The Australian National University

Faculty of Engineering and Information Technology
Department of Engineering

The Examination and Exploration of Algorithms and Complex Behaviour to Realistically Control Multiple Mobile Robots.

Author:	Duncan Crombie
Supervisor:	Dr Robert Williamson
Examiner:	Jon Kieffer
Date of Submission:	29 October 1997

Abstract

Multirobot systems are becoming more and more significant in industrial, commercial and scientific applications including: plant maintenance, warehouse operation, space missions, operations in hazardous environments and military applications. Localised control has advantages over hierarchical control because the robots can be autonomous.

The goals of this project are to closely examine the algorithms that can control multiple mobile robots and to study the complexity of such systems. Ultimately it would be useful if the algorithms are implementable in mechanical platforms. This analysis was made possible through the creation of a Java&#129; application to run dynamic simulations.

The applications created as part of the project show how a group or groups of robots can be controlled by various algorithms. Animation was chosen as the preferred mode of analysis due to the overwhelmingly complex nature of large groups of mobile robots. Each mobile agent is programmed with the same simple algorithm that cause the group to exhibit a complex behaviour.

The report that follows focusses on the behaviours that are possible through the implementation of localised control. It also examines the required mobility of the mechanical base and resolution of detection devices. The algorithms presented make few assumptions about the type of robot used.

Demonstrated in the following chapters are examples of algorithms for Flocking and for Forming Lines. The process of designing for localised control is explained and a structure is presented on which further work can be built.

The results show that it is possible to create complex behaviours using relatively simple algorithms. Using methods presented in this report, the implementation in a physical group of robots is made quite feasible.

Table of Contents

I. Introduction

The nature of this project is theoretical rather than practical. While the report examines the potential applications and attempts to demonstrate the capability of multiple mobile robots, these algorithms are not yet implemented in real robots. The mode of demonstration is to use simulated robot agents referred to as 'boids' to implement various algorithms.

The concept of 'boids' is not new and was initially developed by Craig Reynolds[1] who wrote the first detailed paper on this subject. The basic concept he proposed was to replicate the behaviour of birds, specifically the species such as sparrows which are observed to move in flocks. The behaviour and internal mechanics of a flock or swarm are quite complex to analyse but can be demonstrated using artificial entities.

The potential uses for this type of programming vary from dynamic sculptures to military applications. The inspiration for starting the project was a water sculpture using multiple mobile robots to emulate the behaviour of fish in the water. The fluid environment in this case would cause some difficulties in turning and detection but essentially the same algorithm could be used.

Of a more serious nature, military applications could include offensive robotic swarms that home in on a target and attack en masse. The approach of a large number of mobile robots or drones could confuse and more easily infiltrate a security or defence system than a single large unit. Also being researched by the US military is the concept of replacing military satellites with 'constellations' of small mobile units[4]. This makes the destruction of the satellite much harder as it could still function with reduced numbers.

This report examines the concept of multirobot systems in terms of the complexity and to some extent the computation power of such systems. In the following chapter it will be shown that the system of boids was too complex to model analytically. The path followed has been to use simulations in discrete time.

The algorithm itself is executed locally in each mobile agent of the flock or swarm. This means that each agent makes decisions depending on what is detected in the immediate environment. Communication between robots has been implemented in other studies but for this project it was ruled out as an unnecessary complexity.

Communication can be essential when the environment is very complex. For example in a warehouse with many passageways and dead ends it would be useful for an agent to be able to signal its siblings. Possible protocols would be: STOP, WAIT, GO AWAY and HELP. A simple language like this would enable robots to inform each other of special situations and perhaps initiate and coordinate a higher order function such as moving or lifting a large crate.

A number of algorithms were implemented in the course of this project. Two complex behaviours are presented in detail in this report. The flocking behaviour described above is the most commonly demonstrated phenomena. A number of steps were taken to optimise the flocking the behaviour and make the implementation less dependant on sensory devices.

The forming lines algorithm is a new concept and shows the flexibility of multiple robot systems. Both of these behaviours are achieved without the use of 'transmitters' or 'beacons' which are commonly used to direct agents in existing systems.

A copy of this report will be available on the Internet and the disc attached contains copies of all demonstrated behaviours.

II. Complexity of Multirobot Systems

There is a lot of interest in scientific circles in the study of complex systems. Recent developments have shown that organic compounds such as DNA can be used to implement algorithms and solve complex problems.

There is a question as to whether a similar development could be made using some form of swarming algorithm. There are two ways to approach this problem. The agents can be made more complicated in order to perform in a more complex way. The other approach is to make do with the existing simple agents and create an environment where computation is possible.
The agents used throughout this report are referred to as 'boids' and have very simple properties. They move at a constant speed and have a limited detection range, they are also restricted in how quickly they can turn. With these restrictions is quite hard, if not impossible, to implement a calculating device.

A. Computational Power

The simplest implementation that uses these properties is logical gates. Shown below are two types of 'gates' that represent logical AND and logical XOR. The concept is that a boid represents a TRUE value and no boid represents FALSE. The output of the gate is TRUE if any boids pass through the gate.

The logical function AND should output TRUE only if both inputs are TRUE. The function XOR should output TRUE if only a single boid is input to the gate.

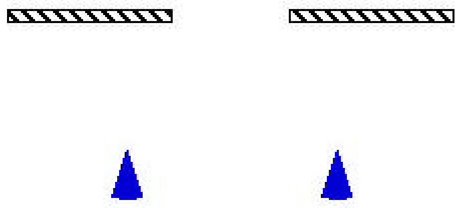


Figure 1: Representation of an AND gate.

This scenario represents a logical AND in that a boid will only appear at the other side of the gate if both are present. With either boid absent the lone boid will be blocked by the wall. With two boids, they will be attracted and take a common path through the gate.

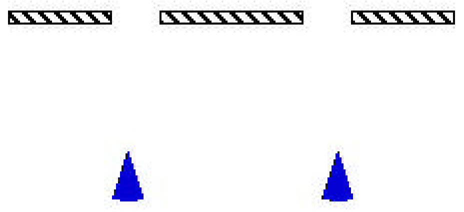


Figure 2: Representation of an XOR gate.

The figure above shows how we can implement an XOR gate. In this case a boid only appears ar the other side of the gate when it is alone. With two boids present they will be attracted to each other and together be blocked by the wall. This model could be changed to an OR gate by removing central portion of the wall

The idea of having no boids is where this model fails. How can the 0 output of a gate be recognised. Also, in the case of the AND gate what can be done with the two boids that emerge. The solution is to make each gate a closed system with input and output channels. At certain times a boid would emerge representing a TRUE value or not emerge representing a FALSE value.

For example a NOT gate would emit a boid every timestep unless a boid had been passed to it during that timestep. This adds some complications to the implementation as boids have to be made to appear or emerge at set times. The AND and XOR gates would be modified as shown below:

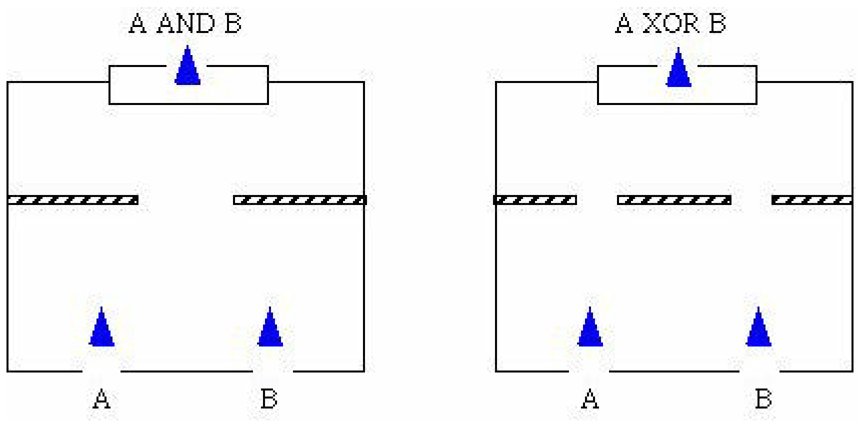


Figure 3: Modular representation of logical gates.

