FORAGE ALGORITHM:
The Theatre of Behavioural Experimentation

PREFACE.

Emergent behaviour is that which arises from simple rules followed by individuals but does not involve any central coordination.
Emergent behaviour is where complex global behaviour can arise from the interaction of simple local rules.
One individual on its own doesn't show much promise. Put a community of individuals together and things can happen that can go beyond just the sum of the component parts.

THE QUEST.

The Theatre of Behavioural Experimentation's ultimate quest is to find perfect examples of emergent behaviour.
Emergent behaviour can usually always be explained or understood. There is always a rational explanation as to why it happens. Hindsight is a wonderful thing. What makes it truly fascinating is that more often than not, it can't be predicted

CONTENTS:
These start on the next page.

GENERAL INTRODUCTION:

What is the Forage Algorithm?

This is a computer programme which simulates the activities of ants and specifically those of workers that are responsible for fetching and carrying objects from their natural environment back to the nest. Compared to real ants, the forage algorithm is very basic both in terms of functionality, population count and size of territory. There is, however, plenty of scope for experimentation and to conduct simple observations using this community.
For the sake of clarity the ants in the rest of this explanation will be referred to as the “Nunbrums” to differentiate them from real ants. The term “Nunbrum” as you probably know is derived from the latin “Rubrum Nuncii” which of course, as you know, means the “Red Messengers”.

What can the Nunbrums do?

When this programme was first written in 1992 for a TV Natural History programme called “The Little Creatures who Run the World” (First aired by the BBC on the 28th February 1993) the motivations for the Nunbrum activity, at that time were just two. These were firstly, to track down and find a food object and secondly, to track down and find a nest point on which to deposit the food and to reiterate this until either food objects or nest points ran out.
Since then, two more basic activities have been added which are firstly, to follow a pheromone trail and secondly, to proceed to a Muster Point.
To “follow a pheromone trail” should more accurately be described as “follow a pathway”, as at the time of writing, the Nunbrums do not lay down pheromone at the moment but only follow it.
Proceed to muster point is just a way of gathering up the Nunbrums in one place after finishing their work or if they need a place to wait where we, their masters, can see them.
The TBE offers lots of other commands besides the four motivational activities mentioned and these are dealt with in detail in the “Guide to Forage Task Lines”

What is so interesting about the Nunbrums?

If there was just one individual Nunbrum (ant) in this computer programme for you to experiment with then, quite frankly, you might be better of spending your time on something else.
More than one individual, even just two, makes a huge difference to any simulation you may run. The reason for this is that rules have to be devised in order for a simulation to work convincingly. It wouldn’t be a very clever programme if it allowed the Nunbrum community to run right through each other as if they were all just ghosts and had no apparent material substance. In the real world, this (intersection) is not possible because of the rules of physics.
Rule one, therefor, is that Nunbrums must not run through each other.

Rules to Prevent Collisions

Problems arise when rules have to be implemented.
Rule one, the prevention of intersection looks on the face of it, easy. Don’t let the Nunbrums intersect with each other.
Have you ever walked along a pavement through a crowd of people walking the other way? What do you do to avoid knocking into them? Are you one of those that just plough on in the hope that someone else coming in the opposite direction will move out of your way? Are you one of those that will choose to move sideways to allow others coming in the opposite direction, a direct and undeviating path?
The system implemented in this algorithm is democratised but does allow adjustment of individual dominance with regard to collisions, for the purposes of experimentation.

Collision Avoidance Strategy.

(Stopping Nunbrums knocking into Numbrums)
Long distance collision avoidance is implemented by each Numbrum by taking a slightly different path if it is likely to run into another. This is referred to as “Imminent Collision Avoidance” and comes into effect if an obstacle (like another Nunbrum) is, by default, within 6 units distance AND within 20 degrees of the Front Field of Vision which is the angle measurement of whether the obstacle is in front or not. These parameters which are “Imminent Collision Distance” and “Imminent Collision FOV” can be changed for the purposes of experimentation.
Imminent Collision Avoidance in not essential but improves the flow of motion in most circumstances.
Collision Avoidance at close quarters uses two strategies. The first is to stop moving for a random period of time in the hope that the obstruction will move out of the way. The values of this random parameter, the “Collision Think Time”, can be changed for the purposes of experimentation.
The second is to plot a new pathway which avoids moving forward into the obstruction but finds a path to the side of it.
This is referred to as just “Collision Avoidance” and comes into effect if an obstacle (like another Nunbrum) is, by default, within 3 units distance AND within 50 degrees of the Front Field of Vision. These parameters which are “Collision Distance” and “Collision FOV”, can be changed for the purposes of experimentation.

