ID by given the position of the obstacle, then invoke the `void Destroy(Object obj)` method from unity API to remove the obstacle. Destroying the obstacle gameobject will automatically remove the obstacles from FAMEManager.

For example:

```
int obstacleID = FameManager.PointInObstacle(10f, 20f);
if (obstacleID != -1)
{
       FameObstacle obstacle = FameManager.GetObstacle(obstacleID);
       Destroy(obstacle.gameObject);
}
```

Adding vector field to the scene

You can add an uniform field to the scene by calling the `GameObject CreateField(Vector3 position, float width, float height, float magnitude, float forceDirectionAngleRad, Type fameFieldType)` method in `FameManager` class. In this method, the "position" denotes the central position of the uniform field; the "`width`" and "`height`" define the width and height of the uniform field; the "`magnitude`" defines magnitude of the force of the vector field; and the "`fameFieldType`" defines the class type of the vector field. This method will return a reference to the created uniform vector field.

For example:

```
Vector3 point = new Vector3(10, 0, 10);
GameObject obj = FameManager.CreateField(point, 100f, 100f, 2f, 30f *
0.0174532925f, typeof(FameField));
```

Similar to adding a uniform field, you can add a circular field by calling the `GameObject CreateField(Vector3 position, float radius, float magnitude, FameCircularDirection dir, Type fameFieldType)` method in FameManager class. In this method, the "position" parameter defines the central position of the circular field; "`radius`" defines the radius of the circular field; parameter "`dir`" defines field force direction type of the circular field (`Clockwise`, `AntiClockwise`, `Attraction`, `Repulsion`); and "`fameFieldType`" defines the class type of the vector field. This method will return a reference to the circular vector field created.

For example:

```
Vector3 point = new Vector3(10, 0, 10);
GameObject obj = FameManager.CreateField(point, 5f, 2f,
FameCircularDirection.Attraction, typeof(FameField));
```

Each vector field created would be assigned a unique vector field ID to identify the vector field. Retrieving the vector field can easily done by calling `FameField GetField(int fieldID)` method within the FameManager Class.

```
FameField field = FameManager.GetField(1);
```

Changing vector fields parameters

Changing vector field parameters requires you to get the reference of the vector field first. Then you can directly assign the value to any parameter of the vector field. When you are done, call `ApplyFameFieldSetting()` to apply the settings inside Crowd Simulation API.

For example:

```
FameField field = FameManager.GetField(1);
field.FieldType = FameFieldType.Uniform;
field.FieldWidthX = 100f;
field.FieldWidthZ = 100f;
field.FieldAngleDeg = 30;
field.FieldType = FameFieldType.Circular;
field.FieldRadius = 10f;
field.CircularFieldDirection = FameCircularDirection.Attraction;
field.FieldMagnitude = 2f;
byte fieldFlag = 72;        //01001000
field.FieldFlag = fieldFlag;
field.ApplyFameFieldSetting();
field.ApplyFieldFlag();
```

Setting Field Flag

You can change "Field Flag" of an vector field easily by assigning the desired value to the FieldFlag parameter and call the `void ApplyFieldFlag()` method in FameField class to update the changes in the Crowd Simulation API library.

For example:

```
byte fieldFlag = 72;        //01001000
FameField field = FameManager.GetField(1);
field.FieldFlag = fieldFlag;
field.ApplyFieldFlag();
```

Removing a vector field

Removing a vector field requires you to get the vector field ID of the vector field. You can call the `int PointInField(float x, float z)` method in FameManager class to get the vector field ID by given the position of the vector field. Then invoke the `void Destroy(Object obj)` method from unity API to remove the vector field.

For example:

```
int fieldID = FameManager.PointInField(10f, 20f);
if (fieldID != -1)
{
       FameField field = FameManager.GetField(fieldID);
       Destroy(field.gameObject);
}
```

Defining a path (FamePath) in the scene

Creating a FamePath requires you to define a series of points that the path passes through. It done by calling the `int CreatePath(Vector3[] points)` method within FameManager class. This method will return a unique path ID to identity the path.

Retrieving the path can easily done by calling `FamePath GetPath(int pathID)` method within the FAMEManager Class.