The difference between these models and the previous are that the boids that enter the gates are not necessarily the ones that emerge. The area behind the gate detect boids that pass through the gate and emit a boid to represent a TRUE value. The exception, as mentioned above, would be the NOT gate which would emit a boid when the input was FALSE (no boid entered).

A NAND gate could be implemented using the AND followed by a NOT. With NAND gates we can create any logical functions. The above analysis shows that swarming agents are to be considered quite seriously in terms of their ability to solve problems. Obviously this could be built on and examined in more detail but that is beyond the scope of this project.

B. Complexity of the swarm

The control of swarming robots is complex enough that all existing research has been done through animation or computational analysis. It would be interesting if we could prove that the algorithm is in fact impossible to model. The closest existing mathematical structure is the three-body problem which occurs in dynamical astronomy.

The three-body problem is a sort of intermediate dynamics and mathematics problem between the two-body problem, which is straightforward and easy to solve analytically, and the n-body problem which has just a few solutions for some very general properties.

The general problem of the motion of three bodies (assumed to be point masses) subject only to their mutual gravitational attractions has not been solved analytically. The two approaches to this problem are:

1. solve the problem using computational analysis/animation, or
2. assume one of the masses is infinitesimally small compared to the other two and thus cannot affect their motion (the restricted three-body problem).

Clearly the flocking system does not have quite as many interactions as a planetary system and momentum properties have not been implemented at this stage. While these points would make the boid system easier to analyse there are some features that combine to make the problem unsolvable for more than two boids. It is also not feasible to make an assumption similar to point 2 above.

In a physical implementation of any boid system each agent would be running a separate internal processor. Calculations for turning the robot would take place at discrete times in each boid but the members of the flock would not perform calculations at the same time. This uneven discretisation is responsible for making even some computational models unreliable.

III. Localised Control

The most commonly accepted form of control is distributed control in which a centralised processor coordinates the motion of all agents. This means that the robot agents have to be in contact with the controller at all times and the controller must divide its processing time between the agents. Adding additional agents adversely affects existing agents by slowing the processor.

By converting to localised control the problems mentioned above are corrected. The robot agents contain their own processors and do not have to remain in contact with the controller. The disadvantages are that the robots become more complex and must be able to react on their own to different situations.

Research into cooperative robots using localised control has produced three distinct types of control:

1. Traffic control: When multiple agents move within a common environment, they typically move to avoid collisions.

2. Cooperative manipulation: This can cover a number of scenarios including the most common examples of pushing or lifting a box[3].
3. Foraging: This title is self-explanatory and could cover anything from cleaning up toxic waste to locating survivors after an earthquake.

The approach of this project has been to restrict study to the first category in this list because this class is not dependent on objects in the environment.

The problem of producing complex behaviour patterns in multiple mobile robots is one of experimentation and iteration. The first step is to look at the desired behaviour and then try to determine what the individual agents would have to accomplish to be part of the proposed behaviour. Once this is established the simulation can be run and modifications made if the result is not satisfactory.

Many useful behaviours are found in the animal kingdom where we can observe behaviours such as flocking, following and foraging. More difficult to define are the behaviours that would be useful in an industrial environment such as a factory or warehouse. The first process described here is the flocking behaviour.

The observations of flocks of birds and schools of fish tell a lot about what is required to implement this behaviour. The points to notice are:

1. The birds/fish form a distinct group,
2. The members of the flock/school tend to be evenly space, not bumping into each other
3. The flock/school tends to have an overall direction.

The way to approach an algorithm is to enforce these three behaviours in each individual boid as described in the next chapter. Once a basic model is implemented it is necessary to refine the boid properties in order to get the best behaviour. This is a matter of changing the weighting factors of different force components. For example, if the boids are bumping into each then the repulsion is increased. This process is described in a later chapter.

The second behaviour described in this report is forming lines. The key points to notice in this behaviour are that the boids only want to have two neighbours in order to form a line. It would be useful if the boid formed a line with its two neighbours and also kept a reasonable distance from them. As with the previous example the algorithm has been implemented and then refined through an iterative process.

IV. Flocking Algorithm

Flocking occurs in nature and is exhibited by birds, fish and some insects. The sight of a migrating flock of birds is one we are all familiar with. This behaviour is based on the principal that there is safety in numbers and the whole is more important than the parts.

A flock of birds gives the appearance of a larger entity and dissuades attackers. If a flock or swarm is attacked, the survivors can scatter and regroup at a safe distance. This scattering can confuse predators and prevent them from capturing more than one or two members of the flock.

The three rules that implement flocking are:

1. Cohesion: Each boid shall steer to move toward the average position of local flockmates.
2. Alignment: The boids will align to the direction their neighbours are travelling.
3. Separation: All boids in the flock will maintain a separation distance from their siblings.

The effect of these rules when implemented in multiple mobile agents is to cause flocking or swarming. Removing or disabling one of the rules removes the apparent cooperation between swarm entities and makes flocking impossible.

The demonstration presented in this report involves some additional features. There can be two schools of boids that flock but remain distinct from each other. To add some realism to the simulation, barriers can be introduced which the boids will avoid by flocking around or away from.

Initially an existing applet[2] was acquired that provided a useful structure for this type of application. The interface required a lot of work and the algorithm that was implemented was not very effective. Assumptions were being made about being able to detect the speed of other boids. This can be quite difficulty.

In a relatively small mobile robot it is difficult to implement a high quality detection system. The most common way to determine velocity is to numerically differentiate the position. When there are small errors in the position estimation then the velocity figures can be quite misleading. The algorithm presented below does not rely on velocity but does require the direction of neighbours to be determined.

A possible solution would be to have beacons present on the front and rear of each robot. Once the distance to the boid has been approximated then the direction of travel can be determined by the angle between the two beacons and which one is closer.

The algorithm that controls flocking calculates the desired direction of travel for each boid. There are two sets of rules in this function, one for boids in relation to siblings and another for barriers or boids of another school.

The algorithm runs as follows for each boid in the flock:

1. For each detectable neighbour or barrier

2. If that neighbour is a sibling (of the same colour) then the target point is a weighted average of alignment and either attraction, or repulsion if the boid is too close.

3. Else if the neighbour is of another colour or a Barrier, the target point is in the opposite direction to the other Bird or Barrier

4. When all the detectable boids have been accounted for, take a weighted average of the various target points.

5. Move towards this point.

The results presented later in this chapter will show this algorithm in action. The following section describes how the weighting factors were decided to give the most effective flocking behaviour.

A. Weighting Factors

The flocking is achieved through a form of vector addition. The various boids and obstacles in the vicinity of a boid effect it in different ways. As shown below there are three types of effects: alignment, attraction and repulsion.

Alignment and attraction are only present when two boids of the same colour are in close proximity. The repulsion effect comes into play when a boid approaches a barrier or another boid of a different colour. Repulsion also takes place when boids of the same colour move too close to each other.

