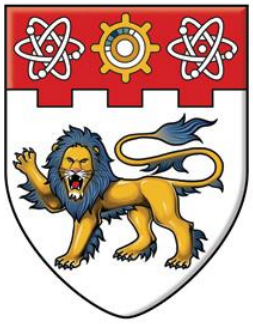


NANYANG TECHNOLOGICAL UNIVERSITY

Master Thesis

Flocking Animation and Modelling Environment: FAME

HO CHOON SING

Supervisor
A/P Ong Yew Soon

SCHOOL OF COMPUTER ENGINEERING

A thesis submitted to the Nanyang Technological University

in partial fulfilment of the requirement for the degree of

Master of Engineering

2012

This page is intentionally left blank

Acknowledgement

The author wishes to express his heartfelt gratitude to his supervisor, Associate Professor Ong Yew Soon, for his patience guidance and generous support throughout the course of his research effort. The author would also wish to extend his gratitude to the staff and buddies in the Centre for Computational Intelligence Lab of Nanyang Technological University (NTU). The encouragement and support by my family and friends are equally well appreciated. The author would also like to thank Singapore-MIT GAMBIT game lab for the funding and support provided for this project.

This page is intentionally left blank

Table of Contents

List of Figures

List of Tables

Abstract

This thesis presents FAME, a comprehensive Flocking Animation and Modelling Environment tool providing soft formation control for large flocks of agents. While many existing available libraries provide means to create flocks of agents equipped with simple steering behaviour, none so far provides an easy and hassle free approach to control the formation of the flock. Here, besides the basic flocking mechanisms, FAME provides an extensive range of advanced features that gives enhanced soft formation control over multiple flocks. These soft formation features include defining flocks in any user-defined formation, automated self-organizing agent within formation, manipulating formation shape at real-time and bending the formation shape naturally along the curvature of the path. FAME thus not only supports the research studies of collective intelligence and behaviours, it is useful for rapid development of digital games. Particularly, the development cost and time pertaining to the creation of multi-agent group formation can be significantly reduced. Using FAME technology, Dark Dot, an iPad game developed in collaboration with Singapore-MIT GAMBIT Game Lab was released in the iTunes app store in Oct 2011. Shortly after launched, it became the top downloaded action game in the iTunes app store for several months and received several positive reviews from the media various game review website, with one notable comment citing Dark Dot as "*not only fun and creative and lovely to look at, but it's got some stunningly original gameplay mechanics.*"

This page is intentionally left blank

1. Introduction

1.1. Background

Advancements in the field of digital games have evolved significantly in the recent decades and becoming ever more complex in game content. Early digital games, due to the limitation of the computer systems, had relatively simple game contents and mechanics. As computer systems are packed with more memory and computational power, game development studios are trending towards creating new generation video games with substantially rich digital contents, while some bearing close resemblances to the real world. Citing the recent popular Grand Theft Auto IV (GTA) video game, for example, which showcases an open world adventure game taking place in a fictional city with over 2000 buildings that is designed based on modern day New York. Thus, modelling of natural and social phenomena including human crowd behaviours and the flocking behaviours within groups of animals is fundamentals to instil the traits of vibrancy and realism in the virtual city.

1.2. Motivation and Objective

While several libraries and source codes pertaining to flock behavioural modelling are readily available on the web, it is noted that most provides only means to achieve simple

steering behaviour. Although these steering mechanisms provides a good underlying building block to achieve natural and realistic motion, these libraries however, do not give high level control over the formation of the group. In the real world, it is easily observed that some animals tend to flock together in some shape or formation, for a common good -- to survive. Hence, a software library equipped with a comprehensive set of tools to allow easy creation of group behavioural animation with flexible controls over the formation shape of the flock would be helpful for games development or collective behaviour related work. This formation constraint is envisioned to be a soft one, such that

1. The agent should be able to temporarily deviate from their original formation when encountering anomalies, such as obstacles, steep terrains, and other agents, and return to its formation when the path is clear.
2. The agent should not be fixed to its assigned position in the formation, but self-organize among themselves to fill up the formation quickly and naturally.
3. The formation should be able to change or moulded according to user preferences during runtime.
4. When the group is navigating in the environment, the formation should bend naturally along with the curvature of the path.

This thesis presents *Flocking Animation Modelling Environment* denoted in short as *FAME,* a soft flock formation library which provides extensive controls over the flock formation. One significant advantage is that development cost and time pertaining to creation of group behavioural animations can be significantly reduced. This library is useful in games development projects, and at the same time, such a toolkit shall also serve to promote and facilitate the research studies of artificial intelligence that pertains to collective intelligence and behavioural studies.

The FAME library is packaged as a dynamic-linked library (.DLL) such that it can be easily deployed in commercial game engines. FAME was developed and tested on a popular commercial game engine, Unity3D. Some key advantages of this game engine and

the reason it was chosen was because of the cross-platform deployment functionality. Games developed on Unity3D could be easily deployed on many various platform such as Apple iOS devices such as iPhone and iPad; Android devices; Windows; MAC and on a web-browser through the Unity3D web player. It has shown that the use of FAME in Unity led to much reduction in the production time of game development pertaining to flock animations.

1.3. Thesis Outline

This thesis consists of six chapters. The current chapter gives a brief introduction about this project. Chapter 2 gives an overview of the related work and the current state of the art in the field of flock behavioural modelling. Subsequently in Chapter 3 describes the work on FAME; a C# game library pertaining to soft flock formation modelling that was developed to simulate the movement of group movement behaviour in a natural and realistic manner. Chapter 4 and 5 each depicts a game example of how FAME can be used in actual game development. In particular, Chapter 4 describes the development details of Meme War; a game designed for artificial intelligence research. Chapter 5 is a short chapter about Dark Dot, a spinoff project in collaboration with Singapore-MIT Gambit Game Lab using FAME, which gained notable publicity from the local media and gaming community worldwide. Finally, a summary of the thesis and concluding remarks can be found in Chapter 6.

2. Literature Review

The art of flock modeling and simulation has received increasing interests and applications in the multi-media and entertainment industry. Particularly in advertising and film production, automating the generation of group animations have been widely pursued as a means of reducing overall production cost or time. This chapter describes the related work and the current state of the arts in the field of crowd behavioural modelling, in particular the flocking behaviour.

2.1. Character Modelling for Computer Graphics

Character modelling in computer graphics (CG) is a tedious and challenging task. There are several steps involved in creating and animating a virtual character that we see in computer games or animation productions. A basic character animation process would require an artist to design and create the geometric model, then build a skeletal rig, and finally animating it. Pushing it further, one could apply physical laws to the virtual body such that it could appear more natural and realistic. Some examples would be the skin deformation process when the character receives a punch in the face and the hair movement as the character swings its head. In order to create a self-animating autonomous virtual character, one would need to model the behaviours of the character. The behavioural modelling process involves representation of environmental stimuli to

computational form and defining the actions that the virtual character should perform upon perceiving these stimuli. Beyond behavioural modelling would be cognitive modelling, introduced by Funge *et al.* [**1**] to this CG modelling hierarchy (see Figure 1), where virtual characters have the ability to learn, reason and plan a sequence of actions to execute in order to accomplish a given goal. Here, a flocking system falls under the behavioural modelling category.

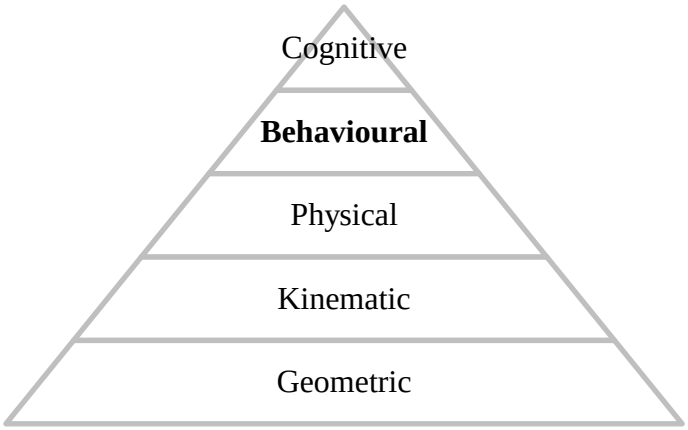


Figure 1: CG Modelling Hierarchy

2.2. Behavioural Models

A popular approach to simulate large crowds of animated characters is through the use of the behavioural model. A behavioural model involves receiving stimuli from the environment and provides an appropriate response [**2**]. In the literature, three core classes of behavioural models can be identified to date to create group movement animation. They are the particle system, the flocking system, and the behavioural system.

2.2.1. Particle Systems

The particle system is a fast, physics based and computationally cheap approach to simulate the movement of large number of agents. Agents controlled using the particle system approach responds to a global force that defines the direction of movement while avoiding collisions. Some studies based on the particle system are listed as follows. Hughes introduces a continuum model which bears close similarities to classical fluid dynamics to study on the flow of a crowd of pedestrians [**3**]. Chenney introduced "flow tiles" [**4**], a tile based method that defines the motion of crowd for each segment that makes up the entire map. These systems are used mainly to simulate the flow of huge and dense crowd of virtual agents and generally do not take into considerations the features of individual agents.

2.2.2. Flocking Systems

The Flocking system, on the other hand, is composed of several simple heuristics rules to simulate the natural and well-coordinated group movement behaviours such as flock of birds, a school of fish and a horde of animals in the virtual computer environment. First published by Craig Reynolds in 1987, Reynolds demonstrated in his *Boids* computer simulation system [**5**] that complex group behaviour can be decomposed into several simple steering behaviours at individual level, by concentrating on local coordination among the agent. He had shown how flocking behaviour is achieved through the use of three simple rules, namely cohesion: to move towards the averaged position of all nearby agents, alignment: to steer and move in the same direction with nearby agents, and separation: to move away from nearby agent that are too close in proximity to avoid collision. This simple computer simulation has thereafter spun-off a vast array of useful real world applications including artificial life simulation, crowd simulation, military

simulation, robotics, movie productions and computer games. The set of possible steering behaviours was later extended to include goal-seeking, obstacle avoidance and path following, fleeing [**6**].

2.2.3. Behavioural System

Agents in a behavioural system are driven by the rules embedded in them. Different from the particle system and the flocking system, the behaviours are guided by the state and intention of the agent rather than local or global tendencies. As such, agents using the behavioural system approach are often seen as being smarter than the flocking system. An interesting research in this area is the simulation of artificial fishes by Tu *et al.* [**7**]. The research group created a virtual marine world simulation, and realistically emulated the appearance, movement and behaviour of the fishes. The behaviour of the fish is driven based on the states of the fish and its intention. Recently, Fridman *et al. a*dopted the social comparison theory in their crowd behavioural model to account for the various social factors influencing the human behaviour [**8**]. Similarly, Pan *et al.* have demonstrated a multi-agent based system which incorporated several social rules to exhibit emergent human social behaviours such as competing, queuing and herding [**9**].

2.3. Recent advancement in flock behavioural model

Recently, Olfati-Saber presents an exposition relating to algorithms and theory in the field of flocking and multi-agent dynamic system [**10**]. Several fundamental questions that a flock model may need to address are highlighted as 1) *Scalability*: how does the model handles massive agents at real-time while maintaining a frame rate of 30fps? 2) *Formation*

control: how does the flock maintain a specific formation? 3) *Obstacle avoidance*: how does a flock avoid colliding with static and dynamic obstacles?

With respect to points 2 and 3, Leonard & Fiorelli [**11**] and Olfati-Saber & Murray [**12**] have described graph approaches for constructing an effective communication model that guarantees collision-free system between multiple vehicles of desired formation. Alternatively, Anderson *et al.* [**13**] considered an iterative sampling method to generate group animations with predefined formation. The method nonetheless was deemed to be too computationally intensive for real-time rendering and they did not consider handling obstacles avoidance during certain phases. Lai *et al.* [**14**] on the other hand was able to produce controlled flock behaviours at significantly reduced computational effort by building group motion graphs with each node representing a cluster of agents. In their work, the form of path and formation constraint are however limited to straight paths and simple planar shapes, respectively. More recently, Xu *et al.* [**15**] proposed a shape constrained flock algorithm to handle complex 2D and 3D shape constrained flocks that move along predefined paths. It is worth noting that most existing works adopt the common process of first sampling the desired formation shape, followed by a path planning stage for each sample points. Subsequently, agents in the flock follow the corresponding sample points, while at the same time exhibiting flocking behaviours.

2.4. Toolkits for flock behavioural animation

Recent research interest in this field of study has been seeking for more efficient ways to achieve more realistic rendering through improved rendering techniques [**16**], refining navigation algorithms to create more natural movement of autonomous agents [**17**,**18**,**19**] and increasing the number of agents that the system can handle in real time through the use of dedicated hardware, specifically the GPUs [**20**,**21**,**22**] and by reducing

computational cost through optimization of the data structure and efficient management of system resources [**23**,**24**]. With the maturing of work in the field, there is an increasing trend to develop toolkits to speed up development process. Recently, Shawn *et al.* developed a set of tools, library and test cases to ease the development effort as well as to measure steering performance of individual agents [**25**,**26**]. Similarly, Erra *et al.* [**21**] developed *BehaveRT*, a GPU-based library to visualize large scales of autonomous agents. However, none of these libraries focuses on formation control in groups of autonomous agents.

3. Flocking Animation and Modelling Environment

This chapter describes the technical details of FAME. FAME is a comprehensive C# game library software library package that allows easy creation of large groups of autonomous agents with soft formation controls within a virtual game environment.

3.1. FAME System Architecture

The system architecture of FAME as depicted in Figure 2 and composed of the following core modules: *FAMEInterface*, *FlockFormationManager*, *AgentManager*, *ObstacleManager*, *PathManager* and *TerrainManager* are described.

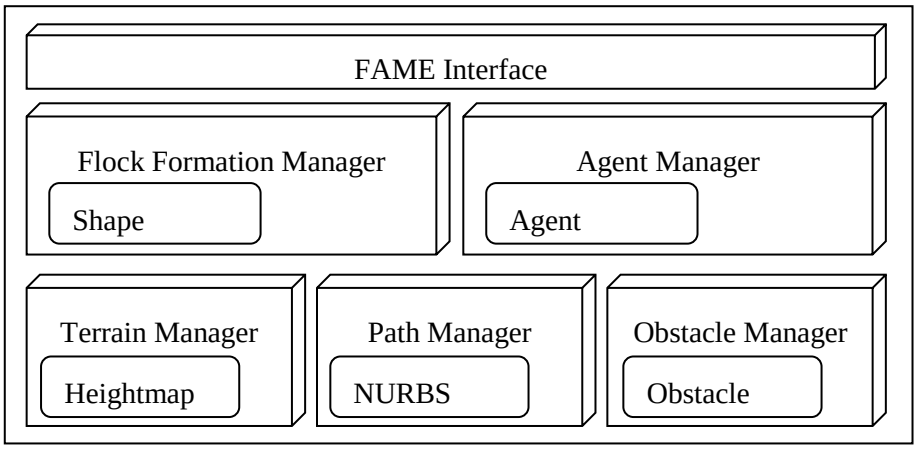


Figure 2: FAME Architecture

FAMEInterface: This is the point of communication between the game application and FAME. It facilitates the creation and removal of flock agents, as well as initialisation the game terrain and environment parameters. Upon initialization, FAME computes the new positions and orientations of each the agent in the flock, for given frame rate of interest.

FlockFormationManager: The flock formation manager manages the creation and removal the flock groups with the desired formations. Here in FAME, each group of agent is bounded by a polygonal shape constraint, i.e. its formation. Each shape constraint is identified by a unique index upon creation and stored within a hash. This is to facilitate fast retrieval of flock object when a query is performed.

AgentManager: The agent manager handles agents at the individual level to endow autonomous and possible emergent behaviours to the flock members independently. Each agent can be identified uniquely and is stored in a hash table upon creation to facilitate fast retrieval of agent object upon query. Each agent is assigned to a flock group at one time. Within each flock group, the agent has a uniquely assigned destination location in the map, which accords to the desired formation defined. The properties of the agent include the mass, maximum and minimum speed, maximum steering angle, and a list of steering behaviour that is applied to the agent.

