Realistic Behaviour Model for Bird Autonomous Agent

University of Auckland

COMPSCI 780

Summer Project

February 23, 2004

By Tae-Rim Han

UPI: than026

ID: 3400704

Contents

Section 1

Introduction and Research Overview

1.1 Aim of the project

The aim of this project is to generate an autonomous agent system for birds that behave in a realistic manner. The first stage of the project is to analyse and to build a Boid system presented by Craig Reynold [Reyn87], which allows the Boid agents to flock in a naturalistic way. Although the Boids are subset of 'a-life', which acts and makes decisions autonomously according to its neighbouring information, its intelligence is limited to its current model of intention - flocking. Therefore the second stage of the project is to further modify this model to make agents to generate intentions and behave more realistically according to these intentions. To do this a way to integrate the Boid agent model with the behaviour/brain model by Xiaoyuan Tu and Demetri Terzopoulos is to be analysed and implemented. To allow other behaviours other than flocking behaviour, types such as collision avoidance, pursuit and flee should be adopted from the [Reyn99].

1.2 Definition of A-life and how it is different from Artificial Intelligence

The Artificial Life, also known as "AL" or "Alife" is a study, which involves simulating and understanding living organism by recreating the process and the behaviours of their biological phenomena within computational environment. Alife concentrates in simulating life like behaviours of the organism such as behavioural intentions, adaptation to dynamic environments, socialization and learning. The result of Alife is very subjective as it depends on the observer to decide whether the simulation is naturalistic or artificial.

The term, Artificial Intelligence (AI), is often used interchangeably with the term Artificial Life where this is incorrect as they are two separate studies. AI is a much older field of study, which employs a top-down methodology. This involves in identifying complex behaviours to build systems that satisfies all the specification of the application. Such an application includes chess games. Alife in the other hand employs bottom-up approach where it takes into account the small behaviours, which then can be combined with other simple behaviours to produce an overall pattern of behaviours. It is largely based on biological synthesis where by combining separate elements or units can form a whole coherent action or being. The combinations of these small

artefacts are often not pre-planned where it uses simple rules based on the subject to produce naturalistic result. Due to this, interesting and unexpected properties can emerge at higher levels.

The Boid model by Craig Reynold can be categorised as a subset of artificial life where it simulates the flocking model of birds by combining several behaviour rules. The Boids are not considered as AI model since it does not comply with some of the basic concepts used in a AI system such as memory, planning and coherent learning. Therefore it is more closely related to Artificial Life since it analysis and studies the lifelike steering properties of the natural subject. Since the Boid model is restricted to the studies of the flocking properties only, some improvements can be made by allowing the Boids to be able to generate other intentions other then flocking in their system. A technique to generate intentions for living organism such as fishes is presented in the [Tu1994] paper. The integration of Boid model and the intention generator model will allow the Boids to perceive other environmental factors such as predators and obstacles, along with the its mental state to decide what action that needs to be applied to compliment the external and the internal state of the subject.

1.3 Overview of Boid Model

Craig Reynold first introduced the Boid model in 1987, which demonstrated a distributed behaviour model that can realistically simulate the flocking behaviours of birds. The definition of the term, “Boid”, refers to objects that simulate bird-like behaviours in the context of flocking where other objects such as fishes which schools are also called Boids. Overall flocking behaviour is simply the result of combining interactions between Boids that are in the flocking neighbourhood. These interactions are governed by three distinctive rules where these are Separation, Alignment and Cohesion. The Separation rule (also known as, Collision Avoidance) is used to prevent collisions and crowding of near by flock mates. The Alignment rule (also know as, Velocity Matching) involves in steering the nearby flock mates in a common direction with matching velocity. The Cohesion rule (also know as, Flock Centring) is used to steer a Boid to the average position of nearby flock mates in attempt to stay close to each other. These rules are combined to generate a steering force that can be applied on the Boid model to determine the force and the direction to which the Boid needs to move to comply with the flocking rules.
Flocking algorithm discussed here is not based on some complex mathematical model but it involves computing the local information gather from its perception system to determine its current and future navigations and orientation. The perception system makes use of the local

coordinate system to measure things in respect to the Boid's position and orientation. Boid's geometric flight motion also uses the local coordinate system to determine the flight path for each frame where the path is an incremental translation along the Boid's "forward axis" or "local z-axis". The yaw and pitch movement along the local x and y-axis can be determined by the steering force. The roll movement around z axis is used for banking where the local y axis of the Boid is aligned with the acceleration that is being applied on the Boid model. Therefore the Boid's local space gets aligned with the "acceleration" coordinate system.

Advantage of this Boid system would become noticeable when developing a complex animation of flocks where each movement of the actor in the scene can be automated rather then edited manually for each frame.

1.4 Overview of the Tu's Mind/Brain model

In the paper presented by Xiaoyuan Tu [Tu1994], it describes in detail on how to generate autonomous fish agents. Much like the [Reyn99], this paper also discusses the motion of autonomous agents in several layers. These layers are action selection, steering and locomotion in [Reyn99] and similarly in [Tu1994], perception, behaviour subsystem and motor. In this project the main focus will be on the perception and behaviour layer to implement the brain for the Boid model. To better understand the purpose of each level see Figure 1.1.

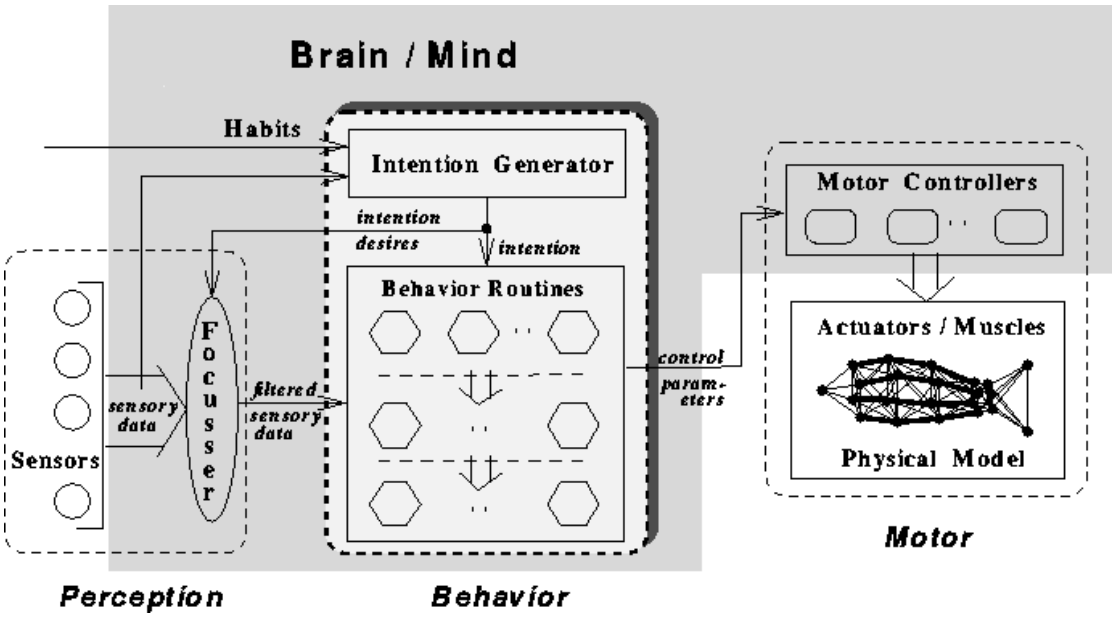


Figure 1.1: Control and information flow in artificial fish

Perception system consists of the sensory data and the focuser. Sensory data is a collection of information gathered from sensors, such as temperature and vision sensors. Sensors are adjusted according to the demand for particular information where this is coordinated by the focuser, which acts as a perceptual attention mechanism. The role of the focuser is to filter out the sensory information so that only required information gets passed to the behaviour system to select the action for the motor layer. The intention generator that bridges the perception and the behaviour system is responsible for telling the focuser on what particular information to focus its attention on. Intention generator is a subset of the behaviour system that does the brainwork for the agents. It uses the data gathered from the sensors along with the agent's habit parameters and the mental state variables to establish intentions for the agents. Further detail on these parameters and implementation details will be discussed in the later sections.