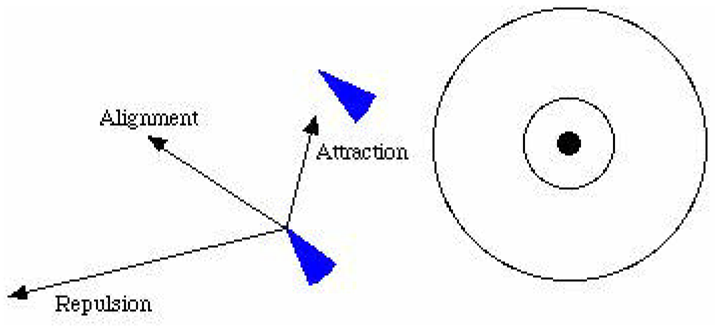


Figure 4: Vectors associated with the flocking algorithm

Using weighted averages as a scheme for combining the effects of all detectable boids allows us to make different effects dominant under different conditions. For example, when two siblings first detect each other at the limit of the detection range, it is more important for them to get closer to each other than to align directions. Similarly when two boids are too close the most important component will be repulsion.

After much experimentation, the following values have been settled on. For sibling boids the alignment component is weighted by a fixed value of 100 and the attraction/repulsion component is weighted by:

$$w_i = 200 \times \left(1 - \frac{\text{distance}}{\text{minDISTANCE}}\right)^2$$

if the boids are too close, or

$$w_i = 200 \times \left(\frac{\text{distance} - \text{minDISTANCE}}{\text{maxDISTANCE} - \text{minDISTANCE}}\right)^2$$

if they are outside the separation distance. The following graph shows how the attraction/repulsion component will dominate when it is most important. In this diagram the detection distance has been set to half the detection distance giving a symmetric parabolic curve as shown below.

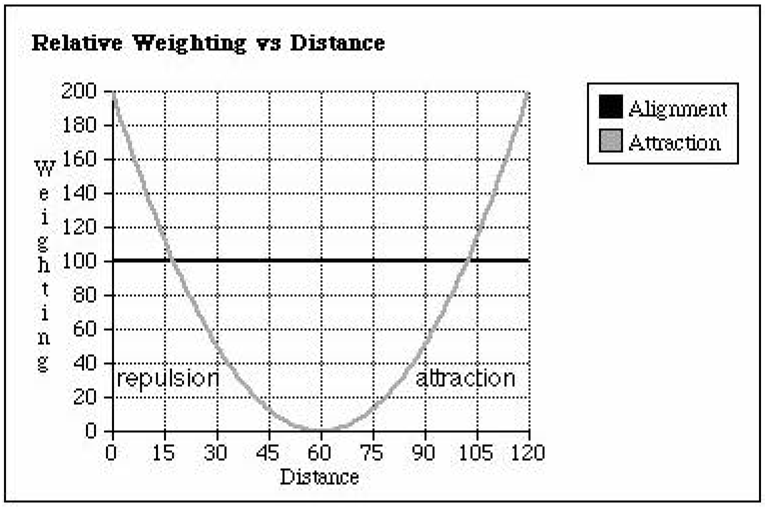


Figure 5: Relative weightings for boids due to sibling proximity

If the boids are moving relatively fast then we may want to keep them closer together so that they do not lose each other. This can be achieved by decreasing the separation distance from 60 in this case to 30. The graph would then be skewed to the left, the end points would be the same but the end of the parabola would move left to lie at 30.

The most important feature of the Attraction curve above is that there is the stationary point at the separation distance. In an earlier swarm model the attraction/repulsion curve was of constant magnitude. This meant that when the boid was close to the separation distance it was propelled in alternating directions. To ensure that the boid motion is smooth it was important to remove any discontinuities and use a smooth curve.

The dominant effect in most cases is that of alignment. The attraction component becomes as much as double the weight when the boid is at the periphery of detection. Similarly the repulsion component prevents the boids from becoming too close. The combination of these two components is scaled to a vector of length 100.

The figure below shows how the target point is determined for a single neighbouring boid. The scheme devised here is such that when the boids are at the desired separation distance the attraction component will be zero and the target point will be determined by the alignment component. In this way the boids tend to align neatly without too much changing of direction.

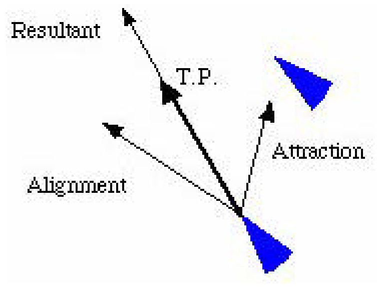


Figure 6: Sum of attraction and alignment

In terms of energy methods we can imagine each pair of boids as being coupled by a linear spring.

The parameter minDISTANCE represents the equilibrium position. When the distance between two boids diverges from this value, there is an increasingly strong impulse applied to restore equilibrium.

The repulsion weighting for barriers and boids who are not siblings is weighted higher so that the obstacle avoidance behaviour dominates in situations where flocking would result in collision with a barrier. As mentioned above, the weighting of the components due to sibling is 100. The weighting for repulsion from other boids and barriers is:

$$w_i = 1000 \times \left(1 - \frac{\text{distance}}{\text{maxDISTANCE}}\right)^2$$

This is again best displayed graphically as shown below:

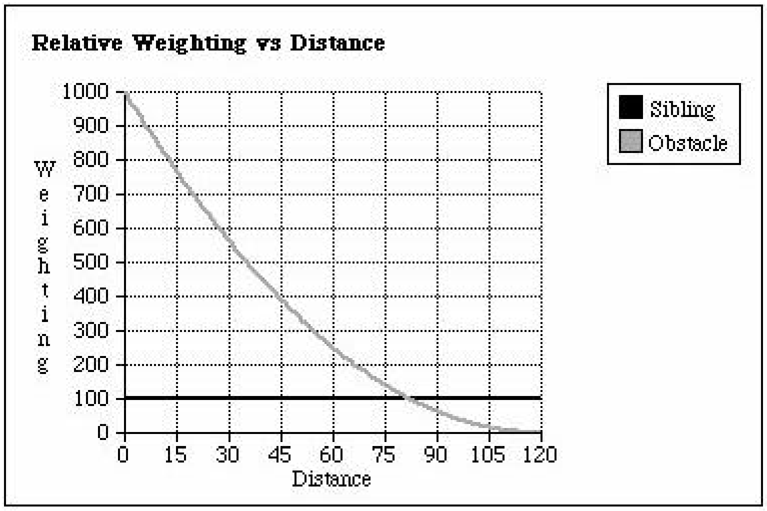


Figure 7: Relative weighting for boids due to barriers

This graph shows that in most cases the effect of a barrier will be weighted much higher than the previously described components. One drawback with this method is that in a very large group of boids the alignment component is multiplied by the number of neighbours detected. In the simulation it is thus possible for a group of ten or more boids to 'ignore' a barrier placed in their path.