The individual movement dynamics of each agent in FAME is controlled via several steering heuristics, which includes 1) Cohesion, 2) Alignment, 3) Separation, 4) Guide, and 5) Obstacle Avoidance. These behavioural forces will be accumulated to generate the actual force that acts on the agent. Details of the steering heuristics will be elaborated further in the later sections.

ObstacleManager: The obstacle manager handles the list of obstacles in the scene at real time. Obstacles in FAME can be defined using a circular object or polygonal representation. Upon creation of the obstacles, the obstacle object is stored using a grid data-structure so as to reduce the real time requirements in computational effort when performing queries on them.

PathManager: The path manager handles the list of paths planned within the game environment. In FAME, a path is defined by a set of control points modelled using Non Uniform Rational B-Spline (NURBS).

TerrainManager: The terrain manager handles the terrain related properties in a game. For instance, using the available including terrain size and surface elevation data, it is used to the controls movement of the agents, such the maximum gradient that the agent is allowed to climb and to adjust the movement speed accordingly when the agent travels uphill or downslope.

3.2. Soft formation mechanisms

The soft formation mechanisms provided aims to simulate large groups of agent flocking in a natural and believable manner, while maintaining the desired formation. These mechanisms includes: 1) *defining flock of irregular formation*, 2) *self-organized agent within formation*, 3) *formation shape morphing* and 4) *flexible formation path following*.

3.2.1. Defining flock of irregular formation

FAME 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.

During the shape sampling process, a bounding box is first created and then divided into M squares (where $M >> n$, and n is the number of the flock agents). A random point is subsequently sampled from each square. The unsupervised learning/clustering algorithm is next used to locate n cluster centres. Here, the simple classical K-means clustering is used. The obtained n cluster centres then poses as the positions of the n guiding targets. The pseudo code of the K-means sampling algorithms provided in FAME is outlined as follows:

PSEUDO CODE

X: a set of M sample points

C_I: initialized n cluster centroids

C: the cluster centroids of n-clustering

$P = \{p(i) \mid i = 1, \ldots, M\}$ is the cluster label of X

KMEANS$(X, C_I) \rightarrow (C, P)$

REPEAT

$C_{previous} \leftarrow C_I$;

FOR all $i \in [1, M]$ DO

IF x_i is in constraining shape **THEN**

Generate new optimal partitions:

$p(i) \leftarrow \arg\min d(x_i, c_j)$; $1 \leq j \leq n$

ELSE

p(i)← **NULL**

FOR all $j \in [1, n]$ **DO**

Generate optimal centroids

$c_j \leftarrow$ Average of x_i, whose $p(i) = j$;

UNTIL $C = C_{previous}$ or maximum iteration

Some properties of this shape constraint include a set of control points defining the shape of the flock, a list of sample points defining the agents' position in the formation, the flock centroid and orientation. This facilitates a fast and easy transformation control (translation, rotation, scale) over the flock formation in the virtual environment.

Translation: The group can move around by redefining the flock centroid position.

Rotation: The group can re-orientate to face a particular direction by rotating the control points as well as the sample points.

Scale: The group can expand or shrink the size of the formation by scaling the control points as well as the sample points.

3.2.2. Self-organised agent within 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, FAME 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. Implementation detail is described as follows:

Given an agent A and its randomly picked neighbour agent B, let $\mathbf{P}_A$ and $\mathbf{P}_B$ be the initial position of agent A and B, respectively, while $\mathbf{D}_A$ and $\mathbf{D}_B$ denote the final destinations of agent A and B, respectively. First the following direction vectors to destination are calculated,

$$\mathbf{d}^{AA} = \mathbf{D}_A - \mathbf{P}_A \tag{2}$$

$$\mathbf{d}^{BB} = \mathbf{D}_B - \mathbf{P}_B \tag{3}$$

$$\mathbf{d}^{AB} = \mathbf{D}_B - \mathbf{P}_A \tag{4}$$

$$\mathbf{d}^{BA} = \mathbf{D}_A - \mathbf{P}_B \tag{5}$$

where $\mathbf{d}^{AA}$ is the directional vector from agent A's position to agent A's destination while $\mathbf{d}^{BB}$ is the directional vector from agent B's position to agent B's destination. Similarly, $\mathbf{d}^{AB}$ is the directional vector from agent A's position to agent B's destination and $\mathbf{d}^{BA}$ is the directional vector from agent B’s position to agent A’s destination.

The agents will thus swap their destination position if the following criterion is met.

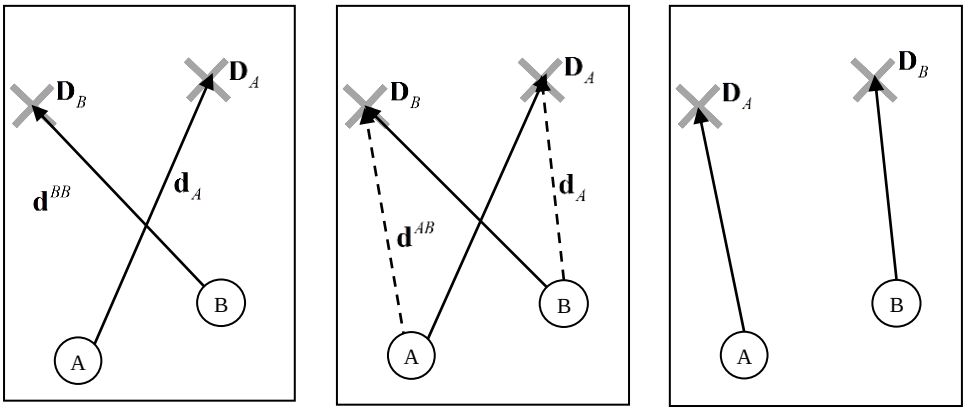
$$\left\|\mathbf{d}^{AA}\right\| + \left\|\mathbf{d}^{BB}\right\| > \left\|\mathbf{d}^{AB}\right\| + \left\|\mathbf{d}^{BA}\right\| \tag{6}$$

which can be computed faster by just comparing the square distance of each vector, such that

$$(\mathbf{d}^{AA} \cdot \mathbf{d}^{AA}) + (\mathbf{d}^{BB} \cdot \mathbf{d}^{BB}) > (\mathbf{d}^{AB} \cdot \mathbf{d}^{AB}) + (\mathbf{d}^{BA} \cdot \mathbf{d}^{BA}) \tag{7}$$

so as to reduce the square root computation when calculating the magnitude of the vector.

It is noteworthy that the above reorganization process does not need to be calculated for every time frame as the agents' position is highly unlikely to change drastically. Rather it is sufficient to calculate periodically after every few seconds to reduce the computational requirements. The algorithm leverage on the neighbourhood calculation process when choosing a random neighbouring agent to swap their destinations, which is already an incurred computation cost needed to compute the steering forces, and hence the additional computation cost incurred is minimal.



a) Calculate $\mathbf{d}^{AA}$ and $\mathbf{d}^{BB}$ b) Calculate $\mathbf{d}^{AB}$ and $\mathbf{d}^{BA}$ c) Swap if criterion met

Figure 3: Criterion for two agents to swap destinations

3.2.3. Formation shape morphing

FAME 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. Here, two formation morphing scenarios are considered: 1) morphing of formation shape from a given initial formation shape to a new user-defined formation shape of interest and 2) a shape deformation approach to changing of formation shape.

Morphing Approach

In the first scenario, the flock of agent changes its current formation to take up a new formation shape. First, a new formation has to be defined. Next, a uniform sampling is performed on the new formation to generate a set of n evenly distributed points inside the shape, where n refers to the number of agents in the initial formation. These points define the new position of that the agents should move to in order to fill up the new formation shape. Finally, a one-to-one mapping process is performed to assign every agent to its new destination position in the new formation shape. The desired mapping should be such that the total Euclidean distance from each agent's current position to its new destination is small. This is to allow the agent to form the new formation within the shortest possible time. The depictions of the mappings result considered in FAME are depicted in Figure 4.

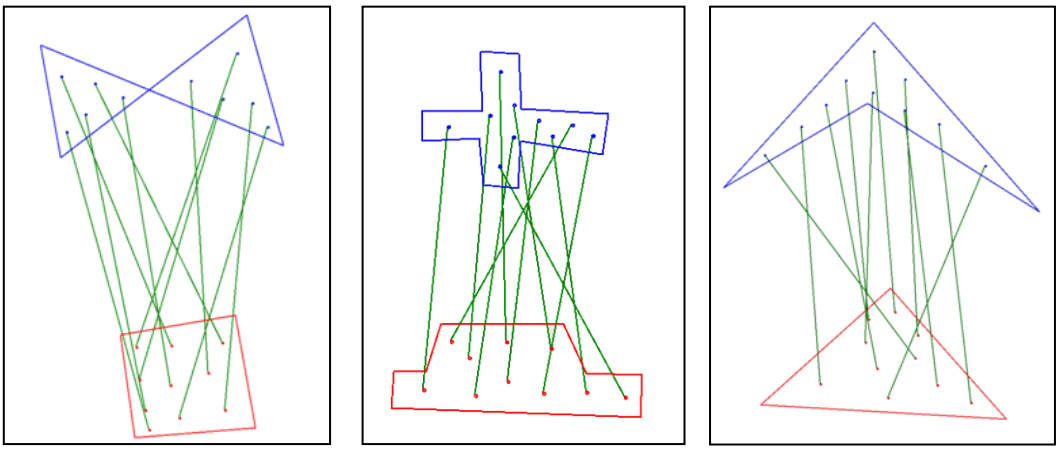


Figure 4: Example of Agent-Destination Mapping (Bottom to top)

Shape Deformation Approach

The second scenario showcases how agents adjust their positions within the formation when it undergoes some shape deformation. The method proposed in previous scenario works well in a one-off computation scenario to morph into a new formation, however it is computational expensive to do so when the changes to the formation shape is minor and continuous over time. Hence, this method is able to complement and make up for the shortfall in the previous method.

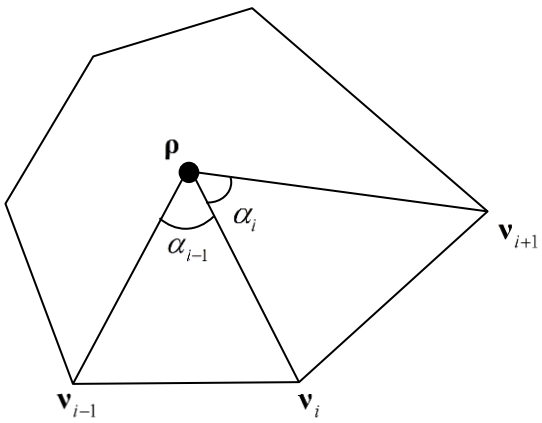


Figure 5: Calculating mean value coordinate for shape constraint

Here, FAME uses the mean-value coordinate [**27**] to calculate the Barycentric weight of each control point on the position of the agents inside the formation. When the initial

shape formation constraint is created and populated with n agents, the Barycentric coordinates is calculated for each of the initial position $\boldsymbol{\rho}^{init} = \{\boldsymbol{\rho}_0, \boldsymbol{\rho}_1, ..., \boldsymbol{\rho}_{n-1}\}$ such that

$$\boldsymbol{\rho} = \sum_i B_i(\boldsymbol{\rho})\mathbf{v}_i \text{ , } s.t. \sum_i B_i(\boldsymbol{\rho}) = 1 \tag{8}$$

where $\mathbf{p} \in \boldsymbol{\rho}^{init}$ is the coordinate of the i control points defining the formation of arbitrary shape (see Figure 5).

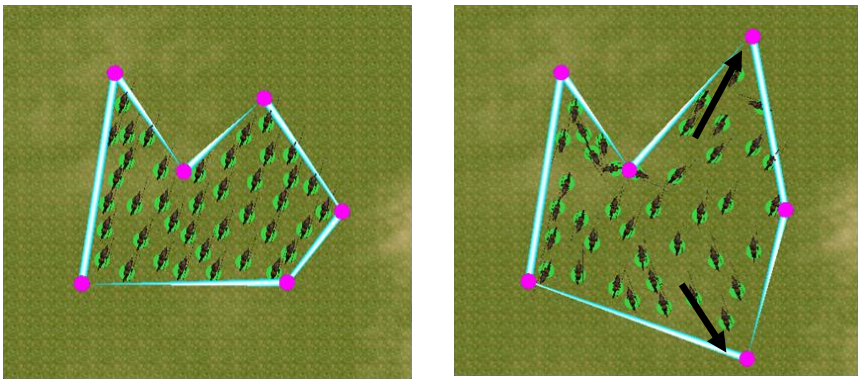


Figure 6: Shape deformation approach to change formation from original formation (left) to new formation (right)

The mean value coordinate $B_i(\boldsymbol{\rho})$ is calculated as follows,

$$B_i(\boldsymbol{\rho}) = \frac{w_i}{\sum_{j=1}^{k} w_j} \tag{9}$$

and

$$w_i = \frac{\tan(\alpha_{i-1}/2) + \tan(\alpha_i/2)}{\|\mathbf{v}_i - \boldsymbol{\rho}\|} \tag{10}$$

where $\alpha_i, 0 < \alpha_i < \pi$ is the angle formed in the triangle $[\boldsymbol{\rho}, \mathbf{v}_i, \mathbf{v}_{i+1}]$.

3.2.4. Flexible formation path following mechanism

The flexible flock formation model tackles the problems of global navigation along a predefined path. At the flock level, the required formation is enforced as a form of soft constraints while at the individual level, flock members are able to react and adapt their own positions when encountering emerged obstacles. It is natural for the formation to bend along the curvature of path while the flock of agents negotiates along the path. Each agent should also adaptively react to the environment so as to avoid obstacles or squeeze through narrow paths and subsequently moving back into the pre-specified formation instead of splitting into smaller groups or losing its formation shape upon avoiding the obstacles.

Paths with large curvature, such as circle or spiral forms, pose great challenges due to the possible consequence of flock formation (especially the corner regions) diverging away from the user pre-imposed path and shape. FAME aspired to attain natural movements with "shape rigging" capabilities in the proposed path-following algorithm to resolve the described inadequacies. The resultant formation is one that bends according to the curvature of the path as depicted in Figure 7.

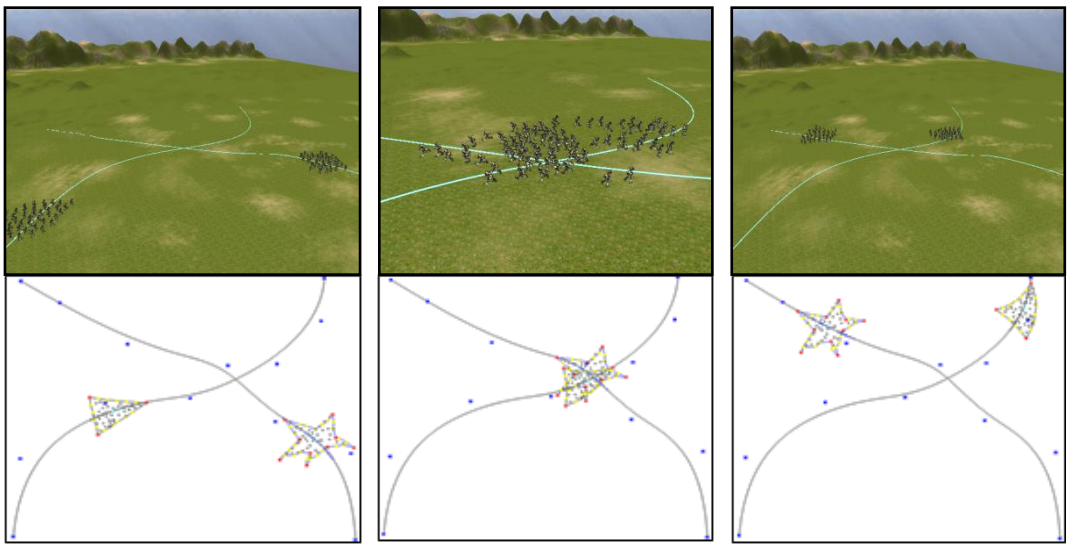


Figure 7: Flock moving in formation along a path

The shape rigging effect is defined as how the formation shape bends naturally along the curvature of the path. A realistic motion of a shape-constrained group moving along a path requires the flock agents to closely follow the curvature of the path, while at the same time fulfilling the defined shape constraint.