Section 2

Implementing the Behavioural Model of Bird autonomous agent

This section will concentrate on the implementation details of the bird autonomous agents by adopting the concepts presented by Reynold and Tu as discussed already in the previous section. This section will look into detail on how the two concepts can be integrated together to generate a single working model. The contents, which will follow, are organised in a logical manner, where it shall first discuss the natural properties of birds in the real environment in order to gather data that could make the simulation more realistic. After having discussed the properties of Birds these information are then used to design the programme structure for the perception system, intention-generator and the behavioural model.

2.1 Properties and behaviours of birds

Before moving on to the technical details on how the birds are to be modelled in the virtual environment, these are some of the important properties of birds that need to be taken into consideration when building the agent model.

Vision is one of the most important features of birds where they depend on their vision to survive, to eat, to flock and to live, therefore their sight is very well developed. The sharpness and distance range at which they can perceive is much superior then those of humans, where bird such as an eagle can see up to 8 times more clearly than a human, with their distance range as far as a mile. The predatory birds uses its visuals to determine how far the prey is located, its size, position and motion. Since the bird eyes are located on the side of their heads and also their eyes are large in relative to their skull, some birds are able to see 300 degrees without turning their head.

Another important property of a bird is their ability to flock. There are several reasons why the birds flock together where the most dominant reasons would be to get protection from predators, to mate and to quest for food. In some situations flocking depends heavily on their natural habitat where birds flock together to migrate from one location to the other as seasons change. Flocking ability only resides in birds that are weak where birds such an owl or an eagle doesn't flock together.

2.2 Program Structure

Much like the [Reyn99, Tu1994], the motion behaviour for the agent model has been modulated into different layers where these areas are, perception, intention generator, focuser, steering behaviour and locomotion. The perception, intention-generator and focuser system has been adopted from the Tu's original model where attempt has been made to modify the model so it better suits the properties of birds rather then fishes. The steering behaviour and the locomotion layers are based on the Reynold's 87 and 99 model where [Reyn99] is used to implement steering behaviours other than flocking to support actions for intentions such to avoid and to escape. In the following sections the purpose and the properties of these layers will be analysed and discussed in further detail.

2.3 Perception and intention-generator

2.3.1 Perception System

One of the important aspects of perception is to analyse what the agent model can see from its current position and orientation. As mentioned above birds are able to perceive things that are very far away and within their viewing angle, therefore the simulated birds should be able to imitate these properties.

In the current implementation and also for the [Reyn87, Tu1994], in order to gather information on what the agent can see, it draws out a sphere from the agent's position with the vision distance used as the sphere's radius. It then does a simple computation to see what lies within this spherical region that satisfies the viewing angle of the agent. In the current model, the vision distance is also affected by the size of the target where the default vision distance is scaled by the factor of the target's size. This concept is true in the situations where small prey can detect the larger predator earlier in the scanning process then the smaller predator located same distance away.

The calculations for perception are done using vector math where offset vector is calculated from the current agent's position to the target object's position where the magnitude of this vector determines the distance the target is away from the current agent. Then the dot product of normalised offset vector with the agent's forward axis is computed to obtain the angle between the target and the agent. After these values are computed, comparisons are made to check whether the target can be seen from the agent's position. Figure 2.1 illustrates the above concept.

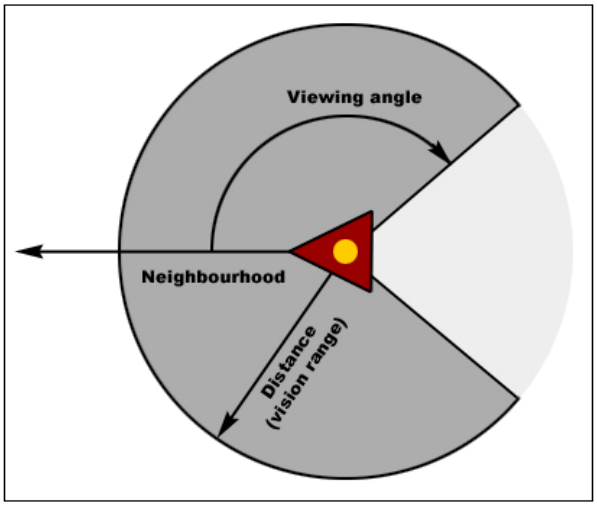


Figure 2.1: Perception system

On every frame update, each agent calculates and stores in a pointer-linked list, of the agents that are within this spherical region. This is to allow the agent to only work with the information that is perceptually available to them and also to quicken the search quarries so that if perception information is needed again in the same frame, it does not need to search through N number of agents in the environment but only C number of agents in the perceived neighbourhood.

2.3.2 Intention Generator

The intention generator gathers the input from the perception system (referred as sensory system in [Tu1994]), the habit parameters and the mental state of the agent, in order to generate an appropriate intention for the given frame. These input variables will be discussed in detail in the following subsections.

Habits

The habit parameters are stored as static binary information where these are defined at the initialisation of the agents. These are different from the desires or the intentions, as its values remain constant through out the simulation process. In the agent model there are three habit parameters where these are flocking, flying and walking. These parameters were chosen to support birds that like or dislikes to flock and whether if the bird are flightless birds or tend to be on continuous flight. If both flying and walking is chosen for the agent then these two values will work complementary to each other by allowing the agents to rest when mental state for tiredness

reaches its maximum threshold and fly again when tiredness is satisfied. Different habits could have been chosen such as darkness, brightness, cold or warmth like the fish habits in Tu's model but for the given environment these three habits were the only ones that seemed appropriate.

Mental State Variables

There are three mental state variables implemented for this agent model where these are hunger (H), fear (F) and tiredness (T). The tiredness (T) has replaced the libido (L) variable that was used in the Tu's model. The computation of the H and F was done according to the formulas that were used by Tu. These variables are computed for every frame update where the values range from 0 to 1. The higher the value, the more the agent wants to eat, avoid predator and rest. Here are formulas used to compute these variables:

$$H(t) = \min\left[1 - n^e(t) R_H(\Delta t^H)/\alpha, 1\right]$$

$$F(t) = \min\left[\sum_i F^i, 1\right], \text{ where } F^i = \min\left[D_0 / d^i(t), 1\right]$$

$$T(t) = \min\left[1 - n^r(t) R_T(\Delta t^T)/\beta, 1\right]$$

The following is a list of all the arguments and their definitions for the above formulas:

Arguments	Definitions
t	Time elapsed for each time frame
$n^e(t)$	Total Amount of food consumed by the agent
$R_H(x) = 1 - p_0 x$	Function which calculates the ratio at which food can be digested in respect to the digestion rate and the time since the last meal
p_0	Digestion Rate
Δt^H	Time since the last meal
α	Constant that indicates the appetite of the bird, where larger birds should have bigger appetites
F^i	Fear towards the predator i.
D_0	Constant that indicate the coward ness of the bird where higher the value the more fear the bird has of the predators.

d^i	Distance to the predator i.
$n^r(t)$	Total Amount of rest consumed by the agent
β	Constant that indicates the maximum rest value of the bird, where birds that take longer rest should have larger β
$R_T(x) = 1 - p_1 x$	Function which calculates the ratio at which tiredness can be satisfied in respect to the tiredness rate and the time since the last rest
p_1	Tiredness Rate
Δt^T	Time since the last rest

There were recommended values given for some of the constants in the Tu's paper such as the p_0 =0.005 for hungry fish and D_0 =500 for coward fish but these values didn't work well when applied to the current implementation.

[Example of mental state variable]

Intention generator

After having gathered all the necessary information, appropriate intention is generated (selected) for the given input. The intention generator process depends on both the external and internal urges of the agent. There are set number of intentions that the generator can select from where these are, avoid (collisions), eat, escape, flock, wonder and rest. Intention generator differs depending on the type of the agent where there are ORDINARY, PREDATOR, LEADER and FOOD agents. It is assumed that ORIDINARY and LEADER type shares the same intention generator where as PREDATOR has its own separate generator. This is because the PREDATOR type does not support the same intention as the ORDINARY and LEADER agent type. Although the FOOD is defined as an agent, which can be further categorised as static food or mobile food depending on the state of the food item, in the current implementation it is assumed that FOOD is static therefore no intention generator is required for this agent type. Further details on each agent type will be discussed below along with its intention generator.

ORDINARY and LEADER

The intentions for these two types of agents are almost of the same except from their flocking properties where if the LEADER agent does not have the flocking habit selected, the two types would be identical. LEADER is a specialized version of ORDINARY where it is naturally designated as a leader to which the other flocking agents can follow. Figure 2.2 is a flow diagram of intention generator for the ORIDINARY and LEADER type.