As stated previously the system is solvable for a system with only two boids. When there is a flock of n boids, however, there can be as many as:

$$\sum_i (n-1)$$

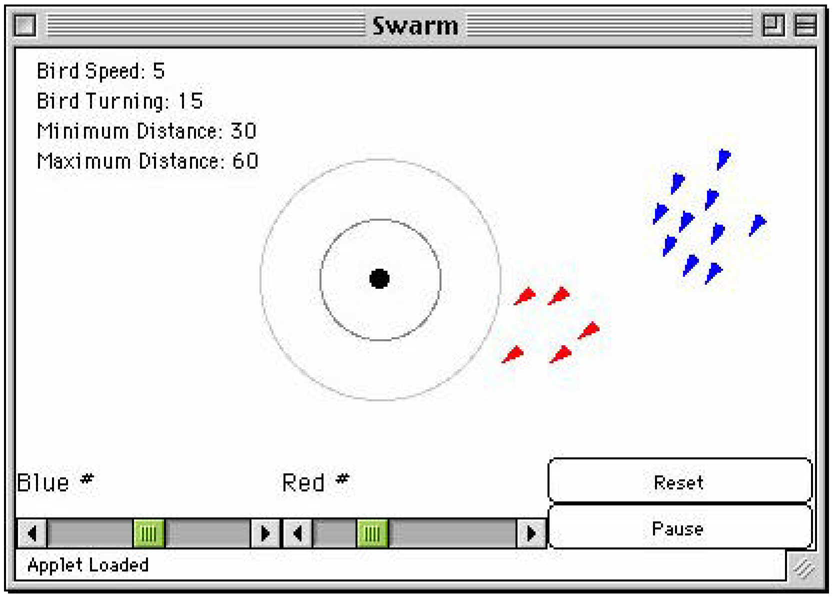
relationships. For a small group of five boids, each would be affected by up to ten forces. This is another reason why to have a clear understanding of the flocking behaviour it was necessary to resort to computer simulation.

To summarise this section, there are a number of components affecting the direction of travel of each boid. For siblings, there is a weighted average of attraction/repulsion and an alignment component. The sum is scaled to a vector of length 100. For other boids and obstacles there is the repulsion effect described above, weighted by up to 1000. The effects of all boids and barriers in the immediate environment are averaged and the boid turned towards the resultant point.

B. The Swarm applet

The applet discussed in this section appears as shown in the illustration below. This depiction is a screen shot of the applet running on a the Macintosh Applet Runner. The appearance is similar inside the Netscape browser and can also be seen through the Applet Viewer which is available on the ANU Department of Engineering computer system.

In this figure the values displayed for Bird Speed, Bird Turning and the two distance parameters are the default values of the applet. To change the first two values the user can use the arrow keys on the keyboard. The distance parameters are modified by holding down shift while using the arrow keys.



Bird Speed: 5
Bird Turning: 15
Minimum Distance: 30
Maximum Distance: 60

Applet Loaded

Figure 8: The Swarm applet

In this figure above all the components of the Swarm applet are visible. The lower section contains a control panel while the upper portion of the applet is a canvas used to display the animation. The applet window can be resized by dragging the lower right corner.

The flock in this case consists of a group of five red boids, a group of ten blue boids and the single barrier object represented by concentric circles. The boids have separated themselves into two schools according to colour.

To control the number of red or blue boids the user can use the scrollbars. This will increase or decrease the population by one. The Reset button will remove any barriers and restore the number of boids to the default value. The Pause button halts the animation until it is pressed a second time.

In the following sections a number of test cases are displayed and explained in detail. It must be emphasised that the flocking behaviour is dynamic and not easily conveyed in printed form. Through the project Internet site or the software included with this report a better understanding of the behaviour can be gained.

C. Boid Parameters

Here we see how the maxTURN parameter directly effects the path taken by the boids. A single boid is used and sent towards a barrier with different values of maxTURN. As expected the higher the maximum turn value, the quicker the boid turns.

From the extreme of no turning, where the boids passes straight through the barrier, the angle turned by the boid gradually increases and its closest approach to the barrier becomes greater. This parameter can be used to determine whether different platforms are suitable for this type of control.

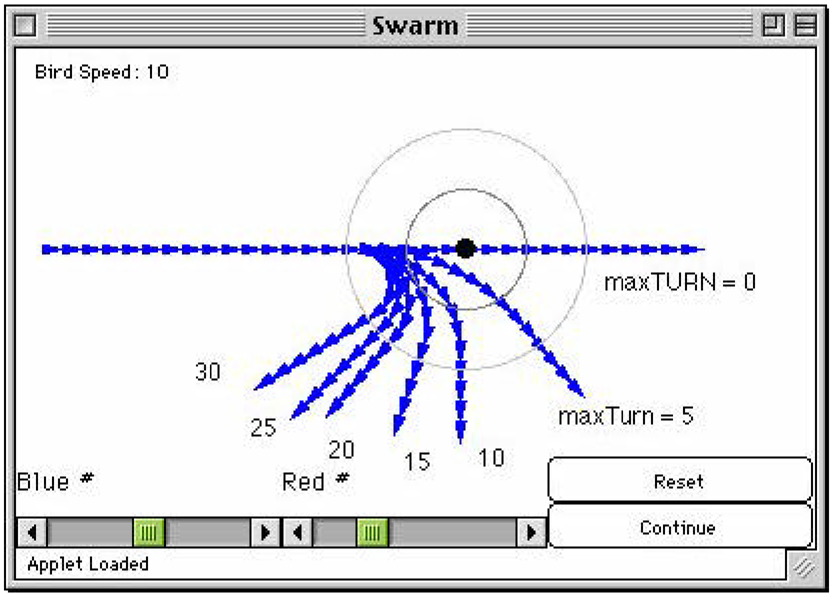


Figure 9: Changing the boid parameters

The other parameters that would change the shape of these paths are: the boid speed and detection distance. Increasing the speed makes control of the boid harder as it can move further during each timestep. Increasing the detection radius would cause the boid to turn earlier and not come so close to the barrier.

The speed parameter mentioned above is not necessarily the physical speed of the boid. What this represents is the distance the boid will travel between each direction calculation. This means that the speed parameter could be decreased to represent an increase in processor speed. Similar the turning parameter defines how far in degrees the boid is able to turn each timestep.

D. Alignment

The first and perhaps the simplest way to demonstrate the flocking algorithm involves just two boids. As shown in the figure below, the two boids start in separate corners and move towards each other at an angle. When the boids come within the detection range the flocking algorithm says that they must align their direction of travel. It is clear from the picture that the final state has the boids moving in parallel paths at the designated separation distance.

The smoothness of this transition is due to the design of the attraction/alignment weighting factors discussed earlier. In an earlier model the boid paths would tend to zig-zag before finding an equilibrium position.

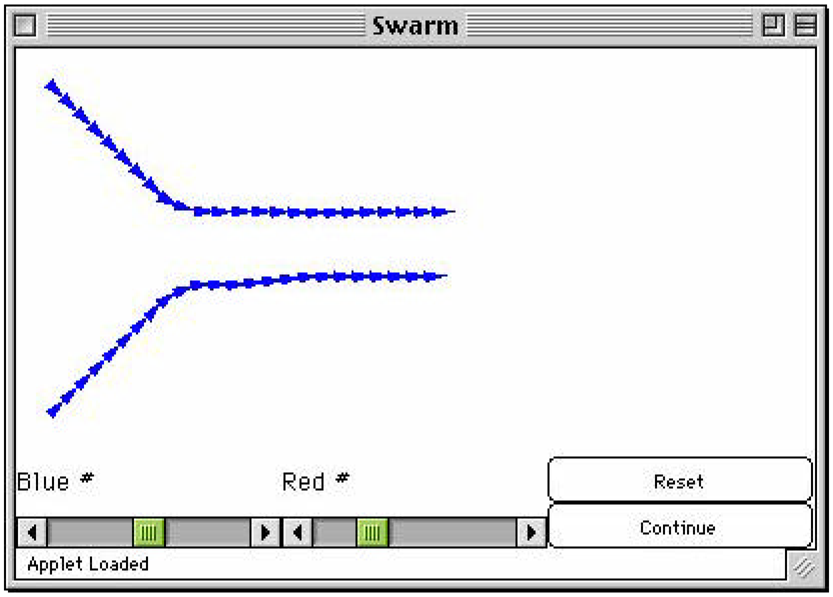


Figure 10: Boid alignment

In the diagram below we see an analysis of different sections of the boids path. When they first detect each other there is an attractive force bringing them together, and a force making them align. The result of adding these two effects each timestep is the curved path seen above.