For example

```
int numCtrlPoints = 10;
Vector3[] points = new Vector3[numCtrlPoints];
for (int i = 0; i < numCtrlPoints; i++)
{
    points[i] = new Vector3(i * 10, 0, i * 5);
}
int pathID = FameManager.CreatePath(points);
FamePath path = FameManager.GetPath(pathID);
```

Getting & Setting of FamePath Control Points

To get the list of control points that was used to define the path, you may simply do so by calling `Vector3[] GetCtrlPointWorld()` within FamePath. You may also change the path control point position during runtime by calling `void SetCtrlPointPos(int index, Vector3 worldPos)` method.

For Example

```
int pathID = 1;
FamePath path = FameManager.GetPath(pathID);

//Getting the list of control points
Vector3[] ctrlPoints = GetCtrlPointWorld();
Debug.Log("The path has " + ctrlPoints.Length + " control points.");
Debug.Log("The points are:");
for (int i = 0; i < ctrlPoints.Length; i++)
{
    Debug.Log(ctrlPoints[i].ToString());
}

//Setting the ctrl point position
path.SetCtrlPointPos(0, ctrlPoints[0] + new Vector3(0, 0, 1));
```

Performing Query in Crowd Simulation API

Getting a list of FlockMembers/FlockGroups in a selected region

This is probably one of the more commonly used features in your game: to be able to get the list of FlockMembers within a particular region. This feature is similar to how user selects the troops in real-time strategy (RTS) games. This can be done by specifying the area that you would like to query and call the `int[] QueryAgents(Vector3[] polygonShape)` method within FameManager.

For example:

```
Vector3[] selectionPoints = new Vector3[]{
    new Vector3(0,0,0),
    new Vector3(0,0,100),
    new Vector3(100,0,100),
    new Vector3(100,0,0)
};
int[] selectedAgents = FameManager.QueryAgents(selectionPoints);
```

Getting the list of FlockGroups can be done by checking the FlockGroupID that the selected FlockMembers belongs to.

For example:

```
List<int> selectedGroup = new List<int>();
Vector3[] selectionPoints = new Vector3[]{
    new Vector3(0,0,0),
    new Vector3(0,0,100),
    new Vector3(100,0,100),
    new Vector3(100,0,0)
};
int[] selectedAgents = FameManager.QueryAgents(selectionPoints);
foreach (int i in selectedAgents)
{
    FlockMember unit = FameManager.GetFlockMember(i);
    if (!selectedGroup.Contains(unit.FlockID))
    {
        selectedGroup.Add(unit.FlockID);
    }
}
string s = "";
foreach (int flockID in selectedGroup)
{
    s += flockID.ToString() + ",";
}
Debug.Log("Selected group are:" + s);
```

Getting FlockGroup in the selected position

The `int PointInFlock(float x, float z)` method within FAMEManager allows you query the FlockGroup based on the given input position. Here, only the X and Z coordinate is required. Crowd Simulation API will return the id the FlockGroup whereby the formation shape contains the input position. If there are no FlockGroup at the specified position, -1 is returned. Below is an example of how to use the PointInFlock method.

```
Ray ray = Camera.main.ScreenPointToRay(Input.mousePosition);
RaycastHit hit = new RaycastHit();
if (Terrain.activeTerrain.collider.Raycast(ray, out hit, 1000f))
{
    int clickedOnFlock = FameManager.PointInFlock(hit.point.x, hit.point.z);
    if (clickedOnFlock != -1)
    {
        Debug.Log("FlockGroup with id " + clickedOnFlock + " is clicked.");
    }
    else
    {
        Debug.Log("No FlockGroup found.");
    }
}
```

Getting FameObstacle in the selected position

The `PointInObstacle` method within FAMEManager allows you query the FameObstacle based on the given input position. Here, only the X and Z coordinate is required. Crowd Simulation API will return the id the FameObstacle whereby the shape contains the input position. If there are no FameObstacle at the specified position, -1 is returned. Below is an example of how to use the `PointInObstacle` method.

```
Ray ray = Camera.main.ScreenPointToRay(Input.mousePosition);
RaycastHit hit = new RaycastHit();
if (Terrain.activeTerrain.collider.Raycast(ray, out hit, 1000f))
{
    int obstacleID = FameManager.PointInObstacle(hit.point.x, hit.point.z);
    if (obstacleID != -1)
    {
        Debug.Log("FameObstacle with id " + obstacleID + " is clicked.");
    }
    else
    {
        Debug.Log("No FameObstacle found.");
    }
}
```