Rules about Food and Nest Point Ownership

Supposing I asked you to go to a shop, to pick up a specific newspaper and deliver it to me personally. “Nip down to the shop for me will you?”
It doesn’t sound too complicated does it and why would we need rules to govern this simple request?
First, you need to remember what I have asked you to do. “Newspaper” or “Specific Newspaper” are useful terms for you to remember so that when you get to the shop, you know what to ask for.
Second, when you get to the shop, you find the newspaper is not there or it has disappeared. Maybe it has been delivered by someone else. Either way, you cannot now complete the task. Maybe if I had asked you to pick up any newspaper without being specific about which one then you might stand a better chance of completing the task. Maybe if you had told the shop to reserve the specific newspaper until you got there, that would have helped.
Third, if you had managed to pick up the newspaper but someone else insists on taking it from you mid transit (steals it) before you can deliver it, then you cannot now complete the task.
Fourth, you need to remember what I have asked you to do next. Come and find me at home is useful to remember.
Fifth, you manage to pick up the newspaper, successfully carry it to my home but I already have a newspaper that someone else has given me and I don’t want another, then this again spells failure. “I’m sorry but you should have explained to me, that on agreeing to do this task for me, that I am not allowed to ask anyone else to do it.”
You might think that making up rules to allow the efficient delivery of a newspaper is silly but these rules exist in real society already. They are invisible because we take them for granted.
The combinations of things that might go wrong in the fetching and carrying of the newspaper all exist within the forage algorithm which does not take these rules for granted and can be changed and manipulated to suit your own experiments.
The chapter on “Ownership Management” explains how to control the issues discussed above.

FORAGE ALGORITHM:
The Theatre of Behavioural Experimentation

FORAGE ALGORITHM TASKS:

Introduction to the Task Line:

The activities of ants, food objects, nest objects and other participants are controlled by 'Task Lines'. These are instructions in the form of lines of code which govern activity and these are put together in an ordered list to form a sequence. There are some examples of interesting sequences listed later in (Examples of Task Line Sequences) and which you can try out in the demonstration and experimentation window.
Embedded in the Task Line is a Task Command. Three examples of commands are:-
FPT = Follow Pheromone Trail.
FFD = Find Food.
PUF = Pick Up Food.

Each task line comprises of firstly, an identifier (Task Label) and then the task itself including a list of arguments that are relevant to the command.

The Format of the Task Line:

Each task line must end with a semicolon (;).
The Task Label and the Task Command must be separated by a comma (,)
The arguments for the command must be within brackets and separated with commas (,).

```
Task Label String, Task Command (Arg 1, Arg 1 + n);
```