The boids approach until they are at or near to the equilibrium distance. Once they are at this distance, there is no need for them to move with respect to each other so they take parallel paths and would keep those paths indefinitely.

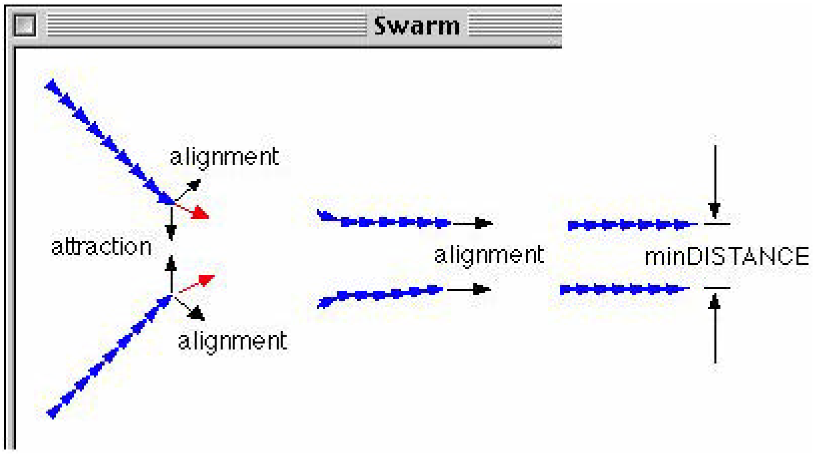


Figure 11: Flocking equilibrium

In this simple case it is easy for the boids to reach an equilibrium. With a few more boids and barriers the behaviour becomes chaotic and predicting the motion of the flock becomes impossible.

E. Flock obstacle avoidance

The case demonstrated here introduces a barrier to the previous example. The boids start in identical positions as the above case but the barrier placed in their path forces them to take evasive action. This should demonstrate the way a flock avoids a stationary barrier.

As mentioned previously, the depiction of the barrier within the applet is used to display some of the flocking parameters, namely the separation distance and detection range. The separation distance, indicated by the darker circle, shows the equilibrium distance sought by boids who are siblings.

The outer circle indicates the range at which all boids can detect objects in their environment. A closer examination of this figure shows that the boids initially start to turn when they come within this distance of one another and then again when they come within the outer circle of the barrier.

When the boids reach the barrier, they do not separate as one might expect. The upper boid reaches the outer circle of the barrier slightly before the lower boid. The response of the first boid (boid 1) to detecting the barrier is to turn upwards, away from the barrier. Once this occurs boid 2 is induced to turn for a number of reasons.

1. Alignment: The boid turns to match the alignment of its sibling,
2. Attraction: The boid is attracted to its sibling as the distance between them increases,
3. Repulsion: When the boid has turned a small distance, the barrier is on the other side.

The first two points induce the lower boid to turn upwards. This is counteracted by the effect of the barrier which is to turn away (down). Due to the weighting scheme described previously, the overall effect is to turn up. Once the turn is started, the barrier is on the other side so boid 2 turns hard left.

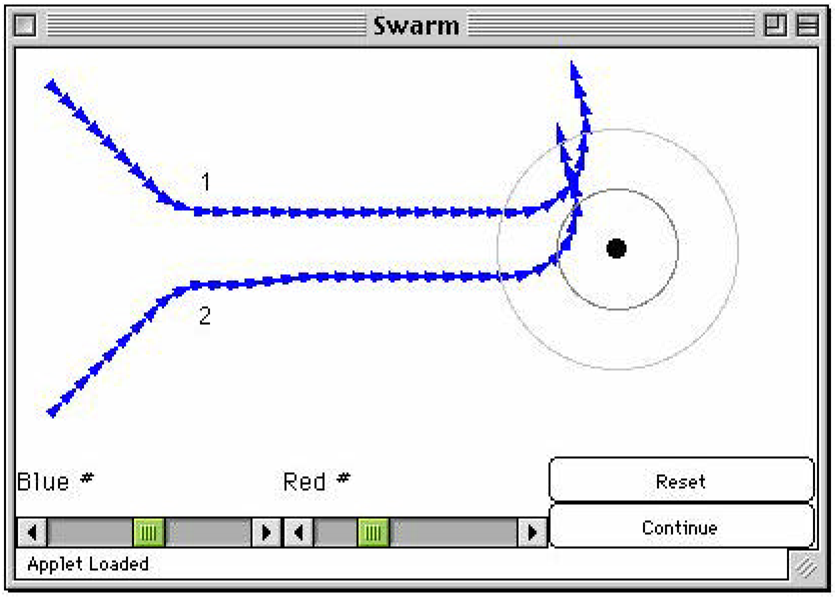


Figure 12: Obstacle avoidance

The alignment effect is very important in implementing obstacle avoidance. When a large group of boids approaches a barrier only the leading boids will come within detection range of the barrier. Following boids will react to the movement of the leaders and turn early, without even detecting the barrier. In this way a form a communication occurs between the boids. By reacting to their environment the leading boids change the environment of following boids.

F. Avoidance between boids

The following picture shows three boids approaching from different directions. The boids coming from the top and bottom of the canvas are of the same colour so they align their direction of travel. The boid coming from the right is a different colour and as such is repulsed from the other two and is turned around. Boids that are siblings will always move to flock together unless there is an overriding repulsive force due to a close barrier or boid of another colour.

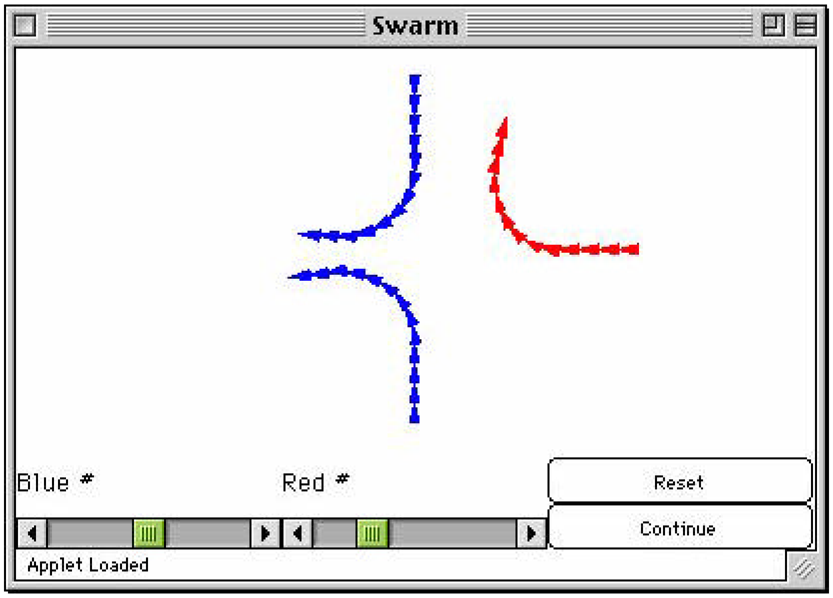


Figure 13: Avoidance between boids of different colours

The figure below shows in detail the forces that are acting on the three boids at a single point in time. The boid on the right is repulsed from the two other boids with the closer boid having the greater effect. From the diagram it is clear that the resultant point will be to the right of this boid forcing it to turn as shown above.