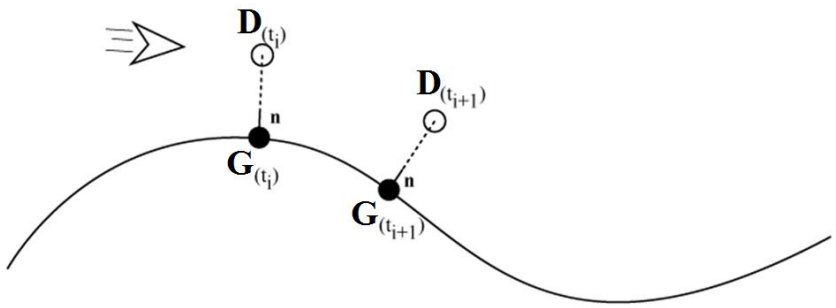


Figure 8: Agent moving along the path's curvature

Here paths are expressed as a function $f(s)$ where s is the displacement w.r.t the starting point of the path. i.e., $\mathbf{G}_{(t_i)} = f(s_{(t_i)})$. The updated position of $\mathbf{G}_{(t_{i+1})}$ at time step t_{i+1} is defined as:

$$\mathbf{G}_{(t_{i+1})} = f(s_{(t_{i+1})}) = f(s_{(t_i)} + \mathbf{v}*(t_{i+1} - t_i)) \qquad (11)$$

where $\mathbf{v}$ denotes the preferred speed of the flock. Let $\mathbf{n}_{(t)}$ and $\mathbf{n}_{(t_{i+1})}$ be the normalized tangent vectors of the curve at the points $\mathbf{G}_{(t)}$ and $\mathbf{G}_{(t_{i+1})}$ respectively, i.e., $\mathbf{n}_{(t_i)} = f'(s_{(t_i)})$ and $\mathbf{n}_{(t_{i+1})} = f'(s_{(t_{i+1})})$, the new position $\mathbf{D}_{(t_{i+1})}$ is obtained by first translating the vector $\mathbf{D}_{(t_i)}$ along $\overrightarrow{\mathbf{G}_{(t)}\mathbf{G}_{(t_{i+1})}}$ direction and subsequently rotating it about $\mathbf{G}_{(t_{i+1})}$ by the angle defined by

difference between the slopes of $\mathbf{n}_{(t_i)}$ and $\mathbf{n}_{(t_{i+1})}$. Let $\mathbf{D}_{(t_i)} = \begin{bmatrix} x_{\mathbf{D}_{(t_i)}} \\ y_{\mathbf{D}_{(t_i)}} \end{bmatrix}$, $\mathbf{G}_{(t_i)} = \begin{bmatrix} x_{\mathbf{G}_{(t_i)}} \\ y_{\mathbf{G}_{(t_i)}} \end{bmatrix}$ and $\mathbf{D}_{(t_{i+1})} = \begin{bmatrix} x_{\mathbf{D}_{(t_{i+1})}} \\ y_{\mathbf{D}_{(t_{i+1})}} \end{bmatrix}$, $\mathbf{G}_{(t_{i+1})} = \begin{bmatrix} x_{\mathbf{G}_{(t_{i+1})}} \\ y_{\mathbf{G}_{(t_{i+1})}} \end{bmatrix}$, the position of the agent at the next time step is as follow :

$$\mathbf{D}'_{(t_{i+1})} = \begin{bmatrix} x_{\mathbf{D}'_{(t_{i+1})}} \\ y_{\mathbf{D}'_{(t_{i+1})}} \end{bmatrix} = \begin{bmatrix} x_{\mathbf{D}_{(t_i)}} \\ y_{\mathbf{D}_{(t_i)}} \end{bmatrix} + \begin{bmatrix} x_{\mathbf{G}_{(t_{i+1})}} \\ y_{\mathbf{G}_{(t_{i+1})}} \end{bmatrix} - \begin{bmatrix} x_{\mathbf{G}_{(t_i)}} \\ y_{\mathbf{G}_{(t_i)}} \end{bmatrix} \tag{12}$$

The rotation is then performed as:

$$\begin{aligned} \begin{bmatrix} x_{\mathbf{D}_{(t_{i+1})}} \\ y_{\mathbf{D}_{(t_{i+1})}} \\ 1 \end{bmatrix} &= \begin{bmatrix} 1 & 0 & x_{\mathbf{G}_{(t_{i+1})}} \\ 0 & 1 & y_{\mathbf{G}_{(t_{i+1})}} \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & -x_{\mathbf{G}_{(t_{i+1})}} \\ 0 & 1 & -y_{\mathbf{G}_{(t_{i+1})}} \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_{\mathbf{D}'_{(t_{i+1})}} \\ y_{\mathbf{D}'_{(t_{i+1})}} \\ 1 \end{bmatrix} \\ &= \begin{bmatrix} \cos\theta & -\sin\theta & x_{\mathbf{G}_{(t_{i+1})}} \\ \sin\theta & \cos\theta & y_{\mathbf{G}_{(t_{i+1})}} \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_{\mathbf{D}'_{(t_{i+1})}} - x_{\mathbf{G}_{(t_{i+1})}} \\ y_{\mathbf{D}'_{(t_{i+1})}} - y_{\mathbf{G}_{(t_{i+1})}} \\ 1 \end{bmatrix} \end{aligned} \tag{13}$$

where θ denotes the angle defined by difference between the slopes of $\mathbf{n}_{(t_i)}$ and $\mathbf{n}_{(t_{i+1})}$. $\cos\theta$, and $\sin\theta$ can be calculated by

$$\cos\theta = \mathbf{n}_{(t_i)} \cdot \mathbf{n}_{(t_{i+1})} \tag{14}$$

$$\sin\theta = \sqrt{1-\cos^2\theta} \tag{15}$$

For the agent to reach $\mathbf{D}_{(t_{i+1})}$, the desired velocity vector is defined in the direction from the current position to the target position as:

$$\mathbf{v}_{desired} = \Delta s / \Delta t = (\mathbf{D}_{(t_{i+1})} - \mathbf{D}_{(t_i)}) / (t_{i+1} - t_i) \tag{16}$$

Finally, the path following force is arrived based on the desired acceleration

$$\mathbf{F}_{formation} = a_{desired} * m = (\mathbf{v}_{desired} - \mathbf{v}_{agent}) * m / (t_{i+1} - t_i) \tag{17}$$

where $\mathbf{v}_{agent}$ is the current velocity vector of the flock agent.

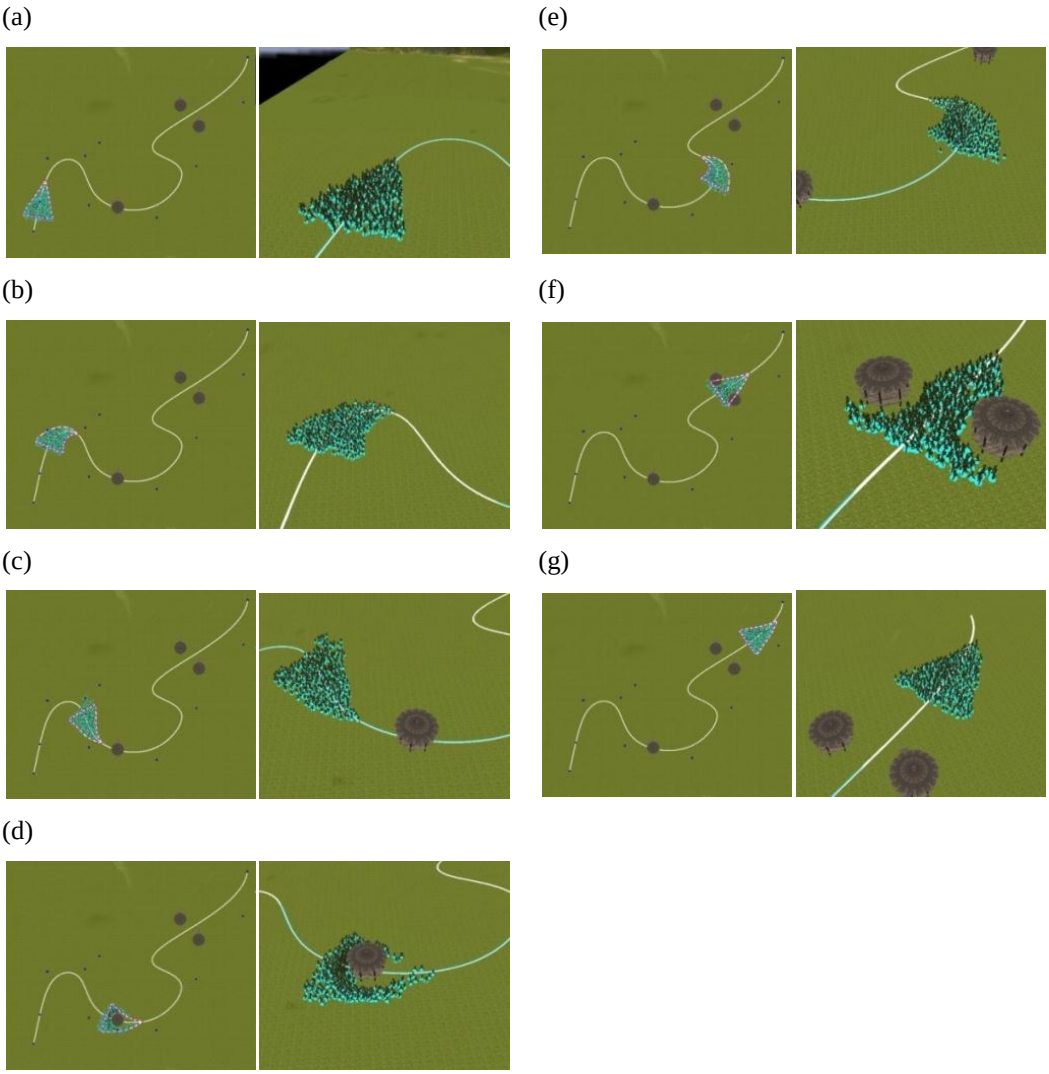


Figure 9: Flock of agent negotiating a path with large curvatures and obstacles

Figure 9a shows the initial setup of the simulation where the agents were created and arranged in the formation shape defined. This experiment was designed to test the model on three different scenarios, 1) navigating on a path with sharp curvature, 2) navigating on a path with newly emerged obstacles that blocks the flock's path and 3) navigating through narrow space.

Initial formation shape **Resultant formation shape on path with large curvature**

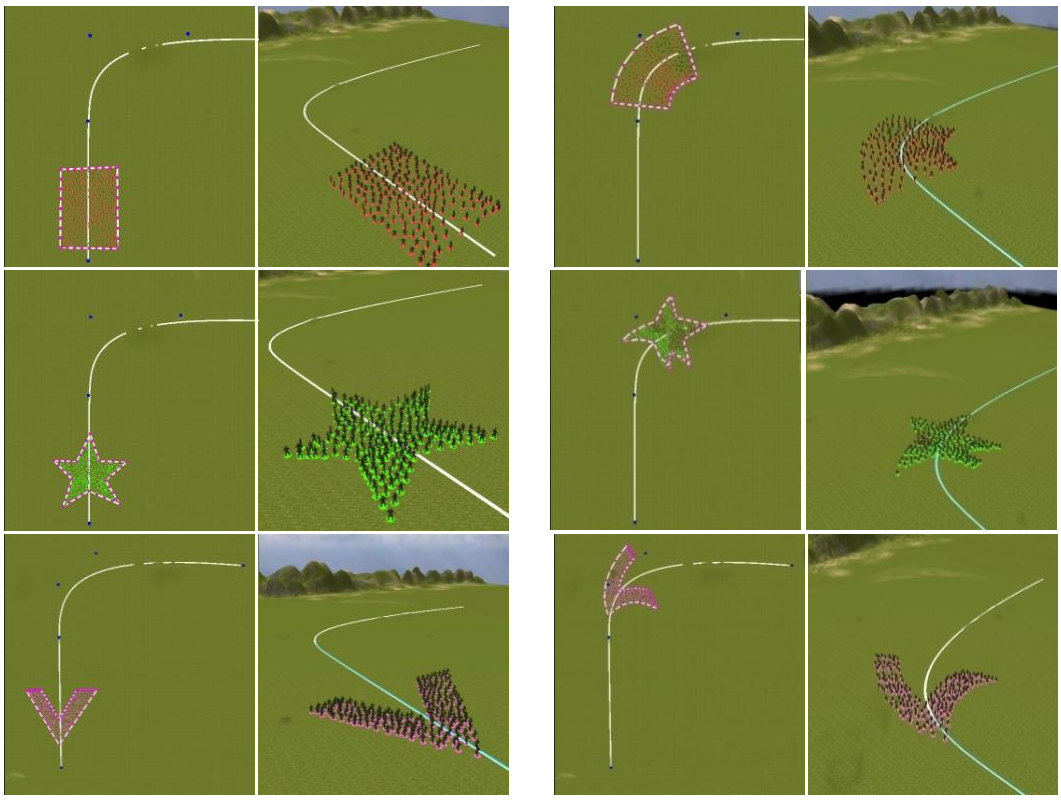


Figure 10: Flexible formation on different shape

As the agent enters the first bend along the path, it is observed that the formation shape is well maintained when the path curvature is gentle. On the other hand, at the path segment with sharp curvature, the formation shape bends naturally, as seen in Figure 9b. Next, when an obstacle is placed dynamically during runtime along the flock's path, it can be observed in Figure 9c and Figure 9d that the agents will automatically disengaged itself from the formation to steer away from the obstacle. The direction that each agent travels is dependent on its relative position and the obstacle. Subsequently, once the flock passed the obstacle, they automatically regroup to its original formation (Figure 9e). Similarly for navigating through narrow space, where two obstacles were placed on the sides of the path as shown in Figure 9f, it can be observed that the agents were able to arrange among themselves to squeeze pass the narrow space. The obstacle avoidance mechanism also acts

as a separation force to prevent flock members from moving too close, and slowing them down if necessary. It is worth noting that the members seamlessly re-join to the original formation after passing the obstacles, as shown in Figure 9g.

The result of the proposed flexible formation model on three different shapes, namely, 1) a rectangular column formation, 2) a star-shaped formation, and 3) a V-shape formation, when navigating along a path with a sharp 90 degree turn, is shown in Figure 10.

3.3. Movement Dynamics of individual Agents

The movement dynamics of the agents in *FAME* are governed by the classical flock model and several additional heuristic rules to achieve various effects. In the classical flock mode, agents are to react to nearby flock mates and make an appropriate adjustment to its movement trajectory. For example, if two agents are too close to each other, they should move apart to avoid collision. To model this perception, each agent is assigned a visible distance. Agents (except itself) that lie within this visible distance are considered as its neighbour. The neighbourhood of an agent is then defined by two parameters, namely, *visible_distance* indicates the proximity for two agents to be regarded as neighbours, while *field_of_view (FOV) angle* defines the perceptual "field of view" of an agent (see Figure 11).

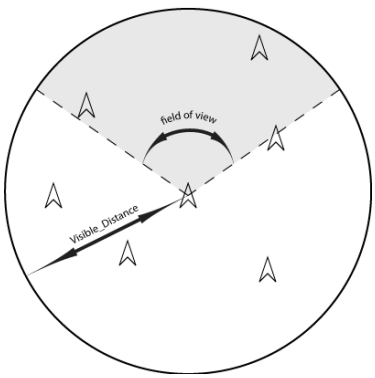


Figure 11: Neighbourhood query with visible range and FOV

The set of neighbour within agent i 's visible range is defined as:

$$S_{neighbors}(i) = \{k \mid k \neq i, \mid \mathbf{P}_k - \mathbf{P}_i \mid < R_i\} \quad (18)$$

where $\mathbf{P}_i$ and R_i denotes the position and the visible distance of agent i, respectively. The computational complexity of neighbourhood query is $O(n^2)$, where n denotes the total number of agents.

3.3.1. Steering behaviours

FAME provides an easy approach to customise the various steering forces to achieve the desired agent behaviour. The steering forces include 1) cohesion, 2) alignment, 3) separation, 4) guide and 5) obstacle avoidance. Developers can easily adjust the radius of effect as well as the contribution weights of each steering heuristics to achieve the desired agent behaviour.