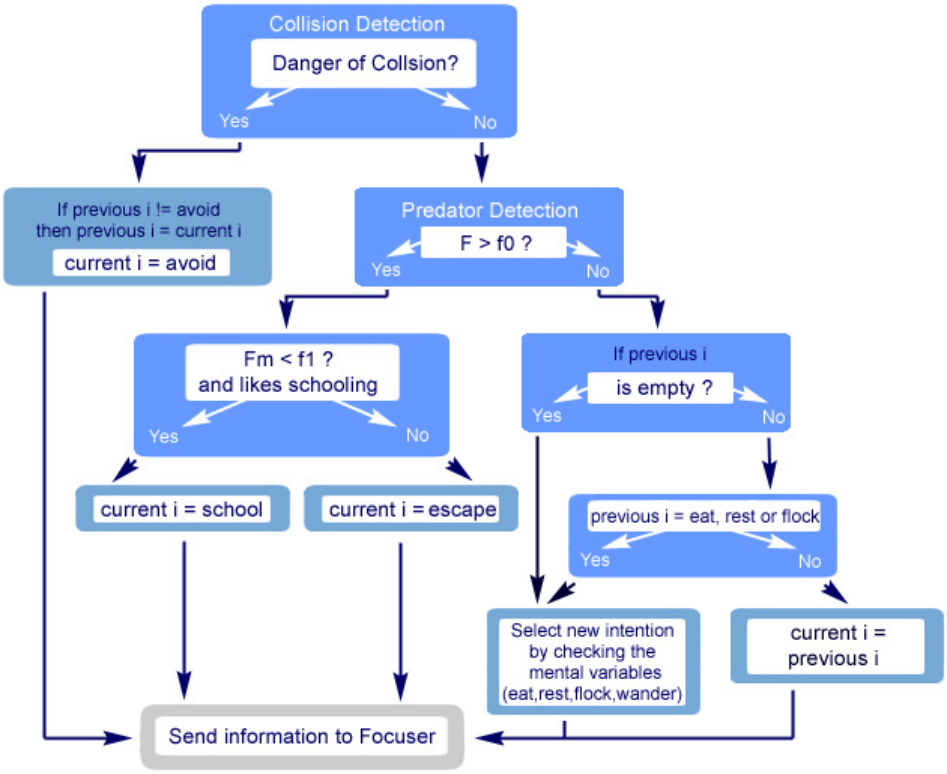


Figure 2.2: Intention generator for ORDINARY and LEADER agents

Above intention generator supports these intentions; avoid (collisions), eat, escape, flock, wonder and rest. These intention-checks are organised in a sorted order with their highest priority being the collision detection. This priority hierarchy model allows most immediate intention to be chosen first before carrying out intentions that are more task specific. It is important to check for intentions this way to avoid unrealistic behaviours carried out by agents in reactions to its dynamic world, such as a hungry bird flying through a wall to retrieve food that is on the other side of the wall.

In this model of intention generator, it first performs the collision avoidance test. In order to detect for potential collisions, the current agent's position must be checked against the positions

of the nearby static or mobile obstacles. Static obstacles are objects in the scene that does not move such as walls and ground where mobile obstacles are objects that are moving, such as neighbouring agents. Collision probing range depends on the type and size of the obstacle and the agent. Depending on the probing distance, this can portray birds that are adventurous or timid. It is important to note that, the main purpose of intention generator at this stage is to decide whether collision avoidance is required and not how the collision avoidance is to be carried out. Once the result of the collision avoidance test has been retrieved, if danger of collision is true, then avoid intention will be assigned as the current intention where if false, other intention checks will be made. In the case of setting current intention to avoid intention, the intention generator first checks to see if the current intention of the previous frame was something other than avoid. If so this intention will be stored in the memory as a previous intention where this can be later retrieved when immediate danger of collision is fulfilled.

If there is no danger of collision, then predator detection is made where this searches for all the predators in the neighbourhood. It checks for the fear state variable F^i of the agent towards the predator i and calculates F^m where m is the most fearsome predator within current neighbourhood. This comparison is done by checking the condition, $F^m > F^i$ for n number of i (see Section 2.3.2: Mental State Variables for calculating the mental state variable F). Along with the mental state variable there are also F threshold variables, f0 and f1, which determines the maximum and minimum tolerance towards the predators. Value range for the threshold variables is the same as the mental state variable, which is [0 to 1]. For f0, its values are normally within the range [0, 0.5] and f1 is in the range [0.5, 1]. Therefore if the total F > f0 then the agent's current intention will be set to escape where this allows the agent to evade from the current location away from the predator. In the case where the most fearsome predator m is not too frightening and the agent has the habit of flocking then its current intention will be set to flocking. $F^m < f1$ where f1 > f0, is the condition that needs to be satisfied for the agent to join a flock in the predator detection stage. If the m is not too frightening but has no flocking habits then this agent will choose to evade where the current intention will be set to escape. These decisions are made to suit the properties of birds where the birds tend to flock when they need protection from the predator. If the birds are not flocking type or flocking type but no flocking neighbours available in the neighbourhood then they have no choice but to escape.

If there is no predator within the current neighbourhood and no collisions to avoid, then the agent will access the previous intention stored away in the memory to decide on the current intention. If

the memory is empty and previous intention is not available then this agent will choose an intention which best suits the current mental state and the habit of the agent. The intentions that are left at this level of the hierarchy are: eat, rest, flock and wander. These are also organised in a priority order, assuming that the birds must eat to survive before it can rest and flock. The agent will only eat and search for food if the value of the H variable reaches 1. Eat intention will be carried out until the hunger is satisfied where H becomes 0 (unless there are immediate danger of collision or predators in the neighbourhood). It is important that the eat intention gets activated only when H is 1 otherwise the agent will take every opportunity to search for food even before the last meal has been digested. Much like the eat intention; the rest intention will be only carried out if the value of T variable reaches 1 where full rest is obtained when the value of T becomes 0. If the agent does not need to eat nor rest then the intention generator checks for the habit variables such as flocking to allow the bird to join a flock, otherwise if all above conditions fail then its current intention will be set to wander until certain desire triggers change of intentions.

PREDATOR

The intention generator for the PREDATOR type is almost same with the intention generator for the ORIDINARY type but with fewer intentions to select from since predator birds don't support the same intentions as the ordinary birds. Figure 2.3 is a flow diagram of the PREDATOR intention generator.

Intention selections for PREDATOR agents are avoid, eat and wonder. Escape intention is not supported in this model since predator birds are most likely to hunt the weaker prey where the cost of hunting for food needs to be as minimal as possible. Therefore it is not necessary to calculate the fear (F) mental variable for predator agents. Also the flocking intention is not supported since there are no reasons for the predators to fly in flocks. If you observe predator bird such as eagles, they tend to survive alone in the wildness.

The way the intention is selected is same as ORIDINARY intention generator where the only difference is that it skips the predator check and goes straight to accessing the previous intention to set the current intention. If the previous intention does not exist then it checks the H variable to decide on whether to eat or wander.