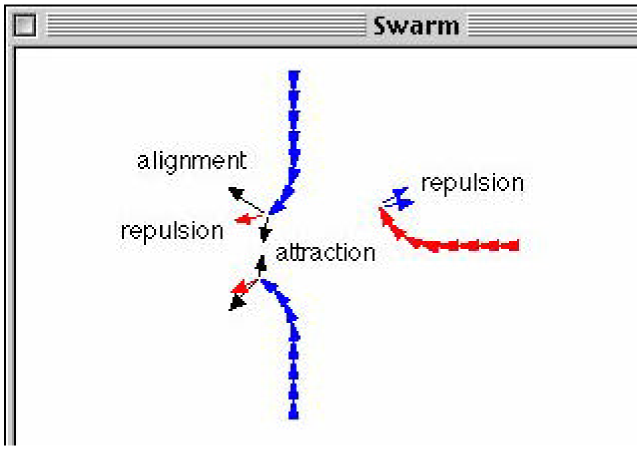


Figure 14: Breakdown of avoidance behaviour

The distance between the two siblings is greater than the separation distance so they are attracted and at the same time align their directions. The third force on these two boids is to move away from the right-most boid.

As with a spring, if the boids approach the each other at too great a rate they can overshoot and oscillate before finding equilibrium. The rounding of boid positions to integer values and the weighting factors described in a previous section serve to damp these oscillations and between two boids equilibrium is quickly established.

V. Forming Lines

To illustrate the flexibility of this application a variation on the flocking behaviour is demonstrated in this chapter. The Forming Lines applet displayed here was created by changing only a small portion or about 30 lines of the program code. In some respects this applet is less complex than the previous model. The rules can be related to three specific cases in addition to the obvious case which is to go straight if there in nothing within detection range.

A line is a two-dimensional object so in this application we need only consider a boid having two neighbours. If there are more than two boids within detection distance then the two closest will be most important. When each boid forms a line of three with its nearest two neighbours then the overall effect is a continuous line of boids as shown below.

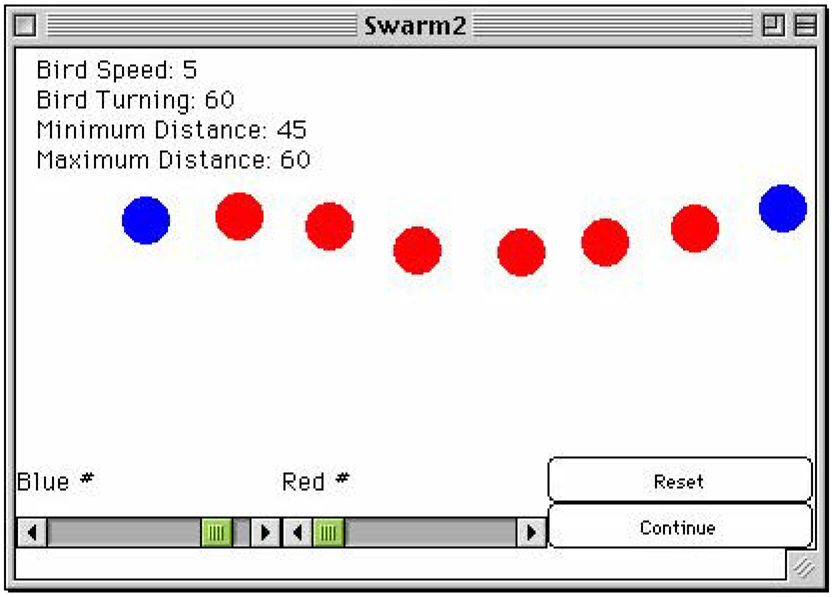


Figure 15: Forming Lines applet

The boid colour in this applet is an indication of the rule that the boid is following. A boid that can only detect one neighbour is coloured blue and those with two or more neighbours display a red colour. In the figure above the end boids should appear darker than those between them. A boid that has no neighbours within its detection radius is coloured grey and will travel in a straight line.

To start with it must be realised that even when the line is formed there will be two different behaviours exhibited. The boids at the end positions will be following slightly different rules to those arrayed between them. What we want to avoid is boids swapping places ie. if the end boid moved around it's neighbour it could conceivably displace it and move up a position in the line.

A. Rule for End Boids

The first rule concerns the end boids. As they can only detect one neighbour then the best they can do is maintain their separation distance from that boid. Note that this only constrains their motion to the perimeter of a circle. The two cases that invoke this rule are shown below.

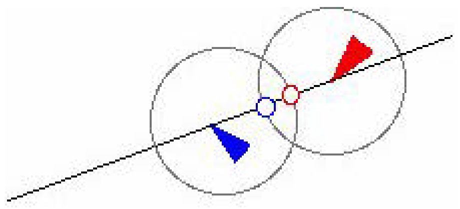


Figure 16: In-line attraction

In this case the boids are further apart than the separation distance. The algorithm for forming lines tells each boid to move towards the other along the line that joins them. It should be noted that although the algorithm gives a point for the boid to move to, this is not achieved immediately as the boid must first turn in the given direction. The act of turning tasks the boid away from the point at which the original calculation was made.

This means that a boid may sometimes never reach any of the points specified to it by the algorithm. In controlling boids one must be aware of this and that the overall behaviour is a result of many small steps. This demonstrates the divergence from classical control where a command is given and completed before another is sent.

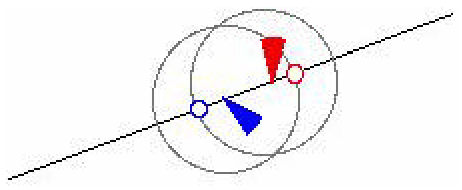


Figure 17: In-line repulsion

The second case pictured above shows two boids that are too close (closer than the separation distance). To satisfy the forming lines algorithm they are told to move away from each other along the line that joins them.

The combination of these two rules under most conditions will cause two boids to oscillate with relation to each other trying to maintain the separation distance. In the first case the boids are told to move towards each other but when they do it is likely that they will overshoot as each is unaware of the movement of the other. A weighting scheme similar to the one used in flocking would damp this oscillation.

B. Rule for Inner Boids

When a boid can detect two neighbours then must move to form a line of three but where does it want to be in this line? The boid has the choice of three positions, between the neighbouring boids or at either end. The different cases are shown below.

If the figure below the lower boid can detect two neighbouring boids. Their average point is further from it than the closer boid. The algorithm says that this boid must move behind its closest neighbour to form a line containing the further neighbour. The target point is marked by a small circle. The larger circles in this illustration represent the separation distance. In all cases the boids seek to maintain this distance.

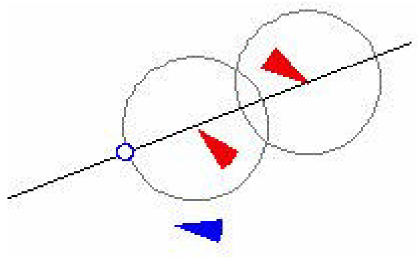


Figure 18: Moving to then end of a line

In the case depicted in the next figure the upper boid can detect two neighbours. Their average point is closer than either of the two boids so it moves to this point, again marked by a small circle. If the other two boids can detect each other then they will follow the previous case and a line will form even quicker.