An Example of a Task Line:-

```
task04,FPT( RouteA_, start,  10, task05);
```

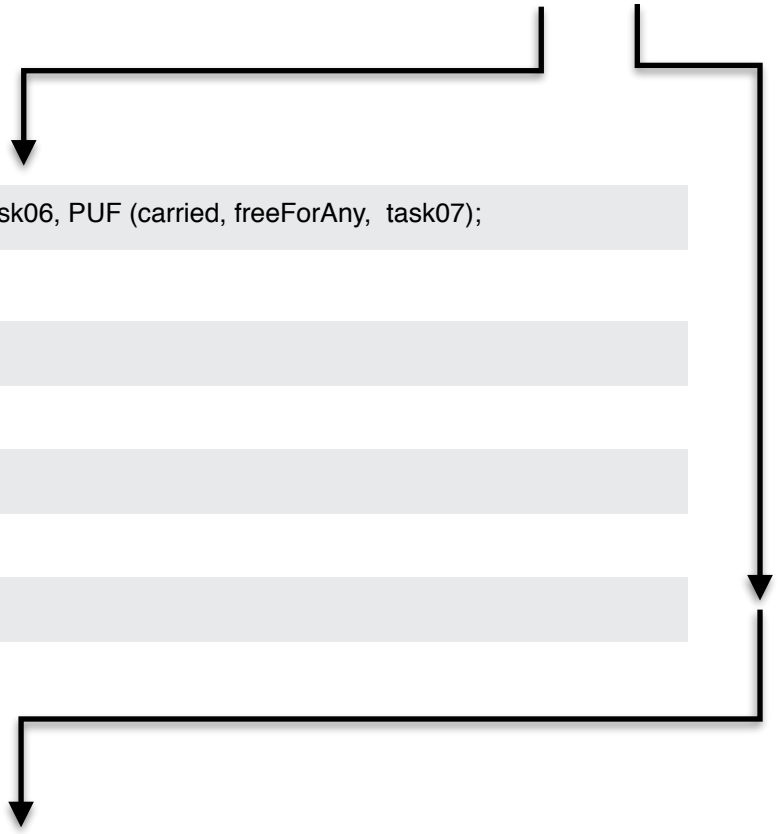
In the example above, the task line is identified by the label 'task04' (the task Label) so that this task can be referred to by other tasks. The task command is defined by three letters and in this example it is FPT (Follow Pheromone Trail) which needs four arguments (four pieces of information) in order to carry out the task. In this example the arguments are 'RouteA_', 'start', 10, 'task05'.
The Task Label can be any word but must not include spaces and should not be enclosed in inverted commas, either single or double.
Examples of Task Labels are:-
task05, followTheTrail, finishHere.

The arguments for the command must be enclosed in plain brackets and separated by commas.

Below is an example of a series of task lines. The Task Labels are used as an identifier so that other task lines can reference to them. Task Lines are not necessarily executed in the order they are listed.

```
task04, FPT( RouteA_, start,  10, task05);
```

```
task05, FFD (Food_GrpA, freeForAny, mine, task06, task10);
```



```
task06, PUF (carried, freeForAny,  task07);
```



```
task10, PUF (carried, freeForAny,  task11);
```

The Fetch and Carry Tasks:

Of the many things that this community of ants can do, one of the most interesting aspects is the fetching and carrying of food objects by the ant community to prearranged destinations called nest points.
Ideally, for every food object there should be a corresponding nest point.
The carrying of food objects to individual nest points is the essence of the Fetch & Carry Tasks and works as follows.

If an ant is not carrying a food object, then it must find and choose one, pick it up, then find and choose a nest point, carry the food object to it and then put the food object down. Go back and do it again

If the community of ants where just one, then this series of tasks would be straight forward. Here is how the tasks are implemented.

1. Find a Food Object (Stop if there are no more food objects available)
2. Pick up the Food Object
3. Find a Nest Point (Stop if there are no more Nest points available)
4. Put down the Food Object
5. Go back to line 1 and do it again.

The commands for these fetch and carry tasks, which are explained later, are as follows:-
FFD = Find Food.
PUF = Pick Up Food.
FNP = Find Nest Point.
PDF = Put Down Food

When there is a community of ants (more than one) then this series of tasks gets much more complicated and rules must be introduced to prevent chaos and the algorithm from failing.
The section on Ownership Management deals with the rules of who owns what and when, by exchanging tokens. There are two types, a search token and a reservation token. When and how these tokens are exchanged is critical to the behaviour of the community.
In the early days the British railways and right up to the 1960's, tokens were used and exchanged to prevent accidents from happening on single line working routes. These single tracks, which are used to run trains both ways on the same line, are a potentially lethal combination. The tokens prevent two trains from sharing the same line and thus prevent accidents.
The fetch and carry tasks use a similar system which keeps the simulation stable.

In the next section, the fetch and carry tasks and their command arguments are explained.