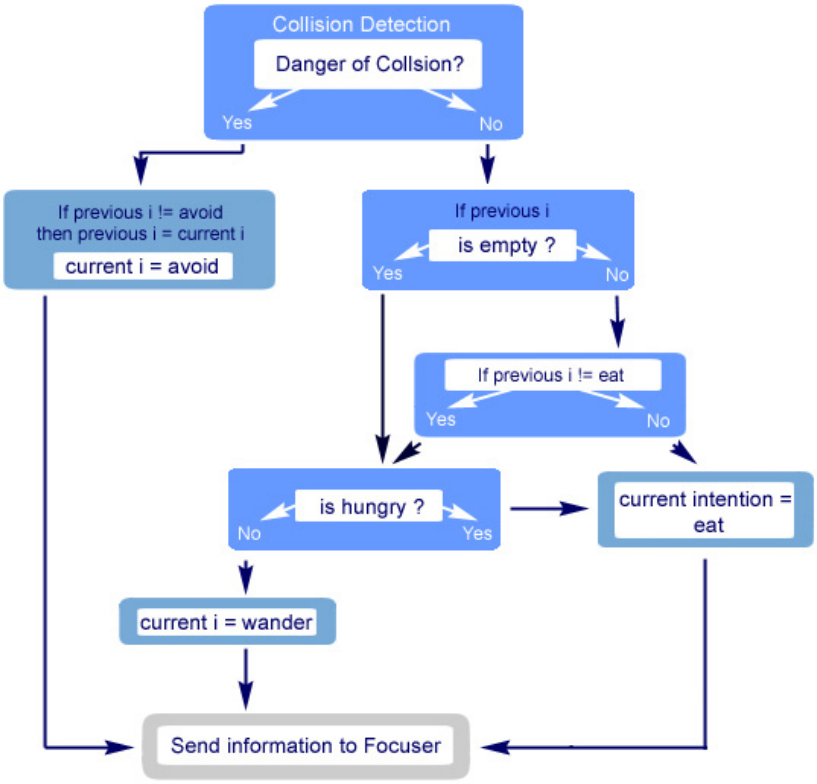


Figure 2.3: Intention generator for PREDATOR agents

For both intention generators the target object for given intention must be stored in memory to avoid the jittering movement of agents when other opportunity arise in the process of pursuing the current target object. This is essential for intentions such as eat and flock, since the intention selection process for these two intentions are affected largely on the current neighbourhood information retrieved from the perception system. Therefore abrupt change in the current neighbourhood may affect the agent's choice of target object for the next frame. To avoid this situation it is critical to store in the memory, the target agent for the current intention, which then can be used to allow persistence in the behaviour of the agent across frames. This mechanism is also effective when the action of agent is interrupted by potential collision or predator detection where sudden change of intention is required. When the immediate danger is out of the way, the agent can probe for the target object and see if the cost of reaching the target object is below the threshold. If false then abandon the original target object and search for a new target object, otherwise continue pursuing the original target object. This same concept is used for keeping the agent inside wall boundaries, where this will be discussed in detail in the wall containment in Section 2.4.

2.4 Focuser and Steering behaviour of agent

After the intention of the agent has been chosen, the focuser is activated. Input to the focuser includes the data retrieved from the perception system and the current intention chosen from the intention generator. The main purpose of the focuser is to filter out any unnecessary information that is not required to fulfil the intention. With the filtered information, the focuser will narrow down the data to a single goal to which the agent can use to process the locomotion of the agent. This single goal can be a seeking point, target agent or direction in which the agent should move. In the current implementation focuser acts as an intermediary between the mind (intention generator) and its physical (locomotion) layer. Where calculating the steering force and the decision making for the steering behaviour for the agent is done in the focuser to have less parameters being passed around different layers of the agent system.

2.4.1 Steering behaviour for the Avoid intention

If the current intention is to avoid, the focuser will search the neighbourhood for the most threatening target obstacle and decides on the steering behaviour to avoid this obstacle. In the implementation, collision avoidance test is carried out in three separate functions where each function deals with obstacles of different types; these are walls, OBSTACLE objects and agent neighbours respectively. From the three different obstacles whichever is present and is the nearest to the agent in the current frame will be chosen as the target obstacle.

Wall containment

The walls in this virtual environment represent the surfaces that act as a container to keep the agents within a restricted space. Therefore if the agent detects the wall, it must then calculate a steering force that will direct itself away from the wall. The [Reyn99] wall containment method was used to keep the agents in a certain region. In the current implementation there are two types of environmental containers where one is a rectangular environment made up of planes and the other is a spherical environment.

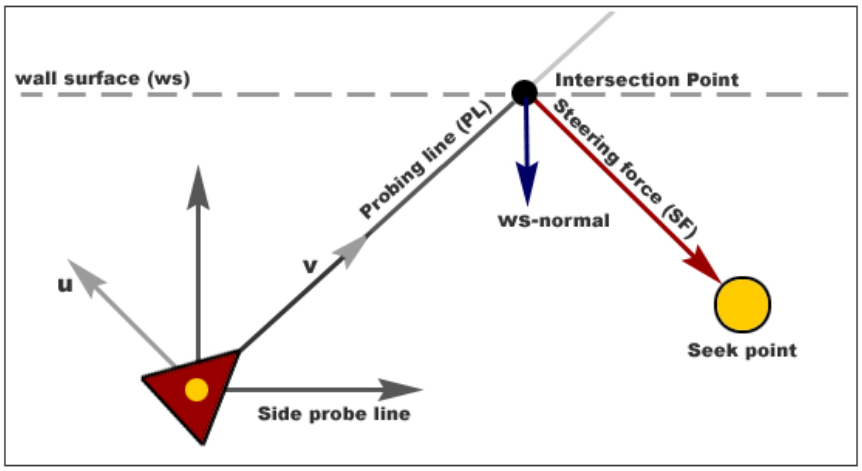


Figure 2.4: Wall containment steering behaviour

To detect the target wall, the agent uses a line probe to find the interception point of the line with the wall surface. The length of the line determines the distance at which the agent can detect the wall where longer the line probe the earlier the agent will react to the obstacle. When the line intersects the wall surface the point of intersection is calculated and the normal of the surface at that point is determined. Steering force is calculated by taking the perpendicular component of this normal to the vehicle's forward direction.

For the rectangular environment the point of intersection is calculated by simple ray / plane intersection test. Since the plane equation is $p \cdot n - d = 0$ and ray equation is $p(t) = s + tc$, the intersection will occur when $(s + tc) \cdot n - d = 0$, where *s* represents the position of the agent and *c* represents the forward direction of the agent (forward axis). *t* then can be calculated by the equation, $t = \frac{d - s \cdot n}{c \cdot n}, c \cdot n \neq 0$. Intersection point *p(t)* is calculated for every wall-plane if *t > 0*.

Point of intersection is then verified to check that it lies within the correct xyz range of the environment. These calculations will determine the wall that is being intersected by the line probe. Determining the normal for the wall surface is simple since the walls are axis aligned. The normals are pointing inwards towards the origin of the space, which is at (0, 0, 0).

For the spherical environment, the point of intersection is calculated by simple ray / sphere intersection test. Intersection of the ray and the sphere occurs when $(s + tc) \cdot (s + tc) - r^2 = 0$ where value of t can be calculated by

$$[t1,t2] = -b \pm \frac{\sqrt{b^2 - 4ac}}{2a} \text{ where } \begin{aligned} v &= p - origin \\ a &= c \cdot c \\ b &= 2c \cdot v \\ c &= v \cdot v - r^2 \end{aligned} .$$

t would be assigned to the smallest value between the t1 and t2 that holds true when (t1, t2 > 0). Since the normal to the sphere is pointing inward, the normal at the intersection point can be calculated by $n = (origin - p).normalise()$. The component of the normal, which is perpendicular to the forward direction of the agent, is computed in the following way, $steeringF = n - (n \cdot c)c$. In [Reyn99], this steering force was applied directly to adjust the steering direction of the agent. It was realised after applying the steering force this way that the method was not very effective in the spherical environment that the steering force was not enough to move the agent away from the wall. This is because when the steering behaviour is being controlled by the intention of the agent, as soon as the agent detects a wall to avoid, it applies the steering force only to avoid the wall for the current frame while the intention is still avoid. This could be problematic in the curved wall situation where as soon as the intention changes from avoid to flock for example, since the line probe is still very close to the wall, it will penetrate the wall again as soon as it changes intention. This will leave the agent in a jerky motion along the wall as long as the agent continuously move towards the wall after the avoid intention. In the current implementation introducing a new type of intention called the immediate intention solved this problem. The purpose of this intention is to keep persistence in the movement of the agent until a certain goal has been achieved. In the Figure 2.4, the seek point is set as the goal point for the immediate intention. Goal point is assigned along the path of the steering force from the intersection point where seek steering behaviour (see about seek in Section 2.4.2) is applied to the agent to move towards the goal. When a certain distance from the goal is achieved the immediate intention is satisfied therefore the agent can continue on with some other intention stored in the previous intention before avoid intention was applied. Another advantage of using the seek point is that it allows the agent to move smoothly away from the wall rather than a sudden jerk away from the wall which would look unrealistic.

If the agent only uses the single probe line along the forward axis of the agent then there are situations where the agent can penetrate the wall even when the forward line probe didn't intersect the wall. This situation occurs when the forward line probe is aligned to the surface of the wall (possible only in rectangular container environment). Therefore two sideline probes were used to detect the walls even when the agent is moving in alignment with the wall. These line

probes are placed slightly ahead and shorter to the either side of the forward probe, see Figure 2.4. Advantage of having these extra line probes is that the agent can perform extra checks to secure that it stays within the containment. Disadvantage would be that this adds on extra computations, which will slow down the simulation but since efficiency in the performance was not the goal of this project therefore three probe lines were used instead of one.