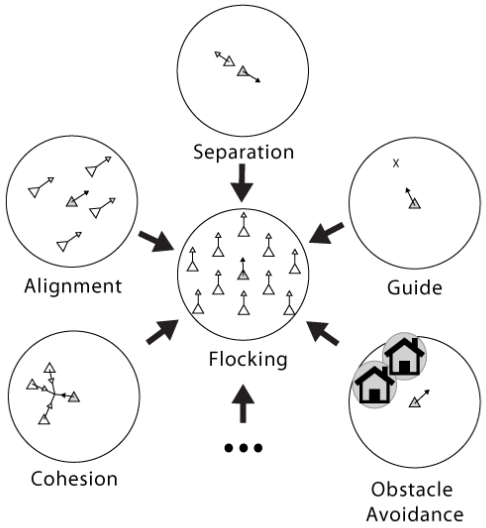


Figure 12: Steering heuristics provided in FAME

Cohesion force

Cohesion behaviour can be found in birds, which tend to fly close to each other. This is probably for communication, navigation and protection (from predators) purposes. To emulate this phenomenon, each agent is modelled with cohesion behaviour. The cohesion behaviour leads the agent to the centre of nearby neighbours. To achieve this behaviour, the cohesion force $\mathbf{F}_{cohesion}$ that attracts the agent i to the "average position" of its local neighbour is defined as:

$$\mathbf{F}_{cohesion}(i) = \sum_{k \in S_{neighbors}(i)} \frac{\mathbf{P}_k}{|S_{neighbors}(i)|} \tag{19}$$

where $S_{neighbors}(i)$ refers to the set of neighbouring agents of agent i and $\mathbf{P}_k$ is the position vector of neighbouring agent k.

Alignment force

Alignment behaviour endows the agent with the ability to align its direction and velocity with other nearby moving agents. Let $S_{neighbors}(i)$ be the set of neighbours of agent i, first the average direction of the neighbouring agents is defined as:

$$\bar{\mathbf{v}}_{neighbors}(i) = \frac{\sum_{k \in S_{neighbors}(i)} \mathbf{v}_k}{|S_{neighbors}(i)|} \tag{20}$$

The alignment force applied on agent i is then defined as:

$$\mathbf{F}_{alignment}(i) = \bar{\mathbf{v}}_{neighbors}(i) - \mathbf{v}_i \tag{21}$$

where $\mathbf{v}_i$ denotes the velocity vector of agent i.

Separation force

Separation behaviour gives an agent the ability to maintain appropriate distance from its neighbours. This avoids both colliding and overly crowded flock agents. Let $S_{neighbors}(i)$ be the collection of neighbourhood agents of agent i, the acting force of separation behaviour is then defined as:

$$\mathbf{F}_{separation}(i) = \sum_{k \in S_{neighbors}} \frac{\mathbf{P}_i - \mathbf{P}_k}{|\mathbf{P}_i - \mathbf{P}_k|^2} \tag{22}$$

where each agent k in the sensing range of agent i contributes a repulsive force $\frac{\mathbf{P}_i - \mathbf{P}_k}{|\mathbf{P}_i - \mathbf{P}_k|^2}$ to the overall repulsive force $\mathbf{F}_{separation}$.

Guide Force

The guide force is a force that guides the agents along its desired travelling path, defined as follows

$$\mathbf{F}_{guide}(i) = \left|\mathbf{D}_i - \mathbf{P_i}\right| * f_{\max} \tag{23}$$

where $\mathbf{D}_i$ is the destination of agent i and $f_{\max}$ is the maximum allowable force that is exerted on the agent at any time-step.

Obstacle Avoidance Force

The obstacle avoidance mechanism serves to drive the agent away from obstacles. *FAME* supports both circular and polygonal shape representations of obstacle meshes. This brings about flexibility and precision to game developers when defining obstacles or regions where flock agents are out of bounds.

Here, polygonal obstacles are defined by a set of points $\mathbf{Q} = \{\mathbf{Q_0}, \mathbf{Q_1}, ..., \mathbf{Q_{n-1}}\}$ whereby the edges of the polygon are denoted by $E = \{\{\mathbf{Q_0}, \mathbf{Q_1}\}, \{\mathbf{Q_1}, \mathbf{Q_2}\}, ..., \{\mathbf{Q_{n-1}}, \mathbf{Q_0}\}\}$ where n denotes the number of points. To avoid collision with the obstacles, each flocking agent A is equipped with a set of three "feelers", i.e., a long feeler directly in front of the agent, while two shorter feelers on each sides (see Figure 13). These feelers serve to steer the agent away from any obstacles before collision can occur.

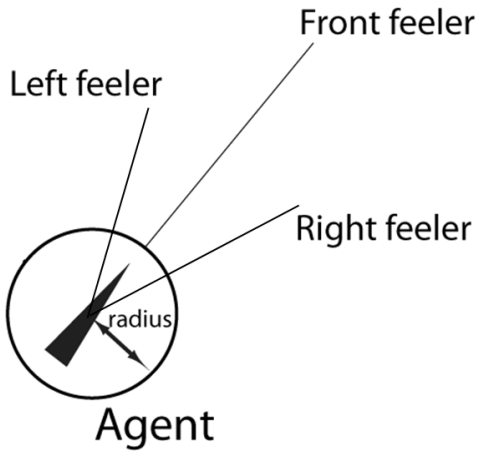


Figure 13: Feelers attached flock agent

The purpose of the side feelers is obvious and intuitive. The left feeler on the other hand, steers the agent to the right when it overlaps with any obstacles. Similarly, the right feeler steers the agent to the left upon overlapping with an obstacle. Here, the feelers are defined as a line segment with the following parametric equation:

$$L(t) = \mathbf{L_s} + t(\mathbf{L_e} - \mathbf{L_s}) \tag{24}$$

where $\mathbf{L}_s$ and $\mathbf{L}_e$ is the start and end point of the line segment respectively, while t is bounded to $0 \le t \le 1$. The starting point of all three feelers is the position of agent *i*, i.e., $\mathbf{P}_i$. The end point or tip of front feeler is then defined by:

$$\mathbf{L}_{e(\,front)} = \mathbf{P}_i + k\hat{\mathbf{o}}_i \tag{25}$$

where k denotes the length of the feelers and $\hat{\mathbf{o}}_i$ is the normalised orientation of the agent *i*. The tip of the side feelers is defined by:

$$\mathbf{L}_{e(sides)} = (\mathbf{P}_i + k\hat{\mathbf{o}}_i) \pm (r_i\hat{\mathbf{S}}) \tag{26}$$

where $\hat{\mathbf{S}}$ is the normalised perpendicular vector of the agent's orientation, i.e., $\hat{\mathbf{S}} \cdot \hat{\mathbf{o}}_i = 0$ and r_i is the radius of the circular space occupy by agent i.

At each time step, collision detection is performed with an intersection test between the feelers and the edges of the polygon. A collision occurs when

$$L(t_a) = E(t_b) \tag{27}$$

where $L(t_a)$ and $E(t_b)$ is the parametric equation of the feeler and the edges of the polygon respectively. By expanding the equation, the following can be derived:

$$\mathbf{L_s} + t_a(\mathbf{L_e} - \mathbf{L_s}) = \mathbf{Q_{n-1}} + t_b(\mathbf{Q_n} - \mathbf{Q_{n-1}}) \tag{28}$$

$$x_{L_s} + t_a(x_{L_e} - x_{L_s}) = x_{Q_{n-1}} + t_b(x_{Q_n} - x_{Q_{n-1}})$$

$$y_{L_s} + t_a(y_{L_e} - y_{L_s}) = y_{Q_{n-1}} + t_b(y_{Q_n} - y_{Q_{n-1}})$$

$$z_{L_s} + t_a(z_{L_e} - z_{L_s}) = z_{Q_{n-1}} + t_b(z_{Q_n} - z_{Q_{n-1}})$$

By solving for t_a, the point of intersection I between the feelers and the obstacles can be obtained when $0 \le t_a \le 1$, otherwise no intersection is deem to have occurred.

The steering force $F_{obstacle}$ employed to steer an agent away from the obstacles is the defined as a sum of the avoidance forces from the three feelers:

$$\mathbf{F_{obstacles}} = \mathbf{F_{frontFeeler}} + \mathbf{F_{leftFeeler}} + \mathbf{F_{rightFeeler}} = \mathbf{F_{frontFeeler}} + \mathbf{F_{sideFeelers}} \tag{29}$$

The obstacle avoidance force for the side feelers, on the other hand, is defined simply as to direct the agent in the opposite side.

$$\mathbf{F}_{sideFeelers} = \begin{cases} \pm\bar{\mathbf{S}} * f_{\max} * (1 - t_a) & 0 \le t_a \le 1 \\ 0 & otherwise \end{cases} \quad (30)$$

where $f_{\max}$ is the maximum force that is applied to the agent in each time step.

For the front feeler, the direction of steering is defined as a function of the angle of incident. First, the normalise vector $\hat{\mathbf{R}}$ orthogonal to the collided edges and heading in the direction of the agent such that $\hat{\mathbf{R}} \cdot (\mathbf{P}_i - \mathbf{I}) \ge 0$, where $\mathbf{I}$ denotes the point of intersection between the feeler and edge (see Figure 14).

$$\mathbf{F}_{frontfeeler} = \begin{cases} \text{sgn}(\hat{\mathbf{R}}.\hat{\mathbf{S}}) * \hat{\mathbf{S}} * f_{\max} * (1 - t_a) & 0 \le t_a \le 1 \\ 0 & otherwise \end{cases} \quad (31)$$

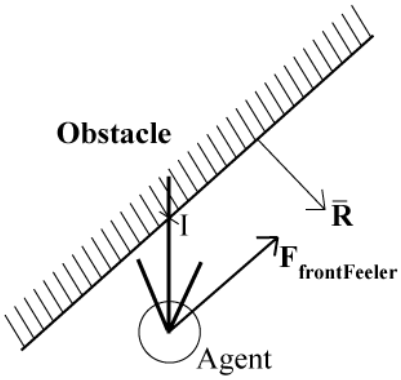


Figure 14: Corrective steering for front feeler

3.3.2. Combining the various forces

The *FAME* library uses the weighted sum of the various force components initiated by various behaviours to calculate the resultant force applied on each agent:

$$\mathbf{F}_{total} = w_1\mathbf{F}_1 + w_2\mathbf{F}_2 + \ldots + w_k\mathbf{F}_k \quad (32)$$

The weight of each individual component can be adjusted to achieve different flocking effects.

In FAME, The movement of the agents is governed by the Newtonian physics. According to Newton's 2nd law, the acceleration of the body is defined by:

$$\mathbf{a} = \frac{\mathbf{F}_{total}}{m} \tag{33}$$

where m is the mass of the body. The velocity of the agent could then be calculated as:

$$\mathbf{v}_{(t_{i+1})} = \mathbf{v}_{(t_i)} + \mathbf{a}(t_{i+1} - t_i) \tag{34}$$

where t_i and t_{i+1} denotes to current and the subsequent time step, respectively.

The new position of the agent is then determined based on the new velocity of the agent and the current position of the agent.

$$\mathbf{P}_{(t_{i+1})} = \mathbf{P}_{(t_i)} + \mathbf{v}_{(t_{i+1})}(t_{i+1} - t_i) \tag{35}$$

3.4. Environmental Constraints

Environmental constraints refer to the set of environmental factors to be considered in the flocking scene, which directly affects the eventual movements of the agents. Examples of environmental factors include terrain conditions, where the agents could be moving slower/faster when travelling downhill/uphill, or wind conditions, where the formation and speed of the flocks can be affected, or water resistance, where the movement of agents in a body of water can be affected by the depth and strength of water current.

3.4.1. Defining Terrain information in FAME

FAME take into considerations the terrain effect on the movement behaviour 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. FAME 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.

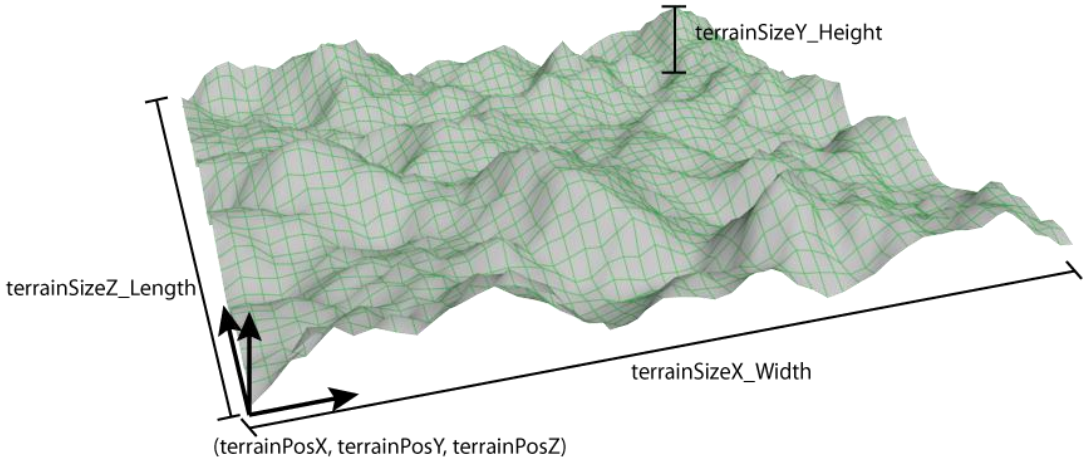


Figure 15: Terrain height information registered in FAME

The speed modification algorithm deployed in FAME follows an efficient and fast algorithm. First, given the position $\mathbf{P}_i$ and the position of the agent *i* at the next time step orientation $\mathbf{P}_i^{Dest}$ of the agent, the slope of the terrain m can be easily determined.

$$m = \frac{y_{P_i} - y_{p_i^{Dest}}}{x_{P_i} - x_{p_i^{Dest}}} \tag{36}$$

Hence, the angle of incline θ can be determined as follow:

$$\theta = \tan^{-1}(m) \tag{37}$$

Here, FAME uses a simple linear interpolation to determine the maximum speed $v_{currentMax}$ of the agent such that

$$v_{currentMax} = \begin{cases} t(v_{\max}) + (1-t)(v_{\max} * \mathrm{M}_{up}) \ \ if\ m \geq 0, \\ t(v_{\max}) + (1-t)(v_{\max} * \mathrm{M}_{down}) \ if\ \ m < 0 \end{cases} \tag{38}$$

where M_{up} and M_{down} is the speed multiplier, a user-defined value, when travelling uphill and down slope respectively.

3.4.2. Emulating environmental forces via Potential fields

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. However, highly accurate and realistic models such as fluid dynamics simulation require high computational power, which is not suitable for applications with real-time visualisation requirement such as games and virtual world simulation. Hence, a fast and seemly realistic model is required. Here, FAME considers potential fields a suitable candidate for emulating these environmental conditions. The advantages of using potential fields can be outlined as such: 1) it is simple and intuitive for any user to use and 2) it is an inexpensive computational alternative to complex fluid dynamics simulations.

Potential 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. FAME provides four different templates of potential fields for ease of use in defining and placing environmental forces within the virtual environment (see Figure 16). These include the uniform field, perpendicular field, attraction/repulsion field and tangential field templates. Fusions of the templates can be easily achieved to arrive at different desired effects.