Getting vector fields (FameField) in the selected position

The `PointInField` method within FAMEManager allows you query the FameField based on the given input position. Here, only the X and Z coordinate is required. Crowd Simulation API will return the id the FameField whereby the shape contains the input position. If there are no

FameField at the specified position, -1 is returned. Below is an example of how to use the `PointInField` method.

```
Ray ray = Camera.main.ScreenPointToRay(Input.mousePosition);
RaycastHit hit = new RaycastHit();
if (Terrain.activeTerrain.collider.Raycast(ray, out hit, 1000f))
{
    int fieldID = FameManager.PointInField(hit.point.x, hit.point.z);
    if (fieldID != -1)
    {
        Debug.Log("FameField with id " + fieldID + " is clicked.");
    }
    else
    {
        Debug.Log("No FameField found.");
    }
}
```

How can a FlockGroup changes it formation its shape?

A FlockGroup may change its current formation shape to another formation defined. Upon calling the morph function, the *flockmembers* with travel toward and fill up the new formation shape uniformly. This is done by calling the Morph function within the FlockGroup class.

For example:

```
int flockID = 1;
FlockGroup flockGroup = FameManager.GetFlockGroup(flockID);
Vector3[] newShape = new Vector3[]{
    new Vector3(0f, 0f, 0f),
    new Vector3(0f, 0f, 10f),
    new Vector3(10f, 0f, 10f),
    new Vector3(10f, 0f, 0f),
};
flockGroup.Morph(newShape);
```

How to change formation shape via control point movement

Another way that a FlockGroup may change its formation shape is by changing the position of the formation control point directly. To change the control point position, just simply call the `void SetCtrlPointPos(int index, Vector3 worldPos)` method within FlockGroup and specify the index of the control point and the desired position in the world coordinate. You may assess the flock's control point by calling `GetCtrlPointWorld()` method within FlockGroup.

For example:

```
int flockID = 1;
FlockGroup flockGroup = FameManager.GetFlockGroup(flockID);
Vector3[] ctrlPoint = flockGroup.GetCtrlPointWorld();

Vector3 point1 = ctrlPoint[1];
point1.x += 10f;
point1.z += 20f;
flockGroup.SetCtrlPointPos(1, point1);
```

How to perform path following for a FlockGroup

In Crowd Simulation API, you can make a FlockGroup travel along a desired path automatically. First, you will have to create a FamePath by specifying a series of points that the spline path will go through. Next, performing the path following can be easily achieved by simply binding the `FlockGroup` to the path. This can be done via calling the `PathFollow(int pathID)` method within `FlockGroup` class.

Binding the path to the FlockGroup can be done by calling the codes shown below:

```
int pathID = 1; // id of the famePath
int flockGroupID = 1; // flockGroup to perform path-following
FlockGroup flockGroup = FameManager.GetFlockGroup(1); // Getting the
flockGroup object
float speed = 10f;
flockGroup.PathFollow(pathID, speed);
```

Note that you will have to align FlockGroup with the head of the path in order to perform path following correctly.

Creating a new sub-formation

Crowd Simulation API also allows you to create a new FlockGroup shape by specifying a list of existing FlockMembers to join this new FlockGroup. There are two way to do so:

1) By specifying the new shape that the ground should be in. This can be done by calling `FlockGroup CreateFlock(Vector3[] shape, FlockType flockType, Type flockComponentType, int[] agents = null)` method and specified the array of FlockMember id in the agents parameter within `FameManager`. After calling this method, the FlockMember will automatically travel towards the new formation shape.

For example

```
Vector3[] pointsToQuery = new Vector3[]{
    new Vector3(0,0,0),
    new Vector3(0,0,100),
    new Vector3(100,0,100),
    new Vector3(100,0,0),
};
int[] agents = FameManager.QueryAgents(pointsToQuery); // Get the list of
agents in the area
FameManager.CreateFlock(typeof(FlockGroup), FlockType.Ground, agents);
// create the flock based on the convex hull, the agents will be transferred
to the new flockGroup
```

2) Let Crowd Simulation API automatically generate the shape based on the convex hull of the FlockMembers' current position. This is done by calling `CreateFlock(Type flockComponentType, FlockType flockType, int[] agents)` method and specify the array of FlockMember id in the agents parameter within `FameManager`.