Obstacle object avoidance

In the current implementation obstacle objects are objects that can be placed inside a bounding sphere. Using the bounding spheres to detect potential collision is sufficient since the goal of the avoidance test does not involve performing a through inspection of intersection between polygons. The method used to avoid the obstacles in [Reyn99] was to have a cylinder that lies along the forward axis of the agent where the radius of the cylinder equals the radius of the agent's bounding sphere. This cylinder is extended few distance away from the agent's position, much like the forward probing line for the wall containment method, to detect any future collisions that may occur if the agent stays on the current path. This distance is always assigned with a value that is a factor of current agent's size radius. Intersection test between the cylinder and the obstacle is done by projecting the centre of the obstacle onto the local x-y plane of the agent to see if the distance between the projected centre position and the local origin of the agent is greater or less then the sum of the bounding sphere radius of obstacle and the agent (see Figure 2.5). If the distance is greater than the sum then there would be no collision with the obstacle, but if the distance is less then the sum then agent will collide with the obstacle if it continues to travel in the current direction.

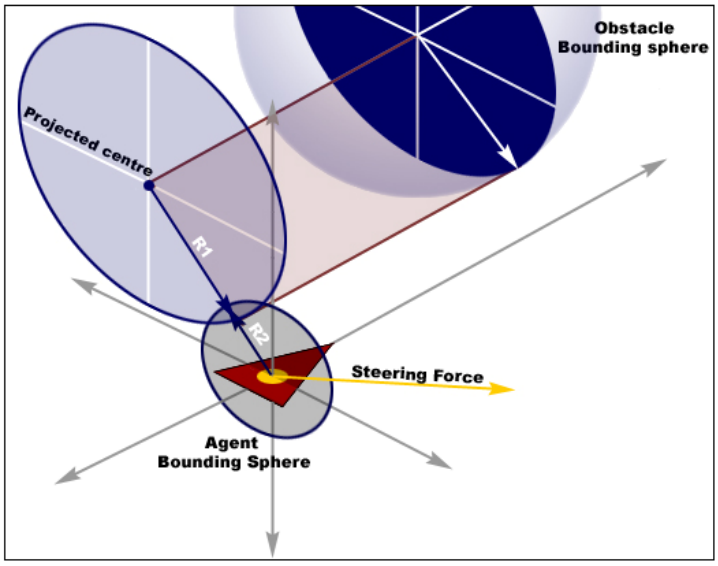


Figure 2.5: Projection of obstacle's centre position to the xy plane of agent

The way Reynold has implemented the obstacle avoidance behaviour is to use the above method as rejection function so that any obstacles that are not on the agent's path can be eliminated quickly before performing a line-sphere intersection test to select the "most-threatening" obstacle. The "most-threatening" obstacle, to his definition is the one that intersects the forward axis of the agent. Implementing the obstacle avoidance this way can result in an unwanted effect because the agent will ignore the presence of the other obstacle in front of the "most-threatening" obstacle if the forward axis does not intersect the obstacle. This situation is illustrated Figure 2.6, where the agent will ignore the Obstacle A since the Obstacle B is chosen as the most threatening obstacle.

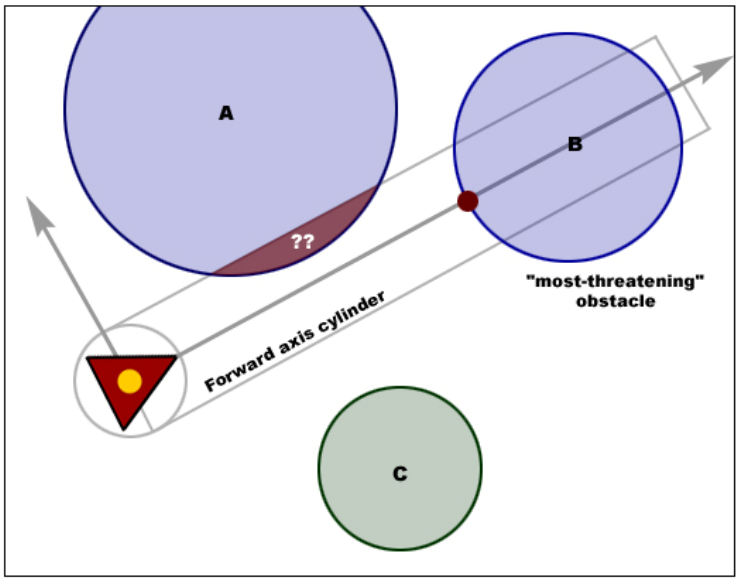


Figure 2.6: Problem with Reynold's obstacle avoidance test

To fix this problem, instead of checking for the line-sphere intersection, just use the rejection function to determine the closest obstacle to the agent that intersects the agent's forward cylinder. Once the nearest target obstacle is chosen the steering force is determined. The steering force is the negated xy projection of the obstacle's centre in the agent's local space (see Figure 2.5, yellow arrow represents the negated centre vector). This is because the agent would need to move away in the opposite direction to which the centre of the obstacle is located.

This obstacle avoidance method is not the most affective way since there is at times that the agent does not steer away from the obstacle even when the obstacle is detected. This occurs when the projected obstacle's centre is near the local origin of the agent space. When this happens the agent's forward direction cancels out the steering force where the agent will continuously move

through the obstacle. This has not been fixed in the current implementation but there are suggestions made by [Reyn87] on how to avoid this situation by using the steer-to-avoid function.

[Avoiding agent neighbours]

2.4.2 Steering behaviour for the Escape intention

When escape intention is set as a current intention for the ORDINARY or LEADER typed agent, the agent's perception system updates the predator list, which is a pointer-linked list of all the predators in the neighbourhood. In the current implementation, focuser uses the predator list to find the most dangerous predator from the current agent's location and determines the steering force to evade from that predator. This may result in a wrong solution if there are other predators in the neighbourhood, which is less dangerous, but nearer to the current agent. Therefore a better solution would be to calculate the offset vector from the agent's position to each of the predators position which then can be normalised and weighted according to the F^i variable (i number of predators). Where the sum of these vectors negated should determine the steering force for the agent.

To discuss the current implementation, once the target predator is selected, the evasion steering-function is called to calculate the steering force. Firstly in the evasion function, the future position of the predator is calculated. Time elapsed between the future and current position of the predator is roughly estimated to be the same as the time elapsed between the previous and current position of the predator, where this value is already known. Therefore future position = current position + current forward direction * time. Having predicted the future position, rest of the calculation is based on [Reyn99] flee (evade) steering function which this is also based on the seek (pursuit) function. To better understand the situation, refer to Figure 2.7.

To quote Reynold on the definition of the desired velocity; "The desired velocity is a vector in the direction from the character to the target. The length of desired velocity could be max speed, or it could be the character's current speed…The steering vector is the difference between this desired velocity and the character's current velocity - [Reyn99]".

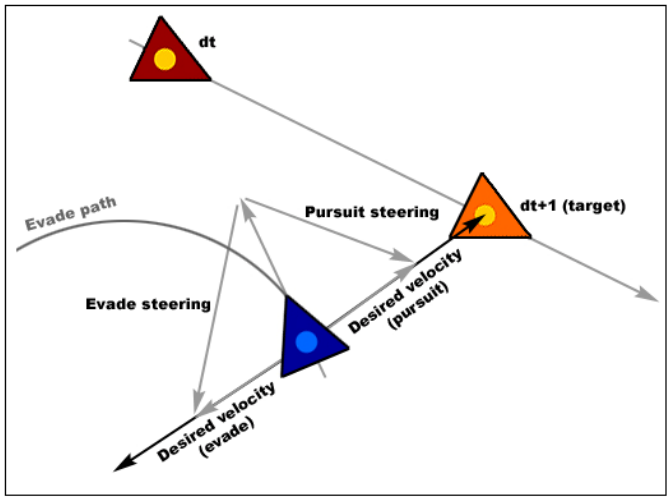


Figure 2.8: Evade and pursuit steering behaviour

Since the agent wants to evade from the target in the case of evasion function, the desired velocity should direct away from the target by negating the original definition of desired velocity. And also in this case the target is a moving target therefore future position of the target must be used to calculate the desired velocity. Following is the code used for the evasion function.

```
CVec3df predictPos = target->getPosition() + target->get_forwardVec()*dt;
CVec3df desiredVelocity = (predictPos - position)*-1;
desiredVelocity = desiredVelocity.normalise() * maxSpeed;
return desiredVelocity - (get_forwardVec() * speed);
```

The seek and flee steering behaviours are almost the same as the pursuit and evade, where the only difference is that the target is static. Therefore instead of using the predicted position of the target use the current position of the target to generate the steering force.