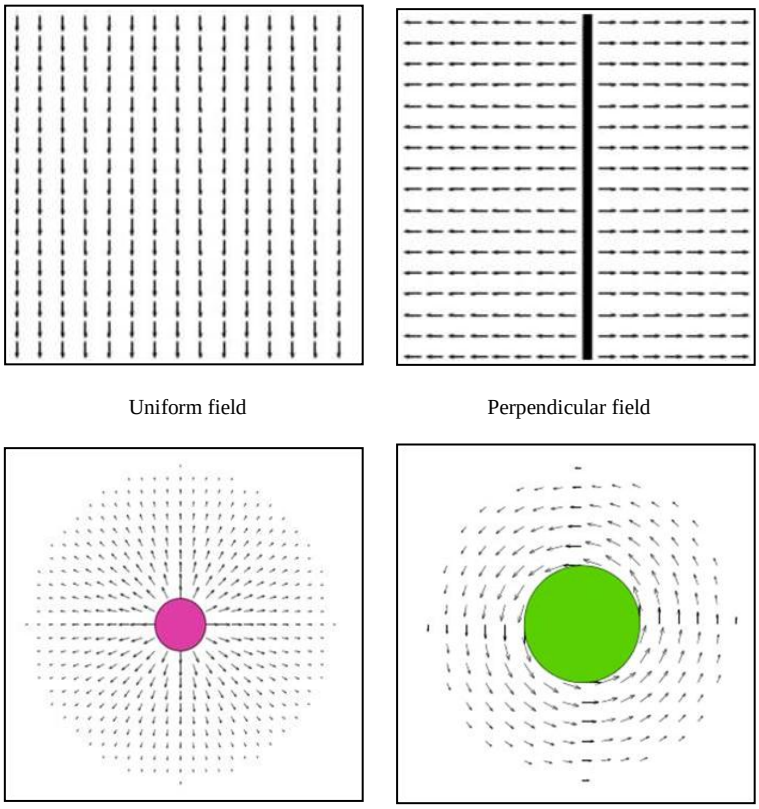


Figure 16: Different potential field templates provided in FAME

Uniform field

In a uniform field, an agent will experience forces of equal magnitude in a particular direction per dimension, regardless of its position within the field. This is represented mathematically with a weight w multiply by a unit vector that defines the desired direction. The weight parameter controls the strength of the potential fields, and is user definable to achieve the desired effects.

$$\mathbf{F}_{uniform} = w_{uniform}\hat{\mathbf{u}} \tag{39}$$

Perpendicular field

The perpendicular field is a special case of the uniform field whereby a centre line is defined and the vectors on each side of the line points either towards or away from the centre line. Mathematically, it is represented as

$$\mathbf{F}_{perp} = \pm w_{perp}\hat{\mathbf{u}} \tag{40}$$

Attraction/Repulsion field

The attraction field has the effect of attracting the agent towards a reference point while the repulsion field as the name implies, has the opposite effect of pushing the agents away from a reference point.

Let (x_0, y_0) be the centre of the field, r be the radius of the field and (x, y) be the position of the agent in the field. The distance between the agent and the centre of the field is computed as follows:

$$d = \sqrt{(x_0 - x)^2 + (y_0 - y)^2} \tag{41}$$

And the angle between the agent and the reference point θ is

$$\theta = \tan^{-1}(\frac{y_0 - y}{x_0 - x}) \tag{42}$$

The attraction/repulsion force $\mathbf{F}_{att} = (f_x, f_y)$ is computed as follows:

$$f_x = \pm w_{att}(x_0 + r - d)\cos(\theta) \tag{42}$$

$$f_y = \pm w_{att}(y_0 + r - d)\sin(\theta) \tag{43}$$

Tangential field

In a tangential field, an agent will experience force that is tangent around the centre of the region of interest. The vectors are thus perpendicular to the radial lines extending outward from the centre. This field can be represented by calculating the equations of the two dimensions.

The tangential force $\mathbf{F}_{\tan} = (f_x, f_y)$ is computed as

$$f_x = -w_{\tan}(x_0 + r - d)\cos(\theta \pm \frac{\pi}{2}) \quad (44)$$

$$f_y = -w_{\tan}(y_0 + r - d)\sin(\theta \pm \frac{\pi}{2}) \quad (45)$$

where d is the distance between the agent and the centre of the field, while θ denotes the angle between the agent and centre of the field.

The tangential field creates a rotation force around the centre in clockwise or anticlockwise direction, defined by the sign shift. This spinning effect is useful for modelling whirlpool effect in the virtual environment.

3.5. Scene partitioning in FAME

FAME provides a seamless and hassle free scene partitioning algorithm to reduce computation requirement, hence supporting a much larger number of agents. Developers are only required to define the size of the map during the initialisation phase and *FAME* automatically prepares the necessary data structures to allow fast retrieval resources when a query is performed.

3.5.1. Hierarchal Partitioning of flock agent into bins of different granularity using Quad tree

In the current model, the flocking behaviour rule requires corrective steering based on actions taken by the agents in the neighbourhood. It is noted that neighbourhood of a given agent is defined as the region within a predefined radius from the agent. A brute force neighbourhood query process, whereby the distance between every pair of agents needs to be calculated, requiring a computation cost of $O\left(n^2\right)$. Such approach does not scale well with large number of agents, i.e., neighbourhood calculation of flock with more than 1000 agents is not possible to be performed in real-time. To improve the computational efficiency and scalability of the current FAME model, a Quad tree data structure is employed to subdivide the game environment into quadrants so that the neighbourhood queries can be performed faster.

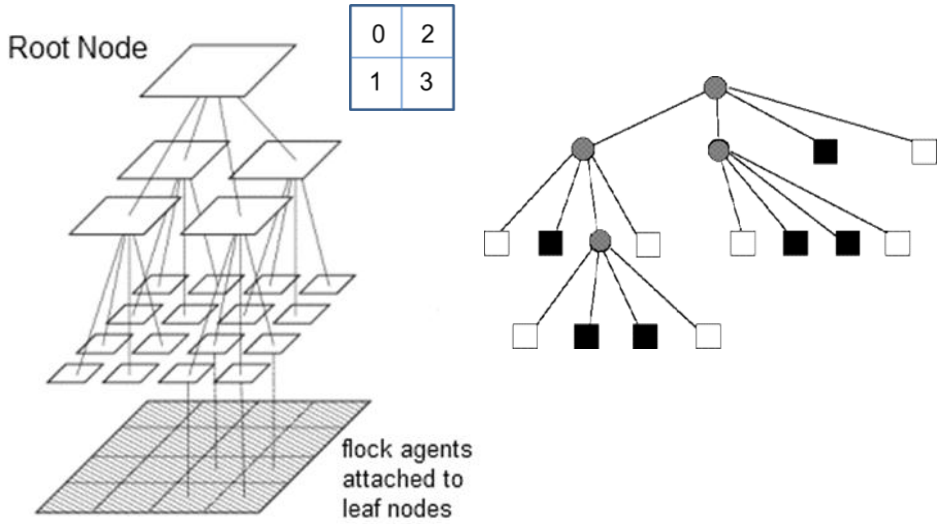


Figure 17: Quad tree data structure

An illustration of the quad-tree structure is depicted in Figure 17, where the neighbourhood query problem is solved by determining which quadrants intersect with the neighbourhood region, via a recursive search from the root node of the tree. In the

example, it can be observed that only quadrant nodes that intersect with the neighbourhood region will be used for updating the next level. The process is repeated until it reaches the leaf node where agents lying within the bound of the node is retrieved and returned as neighbours. This 'divide and conquer' approach brings about a reduction of computational complexity from $O\left(n^2\right)$ to $O\left(n\log n\right)$.

When agents move within the game environment, the leaf node where the agents' positions fall within will be constantly updated. Such updating operation can be done in constant time by traversing the tree top-down from the root node to the leaf node that contains the agents.

3.5.2. Spatial Partitioning of Obstacles

The process to perform collision detection against every polygon edges can be very computational expensive, especially for large polygon and obstacle sizes. Here, a bin data structure to pre-store the positional information of all the obstacles in the scene is proposed to improve the computational efficiency of the system. For a game environment with *n* by *n* rectangular bins, a singly linked-list that stores references to the obstacles that intersect with the bounding volume of the bin (see Figure 18 and Figure 19) is designed for each bin.

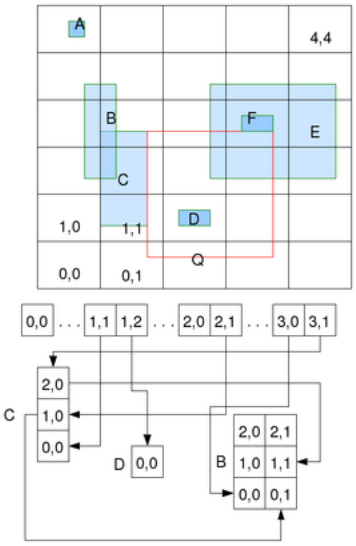


Figure 18: Bin data structure to store references to obstacles

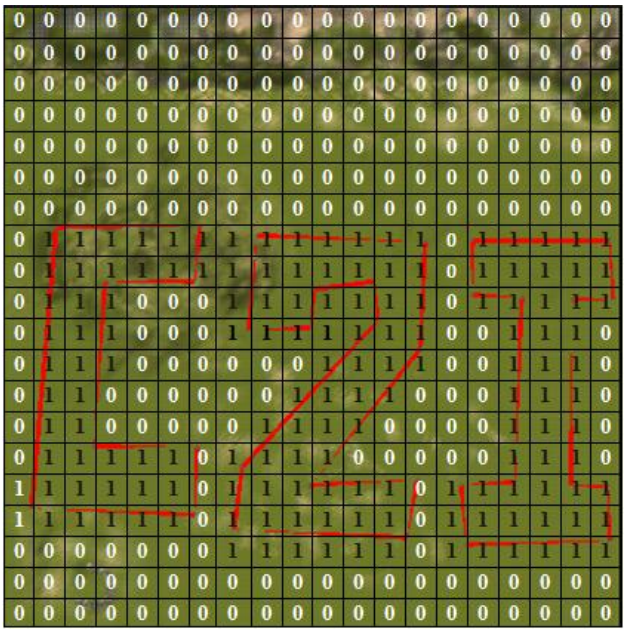


Figure 19: Obstacles forming the word C2I being represented in the bin data structure

Inserting obstacles into a single bin generally takes constant time. For obstacles that span across multiple bins, intersection tests are first performed between the axis aligned bounding box (AABB) bound of the obstacles and the bin, followed by a more precise

intersection test between the actual volume of the obstacles and the bin. Although such a process might be computationally expensive, especially when the obstacle covers a very large area, note that this is computed offline during the initialization process, assuming that most obstacles in the scene are static.

3.6. More information and website on FAME

Interested readers can visit the following website for more information on *FAME*:

`http://c2inet.sce.ntu.edu.sg/fame/`

4. FAME in Game Research Platform - Meme War

To find out how FAME performs in an actual game environment, FAME was deployed in Meme War, a game developed by a team of research staff including the author in the Centre for Computational Intelligence in NTU together with several final year undergraduate students. Here, the FAME library was used to handle the movement dynamics of the game agents. This chapter gives an overview of the game Meme War, and describes how FAME is used in controlling the movement of the agents in the game.

4.1. About the game

Meme War is a medieval style, multi-agent death-match style strategy game designed for artificial and computational intelligence research, developed using a commercial game engine, Unity3D. Figure 20 shows the menu screen of the game. In this game, players will get to choose one of the four agents provided to fight in a death-match scenario. The agents will fight in an arena against everyone else in a time limited battle. Only the most intelligent agent will be able devise a sound strategy to help him survive and destroy its opponent. The units provided are namely archer, swordsman, armoured swordsman and cavalry, each character has its own unique ability and fighting style.

This game is designed to enable casual users, who do not need to have any programming knowledge or any knowledge about AI, to design and embed intelligence

into the game agents. Meme War provides several different types of AI to be embedded into the game agent to govern its overall behaviour to achieve the prescribed game objectives. For the more tech savvy user, they will be able to interface with Meme War with their preferred AI algorithms written in any programming language and assume control over a unit in Meme War via the network interface provided. Meme War is also designed as platform to conduct studies on how game agents can learn and improve in terms of its level of intelligence and believability via imitation from players and other game agents. Hence, the title of the game, Meme War.

Meme War is inspired from the word *Meme* coined by Richard Dawkins in his book titled "*The Selfish Gene*" [**28**]. Meme, analogous to the gene in biological evolution, is defined as a single unit of cultural transmission or imitation that could influence or instil into each other's mind through non-genetic means. The studies on memes within a computational framework belong to the field known as "memetic computing", a paradigm that uses this notion of meme(s) as units of information encoded in computational representations for the purpose of problem-solving. It has achieved relatively high success in tackling complex optimization of large-scale real-world problems in many fields such as bioinformatics, scheduling and routing, aero-dynamic design, machine learning and many more [**29**]. However, it has yet to penetrate into the game industry; a lucrative multi-billion dollar market. In Meme War, meme would represent a single unit of intelligence that could be transferred and assimilated across different agents of arbitrary AI architectures. Through a series of selection, transmission and assimilation process, Meme War aimed to achieve an intelligence agent that would learn and embrace behaviour or traits that would give a perception of human-like intelligence in a believable manner. Ultimately, Meme War strives to explore this new dimension of AI and hopefully it would introduce a more enriching gaming experience for players.

Figure 20: Meme War Main Menu

4.2. Motivation

With the advances in gaming technologies, games developed in recent years are getting more complex and drifting away from the traditional board games style of game play. Nareyet [**30**] hence questioned whether to what extents can techniques developed in the early days be applied in modern computer games. As highlighted in his publication, modern computer games offer a different set of challenges in AI. Take a real time strategy game for example; the real time requirement of the game has left little time for the AI to devise a good and sound strategy. In addition, the AI has to work within the constraints of limited game resources and the incomplete knowledge of the game character, as well as being able to constantly revise and re-plan its strategy in order to adapt to the ever-changing game environment. These are just the tip of an iceberg of the challenges in AI that should be addressed.

Addressing these game AI problems would often require a game platform in order to test and to demonstrate new ideas and techniques. While developing new technologies and algorithms, it is often necessary to have a suitable game platform in order to test, demonstrate and to compare against existing methods. Games development is a time-consuming and painstaking process which involves a considerable amount of effort in many aspects, this includes game design, level design, creation of game and audio assets, coding, testing, debugging and polishing. Moreover, most game development companies often do not release their source codes to the public so as to protect their intellectual property. Hence, it would be beneficial to the research community to develop a game platform designed for research purposes so as to ease development effort.

Thus, Meme War is a game platform designed to facilitate and ease development effort of game AI researchers to conduct studies on agent-based AI. The aim is to develop a game platform that allows researchers to easily conduct artificial and computational intelligence research for use in digital games. As there are many different genres of games available in the market, it is almost impossible to have a game that can be used for all game researches. Hence, the game will be focusing on agent related AI.

4.3. System Design

Meme War is made up by several key components, namely the game agent manager, game manager, Memetic framework library, FAME library, and the web service interface module.