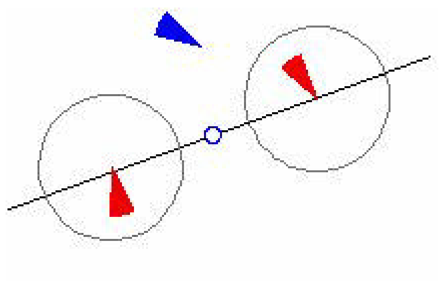


Figure 19: Moving into a line

These rules are quite powerful in practice as not only does the upper boid move to form a line when following this rule but the motion of the other two boids assists in this endeavor. Rather than uncooperative behaviour the usual difficulty is that the boids move too quickly and overshoot their desired position in the line.

For more complex cases where a boid can detect a multitude of neighbours, it ignores all but the closest two and follows the above rule according to their positions. The case of not being able to detect any boids is identical to the previous algorithm and the path taken is straight ahead.

This behaviour is less demanding on the detection capability of the robots than the flocking algorithm presented earlier. For flocking it was necessary to detect the position and direction of travel of neighbouring boids. The forming lines algorithm only needs to be able to detect their positions. This algorithm may serve as a fallback position should the detection of direction prove infeasible.

VI. Discussion of Source Code

A brief study of existing resources showed that by far the preferred mode of research was through swarm animation, typically using the Java&#129; language. Because of existing experience in programming the choice was made to customise a Java&#129; application as a base for studying various swarm models. A list of related sites on the Internet has been included in this report as Appendix 1.

The advantage of Java&#129; as a programming language is its object orientated approach and ability to execute on different platforms without separate compilation. The progress and results of this project have been presenting on the Internet with some positive response. The use of this electronic medium means that the applets presented are available to anyone with Internet access.

The existing resources are primarily demonstrations with only a brief analysis if any of the performance. The decision to create a new application was aimed at obtaining a complete understanding of the process. The applications created have been continuously updated throughout the project and will now perform exceedingly well as a research tool.

This section will discuss in detail the functional sections of the Java&#129; code. An applet can consist of a number of 'class' files which represent modular components of the code and are ideally independent of each other. It is possible therefore to replace a class file with a different version of the same name and execute the applet with no need for recompilation.

The main area of study has been the swarming or flocking behaviour that was demonstrated in a previous chapter. The code is quite short, comprising approximately 10 pages in total (included as Appendix 2 and on disc in text format). It is broken into a number of distinct classes for ease of modification. The diagram below shows the relationship between the various classes that form the Swarm applet:

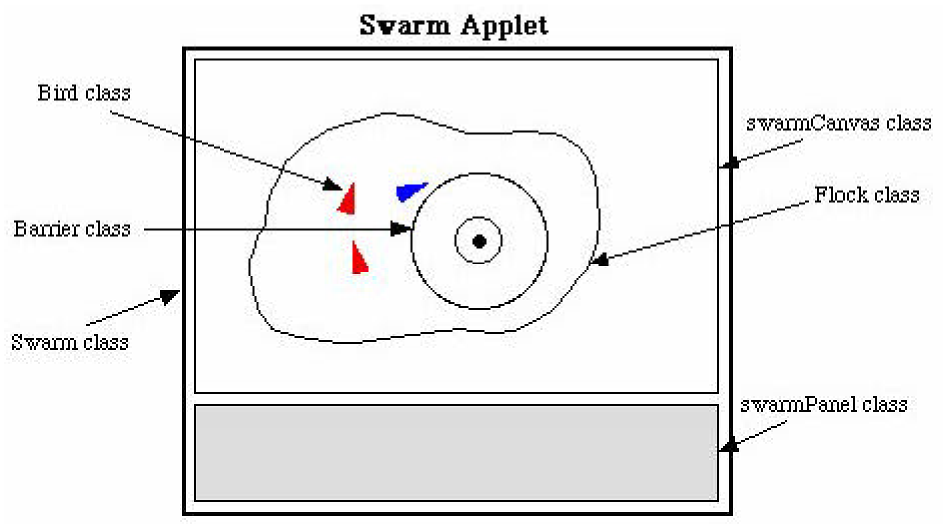


Figure 20: Components of the Swarm applet

A. Swarm.class

This class is responsible for setting up and initialising the applet appearance and contents. The Swarm class also implements the Runnable interface which means that it creates and controls the animation thread.

The operation of this class runs as follows:

1. Create swarmCanvas and swarmPanel objects, adding them to the applet.
2. Start the thread and pass the boundary values to the Bird class.
3. Respond to key press and events passed from the control panel (below).

The Swarm class contains the initial conditions of the applet. The constants defined here are NUMBLUE, NUMRED and MAXPOP for Bird populations; SPEED and TURN for Bird movement parameters; MINDIST and MAXDIST for Bird separation and detection distances respectively.

B. swarmPanel.class

The swarmPanel class contains only one method, designed to initialise and implement a set of controls. The controls appear in the lower portion of the applet and include sliders that allow the user to change the boid populations, and buttons that pause and reset the animation.

C. swarmCanvas.class

This class controls the appearance of the upper portion of the applet where the animation takes place. The animation flicker has been removed through double buffered graphics. This means that each new frame is first drawn off screen and brought to the front seamlessly. There is a slight cost in animation speed but this is outweighed by the improved appearance.

The paint method first blanks the screen with a white background each timestep. A more advanced version could make use of clipping regions to only update areas that have changed but the complexity of this would detract from the purpose of the applet.

This method also draws the Bird parameters, these appear in black at the upper left corner of the applet. The parameters change in response to key inputs as described previously. The painting of the flock elements is achieved through a call to the Flock.display method.

The only other method of the swarmCanvas class allows the user to add a Barrier to the swarmCanvas via a mouse click. To make the implementation of Barriers more realistic the code adds duplicate Barriers outside the region of the canvas to produce a wrap-around effect when they are placed at the perimeter.

D. Flock.class

This is perhaps the most important section of the code as it implements the flocking algorithm and coordinates the Bird objects. In addition to construction, this class includes methods for adding, removing, displaying and directing Birds in the flock.

The flock is implemented as a Vector. This object type is appropriate because it allows objects to be added and removed in any order and is not limited in size. The addBird method simply extends the Vector vBirds by one element of the specified colour. To remove a Bird we have to first locate one of the specified colour and then remove it from the Vector.

An important note here is that the removeBird and move methods of the synchronized type. This means that they cannot both act on the flock at the same time. Without this specified the removeBird function can remove a Bird in the middle of a move operation.

Similar to the swarmCanvas class the display method of the Flock class makes use of the Bird.display method for each member of the flock. The move method takes the same approach but the Flock class must first generate a direction for each Bird to steer towards.

The object oriented nature of this language means that the Flock class need not know how the Bird moves, only that it has to pass a value between 0 and 360 degrees. This value is determined in the generalHeading function which, for each boid in the flock, applies a simple algorithm to come up with the preferred direction.

The flocking algorithm is the most complex section of the code and will be described in a separate section. For each of the applets presented in this report, it is the generalHeading function that changes the behaviour of the boids.

E. Bird.class

The Bird class has a number of static variables, arenaWIDTH, arenaHEIGHT, maxSPEED and maxTURN, these are common to all Birds. The Birds all have four individual properties, these define the position (x, y), direction (Ð) and colour of the Bird.

This figure shows how the Bird parameters are defined. The x- and y- coordinates are taken from the top left corner of the applet. The angle property is measured in a counter-clockwise direction from the horizontal, as shown. The colour property is obviously rendered graphically.