2.4.3 Steering behaviour for the Flocking intention

Original flocking model

When the current intention of an agent is assigned to a flocking intention, the perception system updates the pointer-linked list of flocking neighbours; process of choosing and updating the flocking neighbours will be discussed later. Flocking behaviour is compromised with three separate steering behaviours, which these are separation, cohesion and alignment. In this project, combining process for these steering behaviours are adopted from the concept discussed in

[Reyn99] where each of the steering components gets summed together with weighting factor on each of the steering vectors. In [Reyn87], the combining process is done slightly differently using the prioritised acceleration allocation where the acceleration request (equivalent to steering force) is managed by an accumulator in a prioritised order. A separate accumulator in same prioritised order also manages the magnitude of the acceleration request. When the accumulated magnitude becomes larger than the maximum acceleration value, then the last acceleration request gets trimmed back.

Basically flocking is a combination of behaviour that is generated by each agent in reaction to its local neighbourhood where this is defined by the spherical region around the agent. The radius for this neighbourhood is chosen from the radiuses defined for the three steering behaviours where the largest one is chosen for the local neighbourhood. Flocking behaviour is governed by nine parameters where these are weights, radiuses, and viewing angles (one for each steering behaviour). The weight is used for combining the steering forces together, where the radius decides on the amount of influence the nearby region is to participate in the resulting steering and the viewing angles is the amount of nearby region the agent can see.

Separation

The separation behaviour is used to keep a certain distance away from the nearby agent where this can help to prevent collisions between the agents within the flock. Crowding effects can also be resolved using separation function where this becomes useful in leader following behaviour. To calculate the steering force, all the neighbours that are within the separation radius and the angle are gathered, where the repelling force is created for each of the neighbours by subtracting the neighbouring agent's position from the agent's position. This vector is then normalised and scaled to the length 1/r, where r represents the distance between the two agents. Depending on the distance between the two agents the repelling force will vary. Therefore if the distance between the agents is small then larger repelling force will be generated. When all the forces are calculated they are added together and then averaged by the number of agents in the neighbourhood.

Cohesion

The cohesion behaviour is responsible for keeping the local neighbouring agents together where it allows the agent to steer to the average position of the neighbouring agents. Iterating over the agents within the cohesion radius and angle, and summing their position vectors together and then dividing this value by the number of agents within the neighbourhood can result with the average

position of the neighbourhood. The steering force can be applied in the direction of the average position, where this is computed by subtracting the agent's position from the average position.

Alignment

The alignment behaviour allows the agent to align itself with the other agents in the local neighbourhood. By applying cohesion and alignment behaviour, velocity matching can be achieved between the local neighbours. To compute the alignment steering, the forward axis of the agents in the local neighbourhood is summed up and divided by the number of agents in the neighbourhood. This average alignment vector is the desired velocity therefore the steering vector can be computed by taking the difference between the current agent's forward axis and average alignment vector.

Figure 2.9 illustrates the above three steering behaviours.

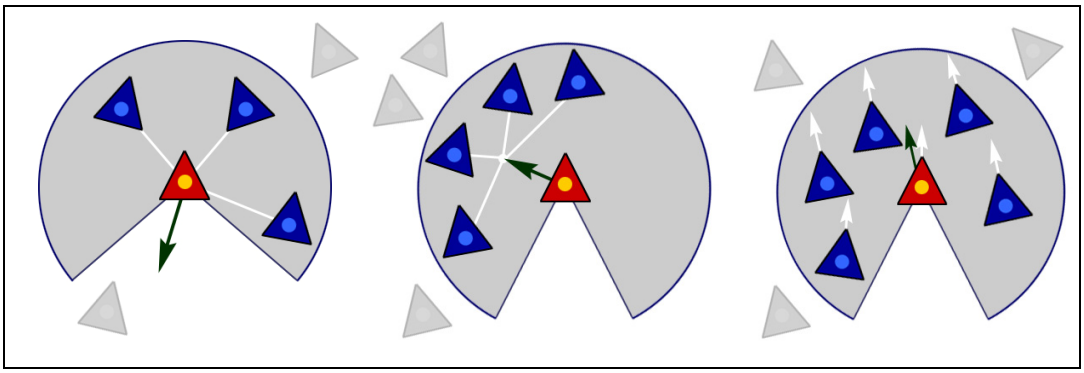


Figure 2.9: 1) Separation 2) Cohesion 3) Alignment

After computing all three steering forces these are combined to generate an overall flocking behaviour for the agent. This is done by, normalising and then multiplying each steering forces by the weight parameter assigned to each of the above steering functions. The weighted steering forces are then summed to produce the overall flocking steering force.

In the current simulation, to get the best flocking result, the radius of the cohesion and the alignment function should be increase so that more than one neighbouring agent is detected within the local neighbourhood. This is due to the fact that the neighbouring agents have great influence on the steering behaviour of the agent therefore if the radius for these two functions is too small then the agents will not be able to form a flock. The viewing angles for these two functions should also be adjusted to achieve best result.

Leader following flocking model

In [Reyn99], he discusses two types of leader following behaviour. The first method involves in having a designated leader that the other agents with flocking intention can follow. The second method involves a single file procession where each agent designates its own leader and follows it. In the current implementation these two methods are combined into one flocking simulation.

As discussed above there is an ORDINARY and a LEADER agent type. The first method utilises the LEADER agent by allowing ORDINARY agents with flocking intention to follow the leader. This method only works when there is at least one agent in the scene with the LEADER type and only when the LEADER agent is within the current agent's viewing range and angle. Otherwise the ORDINARY agent will toggle to the second leader following method. Second method will allow the agent to designate its own leader by searching for the nearest flocking agent travelling approximately the same direction as the agent. The comparison between the travelling directions is calculated by computing the dot product of their forward axis. This condition is necessary since the agent would want to stay in the direction it is moving where following an agent travelling in the opposite direction would require too much effort.

The leader following behaviour using the LEADER type agent depends on the arrival and separation behaviour to correctly portray the flocking behaviour. The arrival behaviour is a modified version of the seek behaviour where the agent slows down to a stop (zero velocity) as it arrives at a target point. The distance from which the agent starts slowing down depends on the distance parameter, where larger the distance the more graceful the deceleration process becomes. Implementing the arrival behaviour is simple where the desired velocity between the position of the agent and the target point is reduced according to the ratio of distance the agent is from the target position. Following was the code presented by [Reyn99] which was applied in the current implementation.

```
target_offset = target - position;
distance = target_offset.length();
ramped_speed = maxSpeed * (distance / slowing_distance);
clipped_speed = MIN(ramped_speed, maxSpeed);
desired_velocity = (clipped_speed / distance) * target_offset;
steering = desired_velocity - velocity;
```

In the leader following behaviour, arrival function is used to allow the agents to arrive at a point slightly behind the leader. Since using the arrival behaviour alone would clump all the agents to one position, separation function was applied to spread out the agents behind the leader.

Separation function prevents the agents from colliding with its neighbours where the tightness of the flock can be adjusted by the strength of the separation force (controlled by the separation weight and the radius). In the current implementation, using the separation function alone was not enough to have the agents from colliding with each other therefore unaligned collision avoidance was applied along with the cohesion and alignment steering to assign more flock like behaviours to the agent. It is important to remember when combining different steering forces from several steering behaviours that each steering force is normalised and weighted before it is summed up to give the finalised result. Since the followers of the leader agent should not block the path of the leader, regular checks to the future position of the followers were made with the leader's sensitivity zone, which is a bounding box along the forward axis of the agent. These checks were made in a similar method to the wall containment behaviour where the forward line probe of the agent was used to calculate the interception point of the line with the faces of the bounding box. When the interception occurs, the evasion behaviour is applied on the point of intersection. Whilst the ORDINARY agents are applying above steering behaviours to follow the agent, the LEADER agent is assigned to the wander behaviour where it moves around the space in a randomised movement and direction. In the current implementation, if there is more than one leader in the scene then the agents can choose the nearest leader and flock behind it. Currently there is no interaction between the leaders.