Our project team consists of several members from the Centre of Computational Intelligence in NTU. The author's contribution to this project as follows:

- Design of game logic and game mechanics
- GUI Development

- Integration of various modules developed by other team members and undergraduate students
- Development of FAME library

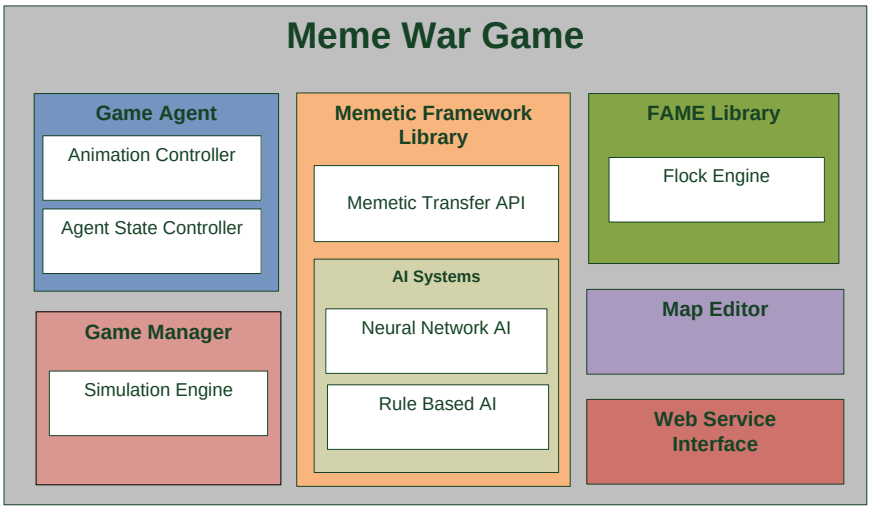
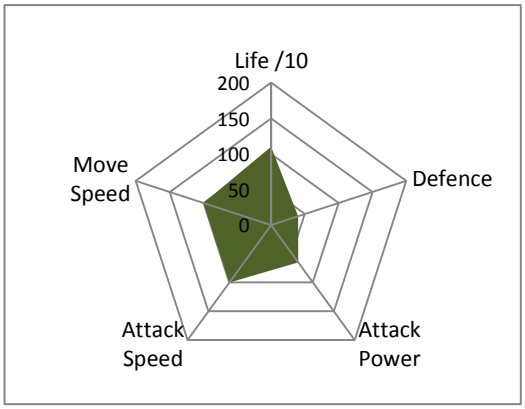


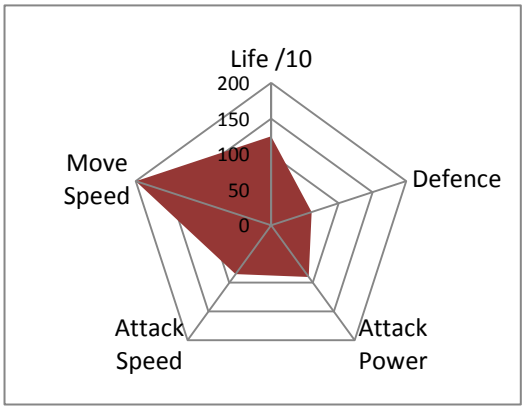
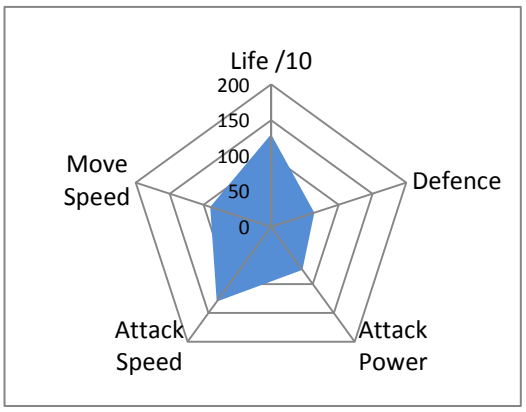
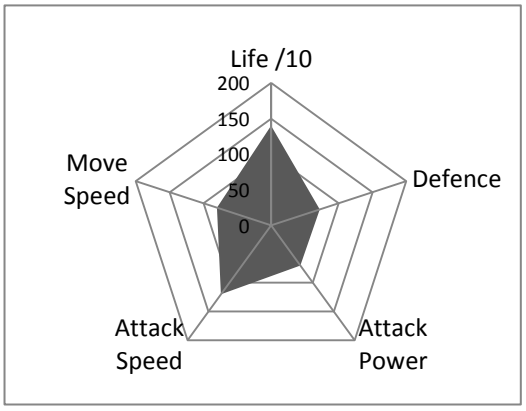
Figure 21: Meme War High Level Architecture Diagram

4.3.1. Game Agent Manager

The game agent manager manages the creation game units. As mentioned in the introduction, Meme War provides four different types of game units to allow the player to choose from. They are the archer, swordsman, armoured swordsman and the cavalry. Table 1 shows a summary of the various unit types provided in Meme War.

Table 1: Various agent type and characteristics in Meme War

Unit Type	Characteristics	Attributes
 **Archer**	Archer is the only unit that uses a long range attack with the ability to nail its enemy down even before they can reach them.	

 **Cavalry**	Cavalry is unrivalled both in terms of movement speed and attack power but slow in attacking.	
 **Swordsman**	Swordsman is a well-balanced close range combat unit. It is the fastest unit in terms of attacking speed.	
 **Knight**	Knight has the highest health and defence but slower attack and movement speed.	

A detailed description of each unit attributes is shown in Table 2.

Table 2: Detailed descriptions of Units Attributes

Attribute Name	Description
Life (L)	Unit's Health Level (Recovering at 3% per every 5 seconds)
Defence (D)	Unit's Armour Level. A higher armour level results in lower damage to taken
Attack Power (A)	Unit's strength. A higher attack power results in higher damage inflicted on enemy unit

Attack Speed	Rate of attack. A higher value means faster attack.
Movement Speed	Maximum speed of the unit.

Agents in Meme War were designed as a simple reflex agent. During gameplay, agents receive stimuli s from the environmental during its decision making process. After which, the selected brain decides on the action α to perform which results a new state s' to be perceived in the next decision making cycle (see Figure 22).

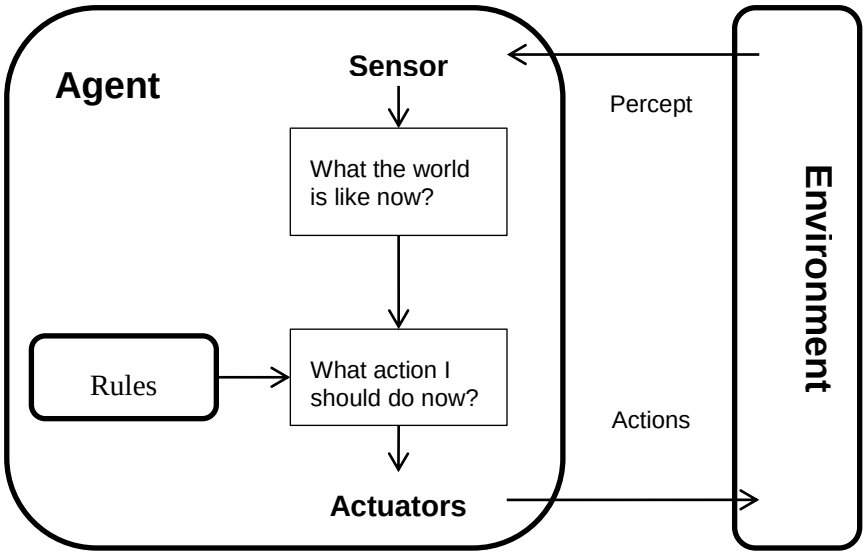


Figure 22: Simple Reflex Agent

The agent perceives the following stimuli from the environment:

Input States:

- *Attacker Count*: Indicate the number of enemy actively attacking any of the agents in the group.
- *Enemy Relative Life*: Indicate the ratio of enemy's current life value compared agent's current life value.
- *Number of Unit*: Indicate the number of agent inside the group.
- *Life*: Indicate agent's current life value compared to its own maximum life.

The actions that the agents can performed are as follows:

Actions:

- *Attack* : Approach and attack the nearest enemy
- *Idle*: Stay in the place and do nothing
- *Escape*: Move away from attacking enemy
- *Approach*: To move towards an enemy

4.3.2. Game Manager

The game manager is in-charge of managing the internal game states and game mechanics. An attack on the enemy agent will dealt a certain damage β calculated based on the *Attack Power* (A) of the attacker and the *Defence* (D) of the enemy agent with a $\pm 5\%$ perturbation.

$$\beta = A^{\text{attacker}}(1 - \frac{D^{\text{enemy}}}{400}) \pm 5\% \quad (1)$$

E.g. When a cavalry attacks an archer, (Cavalry Attack A = 90, Archer Defence D = 40)

Damage $= 90 * (1 - \frac{40}{400}) = 81$.

The game manager also includes a simulation engine to evaluate the performance of the agent in a death-match simulation. During the death-match simulation mode, the agents are graded depending on how well they perform in battle. The scoring system is based on three components: the attack score, number of enemies eliminated and survival. The attacker score is the summation of all the damage dealt in a particular round. For each number of enemies eliminated, a score of 300 points is awarded. The third component, the survival score, is to encourage agents to stay alive till the end of the round. A score of 600 points is awarded to agents that manage to survive the round.

4.3.3. Memetic Framework Library

The memetic framework library consists of two sub-modules, namely the memetic transfer API and the AI modules. The memetic transfer module contains a set of functions to facilitate the transferring of knowledge between agents while the AI module contains several AI frameworks that could be readily plugged in to the agent.

There are currently two different classes of AI systems that are supported by Meme War. One of which is a neural network model that allows an agent to learn the necessary combat strategies from the human players while the players plays the game. The other class is a Rule Based System that allows user to create IF-THEN rules to define the agent's combat strategies.

Neural Network Model

While there are many variants the neural network model that was proposed by researchers over the year, Meme War has considered implementing the neural network model known as Multi Layered Perceptron (MLP) due to its simplicity and ease of development. In a nutshell, MLP is a form of neural network model consisting of multiple layers of nodes in a directed graph whereby the nodes in each layer are fully connected to the next layer (see Figure 23). The MLP utilizes a supervised learning technique known as backward propagation to train the network.

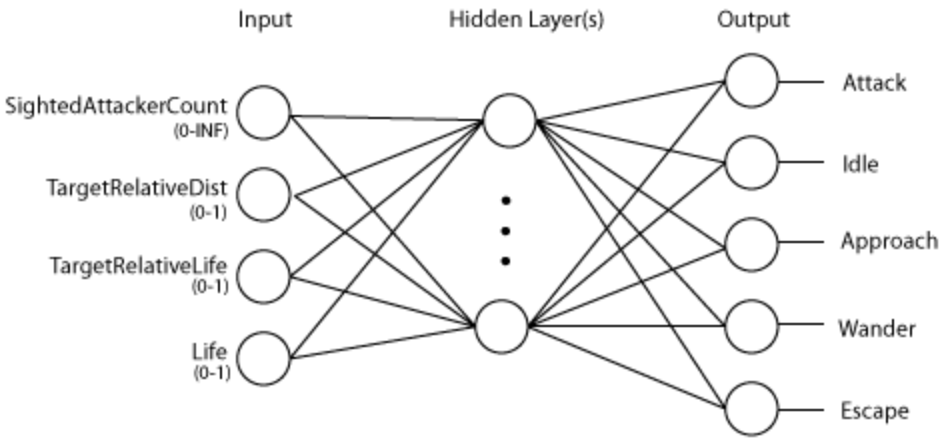


Figure 23: MLP neural network architecture

Rule based System

Meme War also allows players to craft the AI of the agent themselves through the use of condition-action rules. An inference engine is built within Meme War to interpret the rules created. During the gameplay, a rule-based agent will react and execute the defined action accordingly when the rule conditions are satisfied. A priority system is used to resolve conflicts such that the rule with higher priority will be executed when there are multiple rules fulfilling the trigger condition. A rule with a higher priority is executed when multiple sets of rule conditions are met.

While there are no well-established standards for the file format of the rules file, the rule designer studio uses a native rules format in the form of XML. Some advantage of using XML is that it is a platform independent language, well formatted and easily readable by humans, and the XML parser is available in most programming languages.

The structure of the rule format used is as follows. The rule format consist one root element with a *method* tag. A name attribute is defined within this root element to identify the AI type. The various rules created are then appended to the root element. The *rule element* has a name attributed to describe each rule. Inside each *rule element* are a

number of *condition elements* and *action element* which defines the constraints that should be satisfied before the defined action is executed. A sample of the rules XML can be found in Figure 24.

```
<?xml version="1.0" encoding="utf-8"?>
<method name="SimpleRules">
  <Rule name="Basic Escaping">
    <Condition name="Life" type="double" clause="LE" value="0.7" />
    <Action name="Escape" value="4" />
  </Rule>
  <Rule name="Basic Attacking">
    <Condition name="EnemyLife" type="double" clause="GT" value="0.7" />
    <Action name="Attack" value="1" />
  </Rule>
</method>
```

Figure 24: Sample rule format in XML

4.3.4. FAME Library

The movement mechanisms of the game units in Meme War are handled by the FAME library. The steering behaviours and parameters set on each agent are described in Table 3.

The effective range for the separation behaviour is set accordingly based on the action selected by the agent. The reason for having a variable separation effective range is due to the nature of the action itself. Take for example the attack action, having a 0 effective range will effectively turn off the separation behaviour as the agent is required to move close to its enemy in order to attack, while the escape behaviour having a 2.0 effective range will automatically steers the agent away from colliding agents. Figure 25 show a relative comparison between the size of the agent with the effective radius of the steering behaviours.

Table 3: Steering Behaviours and Parameters

Agent Type	Steering Behaviours	Weights	Effective Range
All	Cohesion	1.0	5.0
	Alignment	1.0	5.0
	Separation	2.0	0.0(Attack) 1.0(Idle) 1.5(Approach) 1.5(Wander) 2.0(Escape)
	Guide	1.0	-
	Obstacle Avoidance	1.0	-

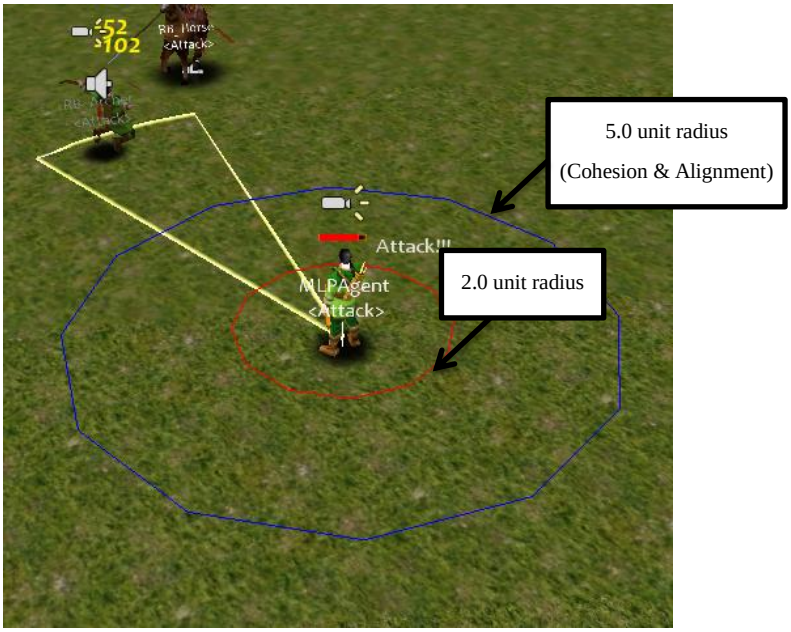


Figure 25: Steering behaviour effective radius

4.3.5. Web Socket Interface

Meme War would be a useful game platform for AI researchers to compare with existing AI methods or to determine the performance of newly developed AI ideas or algorithms. Having that in mind, Meme War is designed to include the above feature by exposing

some of the game variables to external applications and allowing them assume full control over an agent in Meme War during battle through the use of a network socket port. In this way, users may write their AI methods in any programming language desired and gain control of a Meme War agent through the program interface provided.

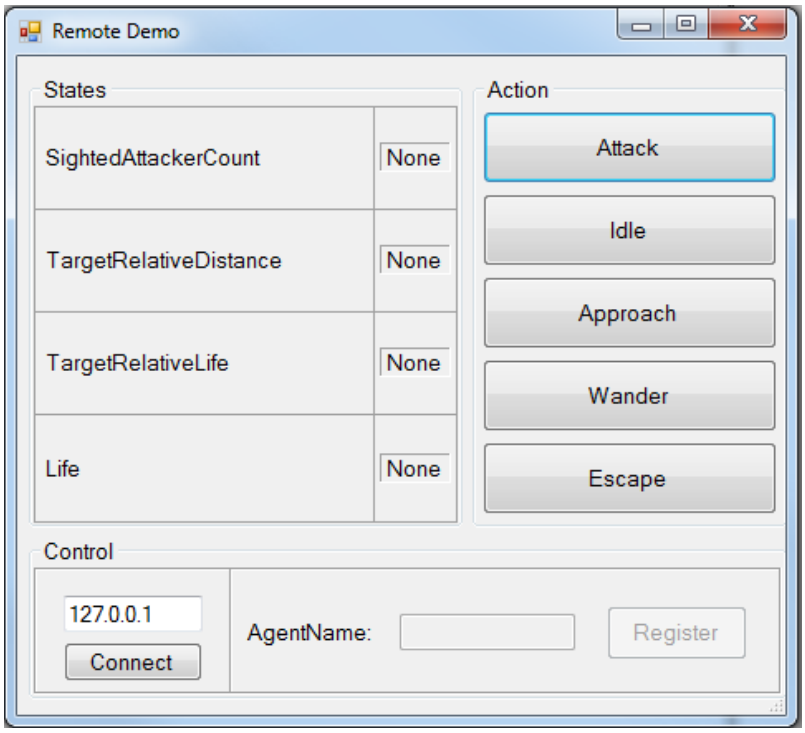


Figure 26: Remote Demo Program

A simple remote controller application has been developed to illustrate the functionality (see Figure 26). The Meme War remote function is illustrated in death-match simulation mode. Before the battle simulation starts, remote applications can register their agent through the program interface provide in Meme War. During the simulation, Meme War will be sending the sets of input that the agent perceives upon completion of every action to the remote application. The remote application should process the input and determine the appropriate action to be taken by the agent. This output action is then transmitted back to Meme War through the same connection port. Meme War will be able to read in output and reflects it according. This cycle repeats until the simulation ends. Tech savvy users

should be able to easily program their own remote application as long as they conform to the Meme War socket interface provided.