For example

```
Vector3[] pointsToQuery = new Vector3[]{
    new Vector3(0,0,0),
    new Vector3(0,0,100),
    new Vector3(100,0,100),
    new Vector3(100,0,0),
};
int[] agents = FameManager.QueryAgents(pointsToQuery);
// Get the list of agents in the area
FameManager.CreateFlock(typeof(FlockGroup), FlockType.Ground, agents);
// create the flock based on the convex hull, the agents will be transferred
to the new flockGroup
```

FlockMembers leave a FlockGroup

FlockMembers may detach themselves from the group so that they can roam around freely in the scene. This can be done by calling the LeaveGroup() function within each FlockMember.

For example:

```
Vector3[] pointsToQuery = new Vector3[]{
    new Vector3(0,0,0),
    new Vector3(0,0,100),
    new Vector3(100,0,100),
    new Vector3(100,0,0),
};
int[] agentIDs = FameManager.QueryAgents(pointsToQuery);

for (int i = 0; i < agentIDs.Length; i++)
{
    FlockMember member = FameManager.GetFlockMember(agentIDs[i]);
    member.LeaveGroup(); // flockMember to leave its group
    Vector3 randomPos = Random.insideUnitCircle * 100;
    member.Destination = randomPos; // moving to a random position
}
```

Joining a flockGroup

FlockMembers can also choose to join any other existing FlockGroup in the scene during runtime. Upon joining the group, it will be automatically travel towards its allocated position within the formation shape of the desired FlockGroup. This can be done via calling `JoinGroup(int flockID)` method within FlockMember.

For example:

```
int roamingAgentID = 1;
int flockIDToJoin = 1;
FlockMember member = FameManager.GetFlockMember(roamingAgentID);
member.JoinGroup(flockIDToJoin);
```

Spreading out FlockMembers to fill up the formation shape uniformly

When some of the FlockMembers left or removed from the formation, there will be holes in the formation shape. You can get the rest of the flockMembers in the formation to spread out and fill up the shape uniformly by calling the `void SpreadOut()` function within the FlockGroup.

For example:

```
int flockGroupID = 1;
FlockGroup flockGroup = FameManager.GetFlockGroup(flockGroupID);
group.SpreadOut();
```

3.3 Advance (Class Inheritance)

Inheriting a FlockGroup Object

It is likely that you may want to add in customized features to suit you games requirement. However, it is not recommended to modify the FlockGroup class directly. The recommended way to do so is via inheritance.

Below show an example of how to create an inheritance from the FlockGroup class.

```
using System.Collections.Generic;
using System.Linq;
using System.Text;
using UnityEngine;

public class MyFlockGroup : FlockGroup
{
    protected override void Awake()
    {
        base.Awake(); // this is important!
        // Add your codes here
        // ...
    }

    protected override void Start()
    {
        base.Start(); // this is important!
        // Add your codes here
        // ...
    }

}
```

Creating your customized FlockGroup class via FameManager can be done simply as follow:

```
Vector3[] points = new Vector3[]{
    new Vector3(0,0,0),
    new Vector3(0,0,100),
    new Vector3(100,0,100),
    new Vector3(100,0,100)
}; //the formation shape
FlockGroup flockGroup = FameManager.CreateFlock(points, FlockType.Ground,
typeof(MyFlockGroup));
int[] agents = flockGroup.PopulateFlock(numAgent, groudUnitPrefab,
typeof(MyFlockMember));
//populating the flock with a customised FlockMember class
```

Inheriting a FlockMember Object

Similarly, you may create a customized FlockMember via object inheritance as well. Below shows an example of how it can be created.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using UnityEngine;

public class MyFlockMember : FlockMember
{
    protected override void Start()
    {
        base.Start(); // this is important!
        // Add your codes here
        // ...

    }
    protected void Update()
    {
        // Add your codes here
        // ...
    }
}
```

Creating your customized FlockMember class via FameManager can be done simply as follow:

```
        int numAgent = 40;
        GameObject groundUnitPrefab; // your own object model
        Vector3[] points = new Vector3[]{
            new Vector3(0,0,0),
            new Vector3(0,0,100),
            new Vector3(100,0,100),
            new Vector3(100,0,100)
        }; //the formation shape
        FlockGroup flockGroup = FameManager.CreateFlock(points,
FlockType.Ground, typeof(MyFlockGroup));
        int[] agents = flockGroup.PopulateFlock(numAgent, groudUnitPrefab,
typeof(MyFlockMember));
        //populating the flock with a customised FlockMember class
```