The second leader-following method also relies on the arrival behaviour to allow the agents to follow its newly designated leader at an offset slightly behind the leader. No other steering behaviour is needed for this method since agents cannot collide with its leader due to the slowing down effect of the arrival function. No two agents will have the same leader since agents always look for an agent that is the closest to its current location and moving in the approximately the same direction as the agent.

2.4.4 Steering behaviour for the Eat intention

Definition of food is different for the different types of agents where the focuser must adjust its attention according to their specific desires. FOOD agent type is used as the food definition for the ORDINARY agent type where if the intention is to eat then the agent will seek for the nearest FOOD agent in the neighbourhood. ORDINARY agent uses the arrival behaviour to retrieve and to eat food. Therefore as soon as the nearest food is located, the agent will slow down to fetch the food. In the current implementation, eat behaviour for ORDINARY type is more suited for a fish simulation then a bird since, it doesn't support the usual bird behaviours to fetch and eat food.

More naturalistic way would require the agent to land, walk towards the food, eat and fly away (or stay on ground) not plucking the food item from mid-air like the current simulation. In the simulation, as soon as the agent's bounding sphere intersects with the food's bounding sphere, the FOOD agent is removed from the food-linked list where value of the food consumed for the agent gets increased by one.

For the PREDATOR agents, the ORDINARY agent is the food definition therefore it will chase the ORDINARY agent when the intention is set to eat. Since the predator agent will chase the prey if the cost of reaching the agent is minimal therefore it will first calculate the cost of the nearby agents before choosing its target. To maximise its net rate of energy intake, the predator agent will compute the cost of feeding on a prey agent i like the following:

$C_i = d_i(1+\sigma 1 S_i + \sigma 2 E_i / \pi)$ where d_i is the distance between the predator and the prey i, if prey i is in a flock then S_i=1 otherwise S_i=0, the $E_i \in [0, \pi]$ is the turning cost (see Figure 2.10) and the $\sigma 1$ and $\sigma 2$ is the weights for S_i and E_i. This computation takes into consideration that the ORDINARY agent will be safe from getting hunted if the agent is within a flock, not in front of the predator and distance to the predator is great.

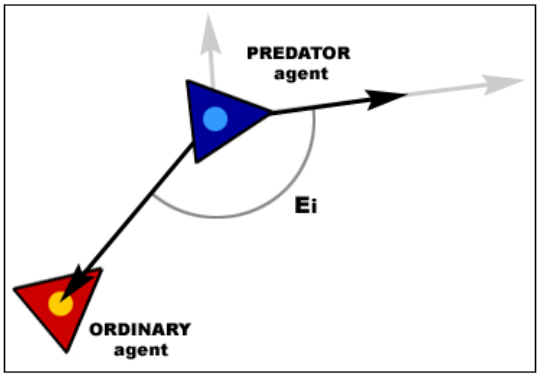


Figure 2.10: Turning cost E_i for the PREDATOR agent to get the ORDINARY agent

Once the target agent is decided, the PREDATOR agent will use the pursuit steering behaviour to chase the target. To view this simulation with a better effect, the maximum speed for the PREDATOR agent should be greater than the ORDINARY agent so that PREDATOR agent won't have to chase the agent forever (since the ORDINARY agent will be evading from the PREDATOR agent).

2.4.5 Steering behaviour for the Rest intention

When the current intention for the agent is to rest, the focuser will identify the ground and find a random spot on the ground that the agent can land on. This also involves in finding the height elevation for that random spot so the agent will be able to stay above the ground level. While the current intention is to rest, the wall avoidance check for the ground has to be disabled so the agent won't be stuck in between the ground the wall test distance. The moment when the focuser decides on the random spot, this target position must be stored in memory so that this can be accessed until the tiredness has been satisfied. The rest intention uses the arrival steering behaviour to approach the target position.

2.5 Locomotion

The locomotion is the last layer of the agent model, which is responsible for applying the steering force calculated from the focuser (and the behaviour layer) to the agent model. The approach used for the locomotion system is adopted from the simple vehicle model presented by [Reyn99]. Therefore the agent vehicle is based on the point mass approximation where the point mass depends on the position and the mass property of the agent. Agent vehicle model has velocity, maximum force value, maximum speed value, orientation, mass and position as its parameters, where on every frame update the steering force gets applied to the agent's point mass. From the steering force the acceleration can be computed where this acceleration gets added to the old velocity to produce a new velocity. This is better explained in the code below:

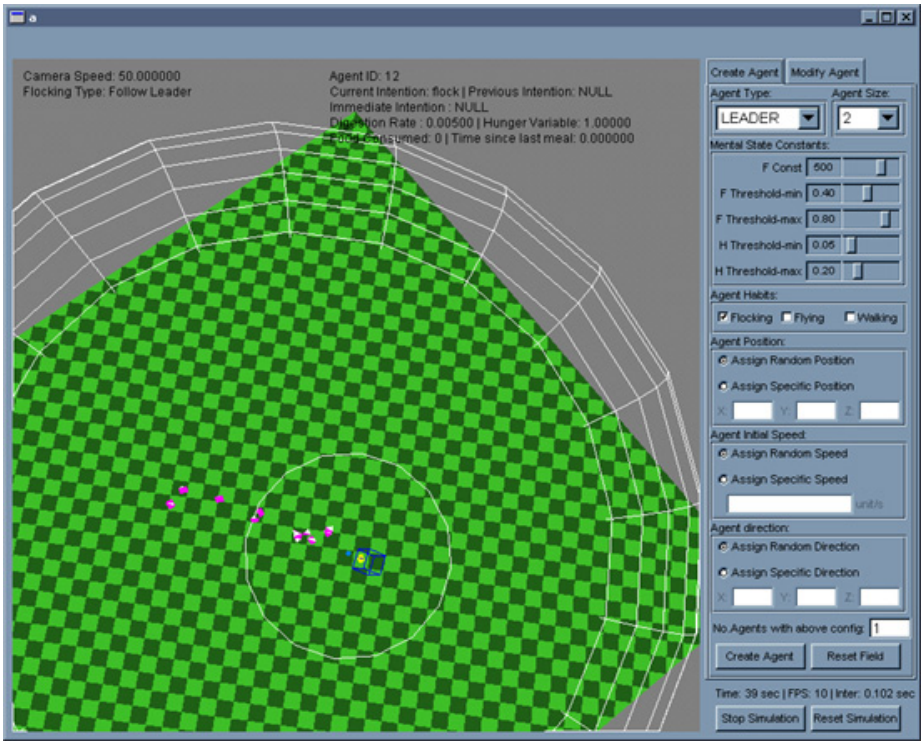
```
Steering_force = truncate (steering_direction, max_force)
Acceleration = steering_force / mass
Velocity = truncate (velocity + acceleration, max_speed)
Position = position + velocity
```

The magnitude of the new velocity becomes the new speed where the new forward axis is the new velocity normalised. Therefore knowing the new forward axis, by using the old up axis as an approximation of the new up axis, the side axis of the agent can be found by cross product of new forward axis with the approximated up axis. From the new side and the new forward axis the new up vector can be calculated using cross product. To implement the banking behaviour of the agent, it is assumed that the vehicle's inverse up vector is aligned to the gravity due to the centrifugal force during the turn. Therefore the new up vector should be aligned with the