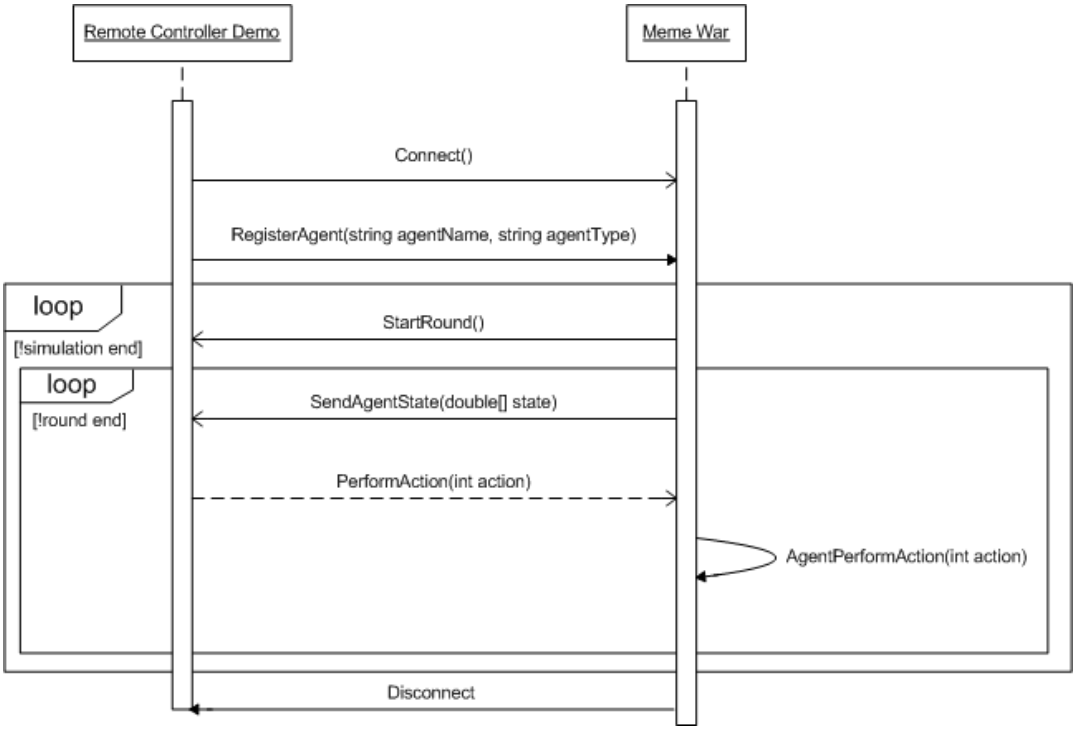


Figure 27: Remote Controller - UML Sequence Diagram

4.4. Game Features

This section describes some of the game features provided in Meme War, namely the different the game modes provided; the Rule Designer Studio, a graphical application to allow user to define rules for the game agent and a Map Editor to allow players to design the map.

4.4.1. Game Modes

Death-match Simulation Mode

An agent that has been trained or rules scripted can be put into a death-match arena where the agent will be put into battle, to fight each other. In the arena, the player does not control any of the agents. The game agent will react according to how it has been trained or configured. Users can specify the number of rounds and the duration of each battle. Figure 28 shows a screenshot captured during the simulation.

The battle is of a one-versus-all nature, where an agent will be fighting all other agents in the arena. The agent's performance is measured using a scoring system. The score will be added for each successful hit dealt to enemy, while getting hit by the enemy will decrease the score. In the end of the arena mode, the score of each agent for each round will be summed up and the average score will be produced. This score is used to assess the performance of the agents.

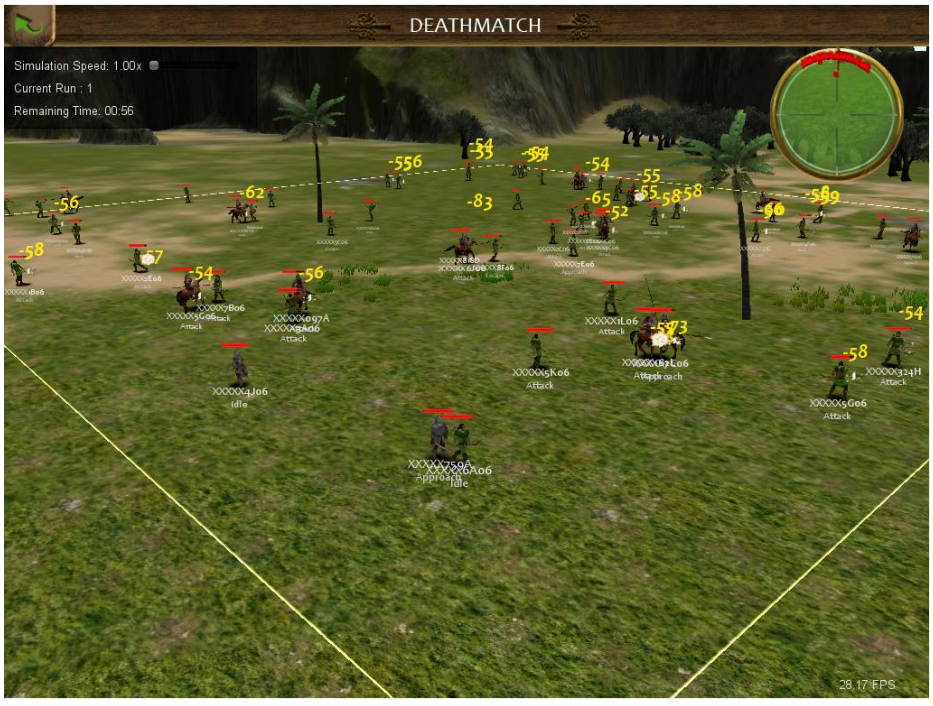


Figure 28: Death-match mode screenshot

Training Mode

The training process of the agent's neural network brain in Meme War consists of two stages. First is the recording stage and next is the training stage.

Figure 29: Training mode screenshot

During the recording stage, user has to take control and play the game. While the user plays the game, Meme War quietly records the states of the agent and action performed by the user in the background (see Figure 29 for a screenshot of the recording process). The data recorded during gameplay are labelled as training records. The training records are saved in a human readable format in XML. After the user is satisfied with the number of training record gathered, these training records could then be used to train the agent's brain. Figure 30 shows a snippet of the training records that where recorded during the gameplay.

```
<?xml version="1.0"?>
<DiscreteRecords>
  <record>
    <input index="1" name="SightedAttackerCount" value="0.0000" />
    <input index="2" name="TargetRelativeDistance" value="1.0000" />
    <input index="3" name="TargetRelativeLife" value="0.5000" />
    <input index="4" name="Life" value="1.0000" />
    <output index="3" name="Approach" value="1.0000" />
  </record>
  <record>
    <input index="1" name="SightedAttackerCount" value="1.0000" />
    <input index="2" name="TargetRelativeDistance" value="0.8991" />
    <input index="3" name="TargetRelativeLife" value="0.5409" />
    <input index="4" name="Life" value="0.8591" />
    <output index="1" name="Attack" value="1.0000" />
  </record>
…
</DiscreteRecords>
```

Figure 30: Sample training record in XML

As for the training stage, Meme War first reads in the training records and uses it to train the intelligence of the agent. Note that only the neural network type brain requires training while the logical based AI such as rule based system does not require any training. After the training is completed, the trained neural network will be automatically saved in the game and can now be used for battle in the death match mode.

4.4.2. Rule Designer Studio

The Rule Designer Studio was developed to ease the rule creation process. This piece of software is designed to be generic stand-alone application to allow future extension to other projects as well. Users can easily construct the rules for the agent through a series of drag and drop function through the graphical interface provided. The graphical interface is inspired by MIT's *Scratch* software (see Figure 31), a piece of software designed to allow

pre-teens users to develop a simple program and animations by defining the actions of the characters in the scene through the use of graphical jig-saw look-alike building blocks.

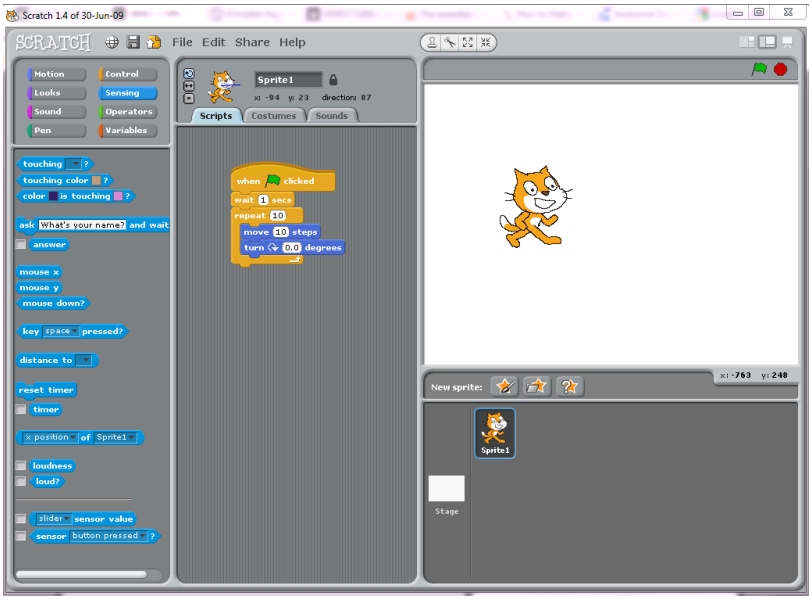


Figure 31: MIT's Scratch

The Rule Designer Studio consists of three panels: the profile panel, the design panel and the XML panel (see Figure 32).

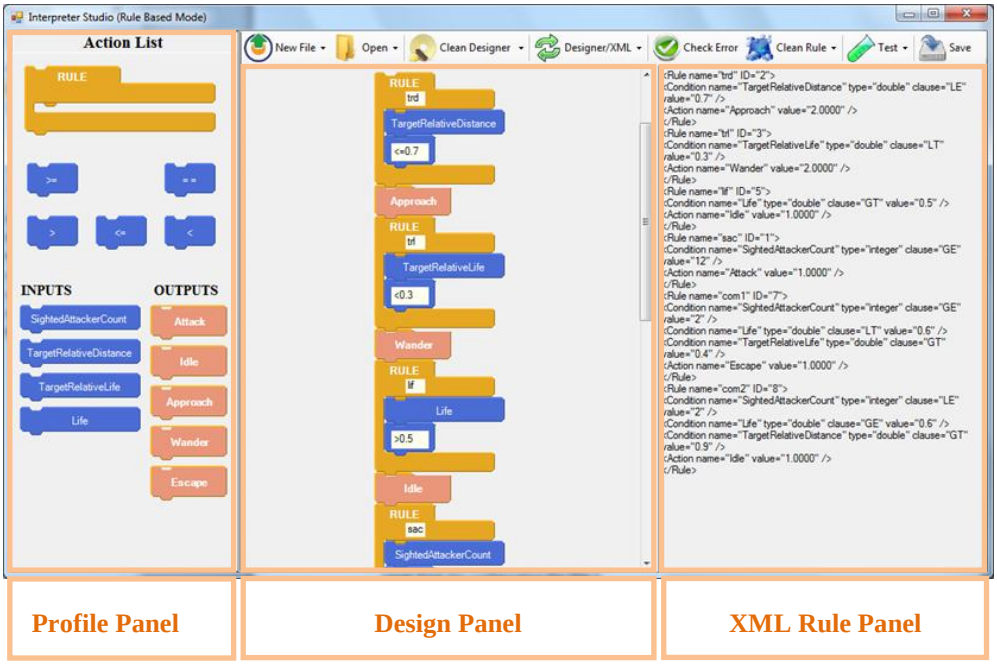


Figure 32: Rule Designer Studio

Profile Panel

The profile panel contains the basic building blocks that are required to design and create the rules. As shown in Figure 32, the profile panel has four different draggable building blocks that are used to design the rules, namely the *Rule block*; the *Operators blocks*; the *Input blocks* and the *Output block* (see Figure 34). The rules definitions for the Input and Output blocks are defined by in an XML file known as the *state profile*. The state profile lists the available inputs types, the valid range for the inputs and the number of valid output actions. A sample *state profile* that is used in Meme War is depicted in Figure 33.

```
<?xml version="1.0" encoding="utf-8"?>
<StateProfile>
  <input index="1" enabled="False" type="integer" name="SightedAttackerCount" />
  <input index="2" enabled="True" type="double" name="TargetRelativeDistance" />
  <input index="3" enabled="True" type="double" name="TargetRelativeLife" />
  <input index="4" enabled="True" type="integer" name="Life" />
  <output index="1" enabled="True" name="Attack" />
  <output index="2" enabled="True" name="Idle" />
  <output index="3" enabled="True" name="Approach" />
  <output index="4" enabled="True" name="Wander" />
  <output index="5" enabled="True" name="Escape" />
</StateProfile>
```

Figure 33: Meme Wars State Profile

Designer Panel

The designer panel is located in the middle panel in the Rule Designer Studio. The designer panel is like the canvas or the drawing board whereby users, on this panel, design the rules by fitting together the appropriate building blocks from the profile panel to create the rules. A complete rule consists of 1) one main *Rule Block*, 2) any number of *Input Blocks*, each with an *Operator Block* attached to defines the conditions and value range

and 3) a number of *Output Block* indicating the actions that should be performed when the conditions are met (see Figure 34).

The rule uses a simple rule format consisting of IF-THEN conditions which denote the actions that the agent should perform.

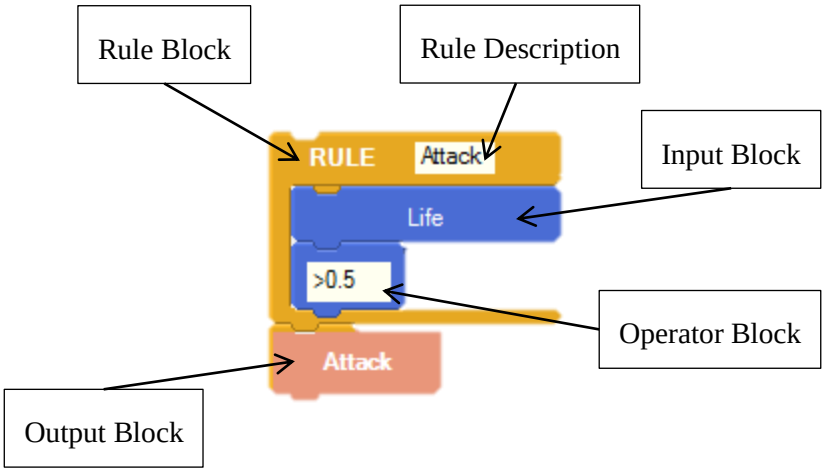


Figure 34: Example of a valid rule in the Rule Designer Studio

XML Rules Panel

The XML panel displays the rules that are designed in the Designer panel as a XML format that could be exported and used in Meme War or any applications that complies with the file structure provided. This rule formation is saved as an XML file to allow easy portability to other rule engine in the future.

4.4.3. Meme War Map Editor

Meme War provides a map editor inside the game to allow a certain degree of customization to the game by allowing players to create new maps within the game environment. Figure 35 shows the heads-up display (HUD) of the map editor. The map

editor provides a comprehensive set of terrain editing tools, as well as various types of 3D models.