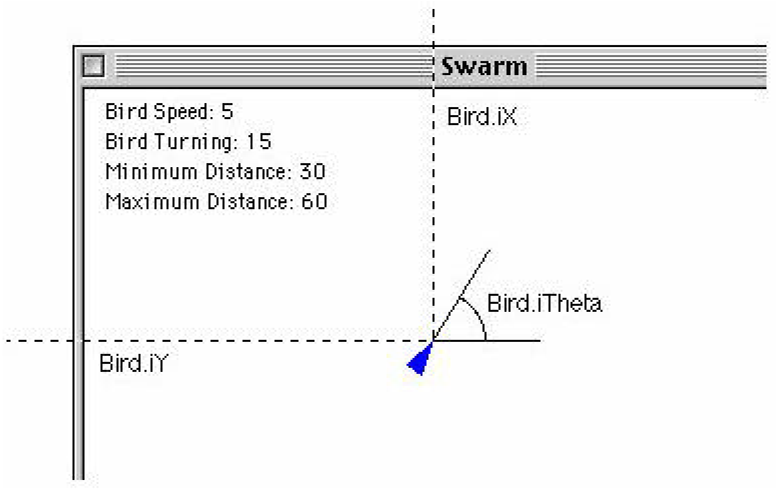


Figure 21: Properties of the Bird class

As described in the preceding section, the Bird.move function takes an angle as its only parameter. When this method is called the Bird decides which direction to turn and changes its direction to the given angle to the extent that this is possible under the maxTURN condition.

The remaining methods in the Bird class are used to extract parameters involving the individual Birds and set the limits of the animation.

F. Barrier.class

The Barrier class is an extension of the Bird class and as such inherits all the properties of that class and a Barrier object can also be referred to as a Bird. The changes from Bird are that we overwrite the constructor, move method and display method.

This means that a barrier is essentially treated as a stationary Bird. Because the barrier colour is black, it repulses all boids that come within the detection range.

All code described in this chapter has been written under Java&#129; version 1.02. The most recent version available is version 1.1.4 which makes some significant changes to the event model. The first step for anyone wanting to work further with the code should be to adopt the new specifications.

VII. Discussion

The limitations of the Swarm applet are that the boids are modeled as point elements. In other words they have no physical size and so cannot obstruct each other. The flocking area is not really realistic as the boids can move off one edge and 'wrap around', coming in the opposite side.

Unfortunately, the mechanics flocking algorithm does not carry across the boundaries of the applet. This means that members of the flock can be lost temporarily when they cross these boundaries. This could be corrected by repelling boids from the boundary or enhancing the code to allow flocking across boundaries.

When using the Applet Viewer or Applet Runner the window can be resized which gives more room for the algorithm to work. Inside a web browser the size of the applet can be restrictive - particularly for the Forming Lines applet. The primary goal of this report is to provide guidance for the creation and programming of real mobile robots. The tools provided with this report will hopefully prove useful in the creation of such robots.

Some conditions that have not been modelled so far include: variations in speed and direction due to uneven terrain, boids that can change speed and collisions when boids come too close. Some kind of subsumptive control would be useful in dealing with the occasional collision. Subsumption means that the boid can temporarily change its behaviour in response to an event.

The most well known robots that use subsumption architecture are the insect-like robots designed by Rodney Brooks. Genghis is a cockroach-like robot the size of a football, built by Rodney Brooks ar the MIT. The Genghis robot has six legs and implements localised control for each of them.

The subsumption architecture divides the control architecture into task achieving modules or behaviours. Each layer forms a complete behaviour able to control the robot. The idea is that the robot starts in a low-level behaviour and when appropriate moves to a higher level. For example, the first behaviour might be to stand still and test the detection system, the second would be to begin moving and the third would be to interact with the environment.

When something goes wrong or a robot gets stuck it can move back to a lower level of behaviour to correct the problem. This could be applied to the flocking algorithm as follows. When the boid comes too close to a neighbour or barrier it could temporarily halve it's speed or go into a more detailed avoidance mode. This behaviour would be completely independent of the normal flocking behaviour.

VIII. Conclusions

A number of behaviours have been described in this report. Each of the above applets presents a complex behaviour generated by a simple algorithm. The first behaviour demonstrated was flocking. The separation property of the flocking algorithm is essential for controlling large groups of mobile robots and preventing collisions. The cohesion and alignment properties serve to coordinate movements in such a way that the boids remain in contact with each other and travel in the same general direction.

The cost saving from having many homogeneous robots could be significant making such technology more prevalent in the future. Potential applications are both scientific and military. The concept of multiple mobile robots can be used for traffic or flow control, foraging and cooperative manipulation.

Traffic control could include both air and road traffic. A collision avoidance system in every car is a basic implementation of parts of the flocking algorithm. If cars are also programmed to maintain the velocity (direction and speed) of their neighbours then that is essentially flocking.

The application of foraging could involve search and rescue teams at disaster sites either above or below ground. Another use could be to collect hazardous materials after a spill or other accident. In the Mars Pathfinder this year we have seen a vehicle that, if produced in numbers, could be used to more effectively explore distant planets.

The tools and information contained in this report should serve as a solid base for future research in this area.

Appendix 1: Bibliography and References

Dilvan de Abreu Moreira PhD, "Agents: A Distributed Client/Server System for Leaf Cell Generation", The University of Kent, Canterbury, UK (1995).

R.A. Brooks, "A Robust Layered control System For a Mobile Robot," IEEE Journal of Robotics and Automation, vol. RA-2, no. 1, March 1986, pp. 14-23.

B.l. Brumitt, A. Stentz, "Dynamic Mission Planning for Multiple Mobile Robots", Robotics Institute Carnegie Mellon University, Pittsburgh (1996).

C.W. Reynolds, "Not Bumping Into Things" in the notes for the SIGGRAPH '88 course Developments in Physically-Based Modeling, pages G1-G13, published by ACM SIGGRAPH (1988).

C.W. Reynolds, "Flocks, Herds and Schools: A Distributed Behavioral Model," Computer Graphics, vol. 21, no. 4, July 1987, pp. 25-34.

I. Stewart, "A Subway Named Turing", Scientific American (Sep. 1994), pp 90-92.

C. Ünsal, J.S. Bay, "Self Organization in Large Populations of Mobile Robots", Bradley Department of Electrical engineering, Virginia Polytechnic Institute and State University, Blacksburg (May, 1993).

"Calculating with DNA", Scientific American (Sep. 1995), pp18-19.

Internet References

[1] The original Boids page by Craig Reynolds:
http://hmt.com/cwr/boids.html

[2] Swarm Java Applet:
http://tqd.advanced.org/2966/Alife/Boids/

[3] Robotics and Machine Intelligence Laboratories Page - Armyant Robots:
http://armyant.ee.vt.edu/

[4] Subsumptive Military Satellite Constellations:
http://www.pcisys.net/~mork/usats.htm

Appendix 2: Source Code

The attached pages contain the following four files:

- Swarm.java (containing also the swarmPanel and swarmCanvas classes)
- Flock.java
- Bird.java
- Barrier.java

The correct method for embedding this applet in an html document is:

```
<APPLET code="Swarm" width="400" height="250">
<PARAM name="numred" value="5">
<PARAM name="numblue" value="5">
<PARAM name="speed" value="5.0">
<PARAM name="turn" value="30">
<PARAM name="min" value="45">
<PARAM name="max" value="60">
</APPLET>
```

The parameters are optional and default values will be inserted if required. The first two parameters control the flock populations, the next two control the boids movement and the last two the separation and detection radii respectively. This is a useful way to change the behaviour without having to recompile the applet.

The code is documented and should be relatively easy to follow for someone with experience in 'C' or Java&#129;. A hard copy is provided here for the flocking algorithm only, because the other applications are essentially identical. For example the Forming Lines applet only changes one method in the Flock class and removes the Barrier class. All of the applets are attached on floppy disc.