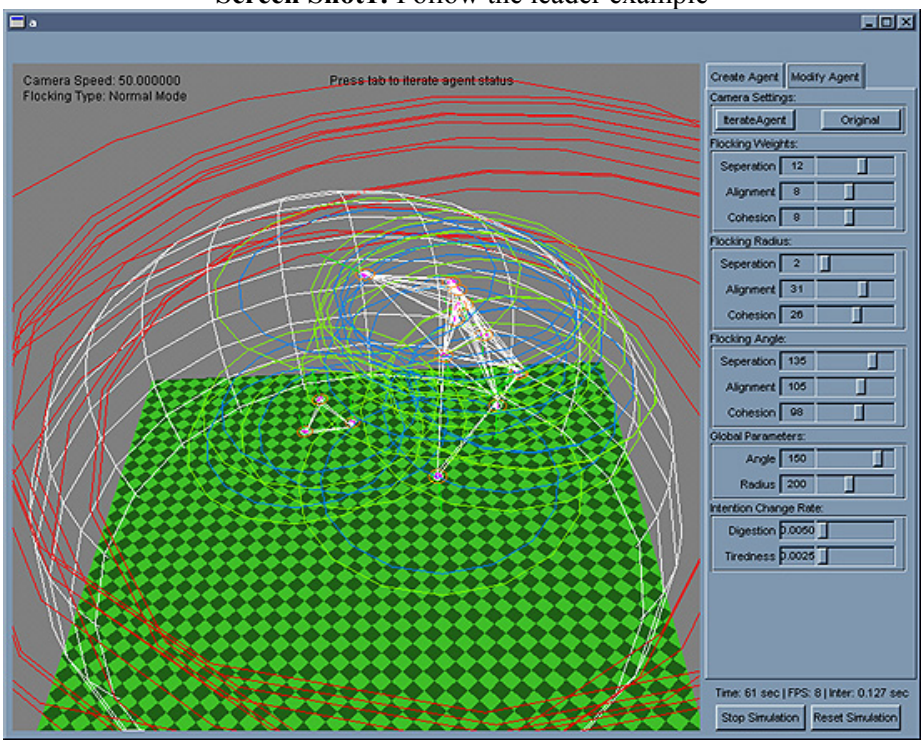
centripetal force. Banking is obtained by computing the approximated up direction be the weighted sum of steering acceleration, gravitational acceleration and the old up [Reyn99].

Section 3

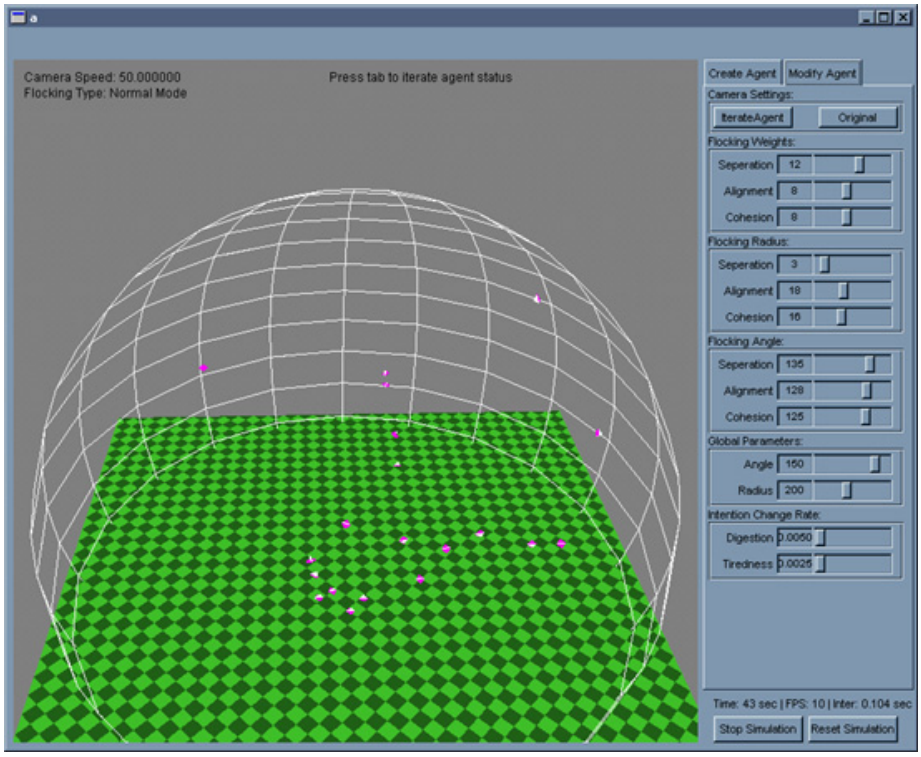
Screen Shots



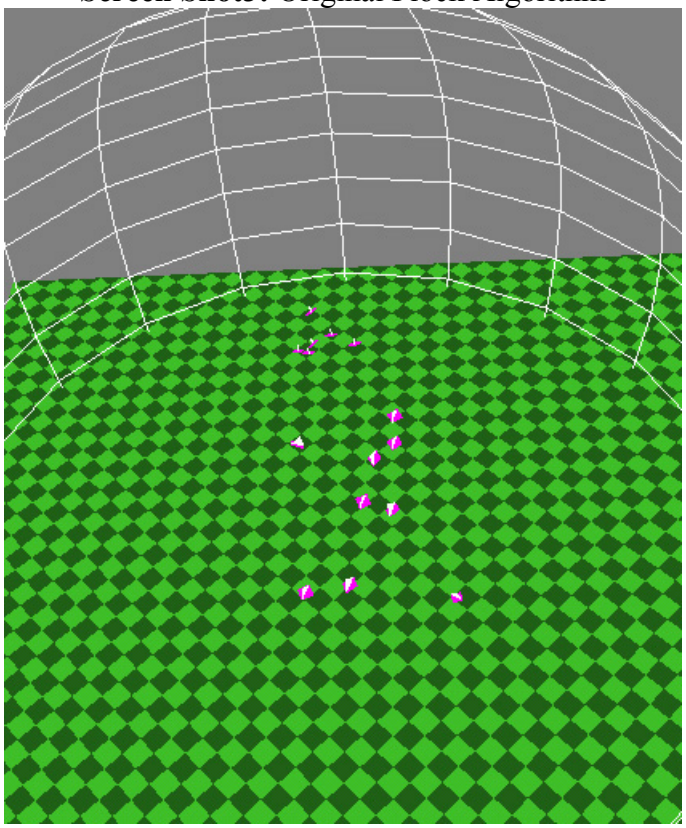
Screen Shot1: Follow the leader example



Screen Shot2: Flock Neighbour Network – Flocking Radius



Screen Shot3: Original Flock Algorithm



Screen Shot4: Original Flock Algorithm

Section 4

Conclusion and Future Work

This project has presented an experimental version of the autonomous agent system for birds, which can simulate behaviours such as flocking, evading and eating. The current agent's intention generator is based on the Tu's model where intention can be selected independently without much input from the user. The project managed to incorporate some A-life concept by allowing the agents to preserve previous intention in case of sudden interruption of the current intention. Where also the bird's properties were analysed to recreate their behaviour in the virtual world environment. There are still a lot of gaps in the current implementation of the project that would need to be completed in order to make the simulation look realistic. Since the project did not concentrate so much on the efficiency of the algorithms therefore only a limited number of agents can be tested at one time. It is difficult to objectively measure how valid the simulation of these birds are where only comparison that can be made is to test how well some of these steering behaviours work. For example, for the separation behaviour, the amount of collisions that occur in relation to the separation weight and radius can be tested. There are still some extra steering behaviours that are left out from the [Reyn99] such as path and flow following where this could be useful to create bird simulations that are more restricted to the user's input. An interesting future work would be to incorporate with this model the plausible simulation so that it can generate a constrained animation of bird agents. In terms of brain model, fuzzy logic could have replaced the Tu's model to give more naturalistic intention generator but due to the short amount of time available to work on the project, this was not implemented.

Bibliography

[Reyn87] Craig, W. Reynold, "Flocks, Herds, and Schools: A Distributed Behavioural Model", Computer Graphics, Volume 21, Number 4, July 1987.

[Reyn88] Craig, W. Reynold, “Not Bumping Into Things”, Course Note on Physically Based Modelling at SIGGRAPH 88, August 1988.

[Reyn99] Craig, W. Reynolds, "Steering Behaviors For Autonomous Characters," Proceedings of Game Developers Conference, (1999).

[Reyn00] Craig, W. Reynold. “Interaction with Groups of Autonomous Characters”, in the proceedings of Game Developers Conference 2000, CMP Game Media Group (formerly: Miller Freeman Game Group), San Francisco, California, pages 449-460.

[Tu1994a] X. Tu and D. Terzopoulos, Perceptual Modeling for the Behavioral Animation of Fishes, Proc. Pacific Graphics '94, World Scientific Publishers, Singapore, pp.165-178

[Tu1994b] X. Tu and D. Terzopoulos. "Artificial Fishes: Physics, Locomotion, Perception, Behavior". Proc. SIGGRAPH'94, Computer Graphics, July, 1994.

[Tu1994] X. Tu, “Artificial Animals for Computer Animation: Biomechanics, Locomotion, Perception, and Behavior”, PhD Thesis 1996