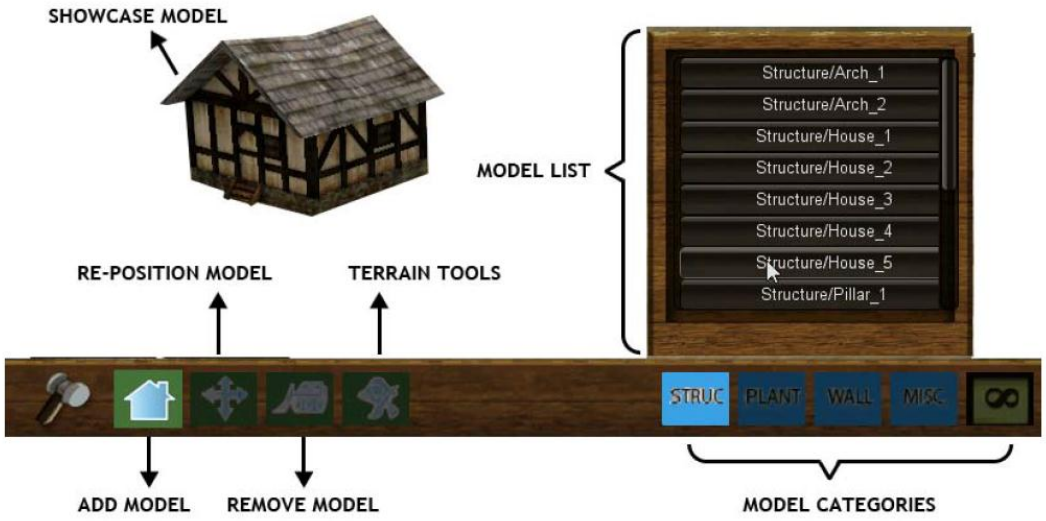


Figure 35: Meme War Map Editor GUI

Terrain Editing Tools

The map editor allows players to define the surface elevation of the terrain by loading in a heightmap. A heightmap is a grey scale image file, whereby the colour information at each pixel represents the height information at each point of the terrain. The colour information within the heightmap (value between 0-255) is converted into a 2D array and is normalised when loaded into the game engine.

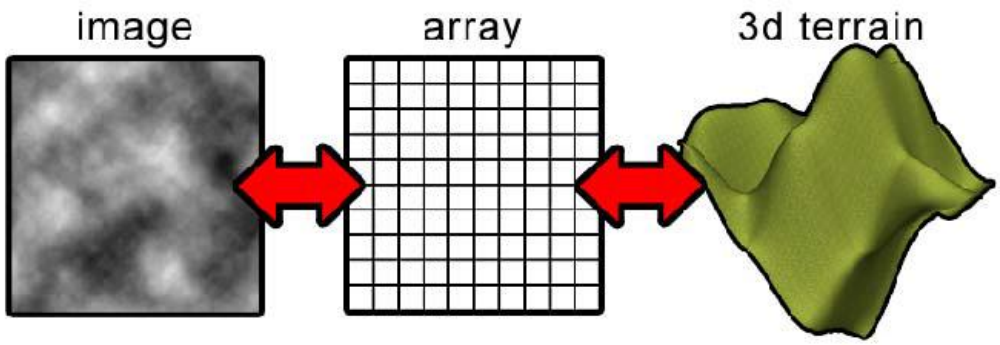


Figure 36: Heightmap to 3D Terrain representation

In addition, the map editor also allows on-the-fly editing of the terrain through a set of tools provided. These tools include allowing user to raise or lower the height of the terrain at the selected region of interest (see Figure 37a) and to flatten a region of the terrain to a user specified height (see Figure 37b).

a) Raise or lower terrain at region of interest

b) Specify height of terrain at region of interest

Figure 37: Terrain surface elevation editing tools

4.5. Play testing

In order to find out how well Meme War performs in achieving its objective on being a good game platform for AI research, Meme War has been play tested with a group of 71 computer science undergraduate students from Nanyang Technological University taking neural network course. The students were told to train an agent of their choice using Multiple Layer Perceptron as the AI method.

4.5.1. Simulation Setup

These students were told to submit one trained agent of their choice and they are graded based on how well their agent performed. Students are free to design the number of layers and neurons in each layer when creating the agent.

The student's agents are placed in a death-match arena for them to compete against each other. A total of 100 simulation rounds of three minutes were performed. This is to reduce the effect of the luck factor that is influencing the performance of the agent in each round. The scores for each round were recorded and averaged to determine the performance of the agent.

Results

Table 4 shows the performance of every agent trained by the students. From the result, it is observed that most students prefer using the archer. This is due to the long range advantage of the archer unit over the other melee units. This suggests that the game probably needs to do some balancing to reduce the advantage that the archer have over the other units. Apart from the game balancing issue, feedbacks received on Meme War are generally positive as the students find the game interesting and they could play and learn something new at the same time.

Table 4: Student's Score

Student ID	Unit Type	Averaged Score
085742K06	Cavalry	835.539
085757A06	Archer	2361.105
085493D06	Archer	2509.52
085762E06	Archer	1654.712
085579c06	Archer	2590.472
085552E06	Archer	2839.94
085482K06	Archer	1586.011
085495H06	Archer	2674.701
U0940324H	Archer	2124.062
N1103097A	Cavalry	711.73
N1103099F	Knight	559.038
085818F06	Archer	2483.368
085662A06	Archer	2961.097
085603G06	Swordsman	732.117
085629C06	Cavalry	1276.614
U0922162L	Cavalry	746.891
085715G06	Cavalry	1028.608
085557D06	Archer	1573.813
085465k06	Archer	1768.786
085522G06	Archer	1597.36
U0921759A	Knight	687.79
085863L06	Archer	2614.574
N1102818D	Knight	763.141
085678E06	Cavalry	1085.627
085616E06	Archer	2126.306
085809E06	Archer	1913.055
085618J06	Archer	2355.378
U0922489C	Cavalry	951.166
085707H06	Cavalry	958.316
085815L06	Archer	2790.065
U0921721J	Archer	3054.011
075152E06	Cavalry	570.781
087401G16	Swordsman	843.663
085825C06	Archer	2763.988
085667L06	Archer	2274.092
085719D06	Archer	1155.29

Student ID	Unit Type	Averaged Score
N1103033C	Archer	2300.99
085505G06	Archer	2787.869
085838A06	Cavalry	849.909
085576H06	Archer	2800.791
085726A06	Archer	2444.97
085690F06	Archer	1650.949
085730C06	Archer	2345.437
085768F06	Archer	2218.688
085668B06	Archer	2643.248
085729G06	Archer	2369.268
085467C06	Cavalry	1019.894
085666J06	Cavalry	855.877
085516A06	Archer	2307.206
085466A06	Archer	2628.301
085544F06	Archer	2743.972
085926A18	Swordsman	593.587
085604J06	Archer	2001.5
U0940275D	Archer	2233.552
085541L06	Archer	2376.75
085671B06	Archer	3037.66
N1102807J	Cavalry	983.865
085743A06	Archer	1496.136
085806K06	Archer	1678.129
085637B06	Archer	2998.768
087385B16	Archer	1573.169
U0922094H	Cavalry	884.509
085775C06	Archer	1897.966
085601C06	Swordsman	576.634
085771F06	Archer	3101.49
085654B06	Archer	1747.874
085553G06	Cavalry	1338.276
085548C06	Archer	2014.435
085857E06	Archer	1669.812
085714E06	Archer	2306.253
085652J06	Knight	827.274

	Archer	Cavalry	Swordsman	Knight
Choice of Unit	48	15	4	4

4.6. Website with more Information and downloadable executable

The website for Meme War can be found in the following link:

`http://www.c2i.ntu.edu.sg:81/memewarweb/memewar.php`

The website includes some project information, instructions, downloadable executable as well as a web player that allows user to play the game on their web browser.

5. FAME in Commercial Game Launched – Dark Dot

FAME library package was used successfully in collaboration with Singapore-MIT Gambit Game Lab in the development of a commercial game, *Dark Dot*. Dark Dot is an action shooter games released on the Apple iPad platform The player controls a group of minions known as Darklets in a quest to defeat the enemy [**31**] (see Figure 38). Here, the movement mechanics of the Darklets is controlled using FAME.

Figure 38: Dark Dot Screenshots

Shortly after it was released in October 2011, *Dark Dot* became the top action game in 48 countries and clinched the number one position in the iTunes apps store, with a download of well over 448,000 players worldwide, with 27% of its players from China

and 17% from the USA [**32**]. Figure 39 shows a screenshot of iTunes App Store application, depicting Dark Dot as the number 1 recommended apps in the Top Free iPad Apps list. *Dark Dot* utilized the advanced mechanics provided in *FAME* to control the movements of the agent in the game. It received notable comments from the press, citing Dark Dot as "*not only fun and creative and lovely to look at, but it's got some stunningly original gameplay mechanics.*"

Figure 39: Dark Dot, #1 Position in iTunes App Store

6. Conclusions

This thesis presents FAME, a comprehensive C# library designed for the easy creation of soft flock formation controls. In this soft formation control, the agents are able to temporary deviate from their formation when encountering anomalies and return to its formation when the path is clear, as well as being able to self-organize within the formation to reduce the overall time required to fill up the formation. Similarly, the formation is able to change or mold according to users preference during runtime and is able to bend naturally along with the curvature while negotiating the path. FAME also provides a seamless way to partition the scene so that the overall computation requirement is reduced. FAME, was developed as well to handle the movement behaviour of the agents. FAME is able to achieve natural and realistic group movement behaviour with a large number of virtual agents.

FAME was deployed and tested in two game applications, Meme War and Dark Dot. Meme War is an agent-based death match-style simulation based game that is co-developed by the author together with a team of research staff in the Centre for Computational Intelligence lab. The game is designed for academic research based on AI methodology. The game was play-tested and used as a course material for the neural network course taught in Nanyang Technological University. It has proven that the FAME works well in simulating the movement behaviour of the agents in the game. Another success story was Dark Dot, an iPad game developed in collaboration with Singapore-MIT GAMBIT game lab using FAME technology. The game was launched in the Apple iTunes

store in October 2011. Shortly after its launch, Dark Dot became the number 1 recommended iPad game in the iTunes App store, with over 448,000 downloads from player worldwide. Dark Dot received generally positive review from various game review website and the local media, of which one notable comment cited Dark Dot as "*not only fun and creative and lovely to look at, but it's got some stunningly original gameplay mechanics.*"

Contributions

Publications

Ho, C. S., Ong, Y. S., Chen, X. S., Tan, A. H.: "FAME, Soft flock formation control for collective behaviour studies and rapid game development", The Ninth International Conference on Simulated Evolution And Learning (SEAL), 2012. (Accepted)

Ho, C. S., Nguyen, Q. H., Ong, Y. S., Chen, X.: Autonomous multi-agents in flexible flock formation. In : Motion in games, Utrecht, The Netherlands, vol. 6459, pp.375-385 (2010)

Games

- Meme War
- Dark Dot

Software Tools

- Rule Designer Studio

Website:

FAME: http://c2inet.sce.ntu.edu.sg/fame/

Meme War: http://www.c2i.ntu.edu.sg:81/memewarweb/memewar.php

Reference

1. Funge, J., Tu, X., Terzopoulos, D.: Cognitive Modeling: Knowledge, Reasoning and Planning for Intelligent Characters. In : SIGGRAPH, pp.29-38 (1999)

2. Durupinar, F.: From audience to mobs: crowd simulation with psychological factors. PhD Thesis, Bilkent University (2010)

3. EL, H.: The flow of human crowds. Annual Review Of Fluid Mechanics, 169-182 (2003)

4. Stephen, C.: Flow tiles. In : Symposium on Computer animation (2004)

5. Reynolds, C.: Flocks, herds and schools: A distributed behavioral model. In : SIGGRAPH Computer graphics and interactive techniques (1987)

6. Reynolds, C.: Steering behaviors for autonomous characters. In : Game Developers Conference, pp.763-782 (1999)

7. Xiaoyuan, T., Demetri, T.: Artificial fishes: physics, locomotion, perception, behavior. In : SIGGRAPH Computer graphics and interactive techniques (1994)

8. Fridman, N., Kaminka, G.: Towards a cognitive model of crowd behavior based on social comparison theory. In : AAAI, pp.731-737 (2007)

9. Pan, X., Han, C., Law, K.: A multi-agent based simulation framework for the study of human and social behavior in egress analysis. In : The International Conference on Computing in Civil Engineering, pp.12-15 (2005)

10. Olfati-Saber, R.: Flocking for multi-agent dynamic systems., 401–420 (2006)

11. Leonard, N., Fiorelli, E.: Virtual leaders, artificial potentials and coordinated control of. In : IEEE Conference on Decision and Control, Orlando, vol. 3, p.2968–2973 (2001)

12. Olfati-Saber, R., Murray, R.: Graph Rigidity and Distributed Formation Stabilization of Multi-Vehicle Systems. In : Decision and Control, vol. 3 (2002)

13. Anderson, M., McDaniel, E., Chenney, S.: Constrained animation of flocks. In : ACM SIGGRAPH/Eurographics Symposium on Computer Animation, pp.286-297 (2003)

14. Lai, Y., Chenney, S., Fan, S.: Group motion graphs. In : ACM SIGGRAPH/Eurographics Symposium on Computer Animation, pp.281-290 (2005)

15. Xu, J., Jin, X., Yu, Y., Shen, T., Zhou, M.: Shape-constrained flock animation. In : Computer Animation and Virtual Worlds, vol. 19, pp.313-330 (2008)

16. Aubel, A., Boulic, R., Thalmann, D.: Real-time display of virtual humans: levels of details and impostors. Transactions on Circuits and Systems for Video Technology, 207-217 (2000)

17. van den Berg, J., Lin, M. C., Manocha, D.: Reciprocal Velocity Obstacles for Real-Time Multi-Agent Navigation. In : IEEE International Conference on Robotics and Automation, pp.1928--1935 (2008)

18. Lin, M. C., Avneesh, S., Jur, B., Russell, G., Sean, C., Hengchin, Y., Stephen, G., Eric, A., Sachin, P., Jason, S., Dinesh, M.: Real-Time Path Planning and Navigation for Multi-agent and Crowd Simulations. In : Motion in Games (2008)

19. Ho, C. S., Nguyen, Q. H., Ong, Y. S., Chen, X.: Autonomous multi-agents in flexible flock formation. In : Motion in games, Utrecht, The Netherlands, vol. 6459, pp.375-385 (2010)

20. Ugo, E., Rosario, D. C., Vittorio, S., Maurizio, T.: Massive Simulation using GPU of a distributed behavioral model of a flock with obstacle avoidance. In : Vision, Modeling and Visualization (2004)

21. Ugo, E., Bernardino, F., Vittorio, S.: BehaveRT: A GPU-Based Library for Autonomous Characters. In : Motion in Games (2010)

22. Alessandro Ribeiro Da, S., Wallace Santos, L., Luiz, C.: Boids that see: Using self-occlusion for simulating large groups on GPUs. Comput. Entertain., 51:1--51:20 (2009)

23. Reynolds, C.: Interaction with Groups of Autonomous Characters. (2000)

24. Erick Baptista, P., Mark, J., Marcelo, Z., Esteban Walter Gonzalez, C., Anselmo, M., Aura, C., Bruno, F.: A bidimensional data structure and spatial optimization for supermassive crowd simulation on GPU. Comput. Entertain., 60:1--60:15 (2010)

25. Shawn, S., Mishali, N., Mubbasir, K., Petros, F., Glenn, R.: Watch Out! A Framework for Evaluating Steering Behaviors. In : Motion in Games (2008)

26. Shawn, S., Mubbasir, K., Petros, F., Glenn, R.: An Open Framework for Developing, Evaluating, and Sharing Steering Algorithms. In : Motion in Games (2009)

27. Michael S., F.: Mean value coordinates. Comput. Aided Geom. Des., 19-27 (2003)

28. Dawkins, R.: The Selfish Gene. Oxford University Press (1976)

29. Xianshun, C., Ong, Y., Lim, M., Kay Chen, T.: A Multi-Facet Survey on Memetic Computation., 591-607 (2011)

30. Nareyek, A.: Intelligent Agents for Computer Games. In : Computers and Games, pp.414-422 (2002)

31. Dark Dot. In: Singapore-MIT Gambit Game Lab. Available at: http://gambit.mit.edu/loadgame/darkdot.php

32. Sunny, H.: Local game developer tops the Chart. TODAY, Singapore (22 DECEMBER 2011)