FETCH & CARRY TASK DEFINITIONS and their ARGUMENTS:

TASK: Find Food:

FFD (Arg 1, Arg 2, Arg 3, Arg 4, Arg 5)
Arg 1 = string Food Group Name
Arg 2 = string Search token (a match with Food-Status signifies availability).
Arg 3 = string Reservation token (placed in Food-Status to signify ownership).
Arg 4 = string The next Task Label to go to when this one is completed .
Arg 5 = string The next Task Label to go to when no Food Objects Available.
.

An Example of an FFD:-

FFD (Food_GrpA, freeForAny, Carried, next3, task10)

Find the nearest food object within a group called 'Food_GrpA' and only if the food object's Food-Status reads 'freeForAny'. When found, change the food object Food-Status to 'Carried'. Go to task 'next3' for the next instruction. If no food available then go to task 'task10'.

TASK: Pick Up Food:

PUF (Arg 1, Arg 2, Arg 3)
Arg 1 = string Reservation token (placed in Food-Status to signify ownership).
Arg 2 = string Nest Available token (if applicable) (placed in Nest-Status to signify availability)
Arg 3 = string The next Task Label to go to when this one is completed.

An Example of a PUF:-

PUF (carried, freeForAny, nextLine1)

Pick up the food object and set the Food-Status to 'carried'. Set the Nest-Status, if there is a nest point, to 'freeForAny', then go to task 'nextLine1' for the next instruction.
PUF is usually executed directly after an FFD.

TASK: Find Nest Point:

FNP (Arg 1, Arg 2, Arg 3, Arg 4, Arg 5);
Arg 1 = string Nest Group Name
Arg 2 = string Search token (a match with Nest-Status signifies availability
Arg 3 = string Reservation token (placed in Nest-Status to signify ownership).
Arg 4 = string The next Task Label to go to when this one is completed.
Arg 5 = string The next Task Label to go to when no Nests Available.

An Example of an FNP:-

FNP (Nest_GrpA, freeForDrop, Occupied, task9, task14);

Find the nearest nest object within a group called 'Nest_GrpA' and only if the nest object's Nest-Status says 'freeForDrop'. When found, change the nest object Nest-Status to 'Occupied', then go to task 'task9' for the next instruction. If no nest object available then go to task 'task14'.

TASK: Put Down Food:

PDF (Arg 1, Arg 2, Arg 3);
Arg 1 = string Reservation token (placed in Food-Status to signify ownership or availability).
Arg 2 = string Reservation token (placed in Nest-Status to signify ownership)
Arg 3 = string The next Task Label to go to when this one is completed.

An Example of a PDF:-

PDF (notAvail, occupied, task7);

Put down the food object and set the Food-Status to 'notAvail'. Set the Nest-Status, if there is a nest point, to 'occupied', then go to task 'task7' for the next instruction.
PDF is usually executed directly after an FNP.

In the next section, the issues of ownership and the use of tokens are explained.

OWNERSHIP MANAGEMENT:

The ownership of food objects and nest points by ants has to be managed, in order to remove unstable or chaotic scenarios. If ownership is not managed for instance, a situation will occur where several ants will select the same food object. This conflict would make the algorithm unusable because it would come to a stop. The management therefor, is implemented by the use of exchange of tokens.

A food object has the following variable:-

Food-Status: Indicates the current status (Examples could be “none”, “Carried”,”not_Available”)

A nest point has the following variable:-

Nest-Status: Indicates the current status (Examples could be “none”, “Occupied”,”free_again”)

Token exchange for FFD (Finding FooD):

An ant that is trying to find a food object must match Argument 2 of the FFD task, the Search token, with the food object’s Food-Status. Any word or phrase will do providing there are no spaces. The word “none” however, has special significance and signals that the token in Food-Status has not been changed.

The two examples below show that the Search token does NOT match the status of the food object and therefor the food object is NOT free to find or pick up by this ant.

FFD Argument 2 Search token = “none”	COMPARE with	Food Object Food-Status = “taken”
FFD Argument 2 Search token = “free”	COMPARE with	Food Object Food-Status = “i_am_free”

The two examples below show that the Search token matches the status of the food object and therefor the food object is free to find or to pick up by this ant.

FFD Argument 2 Search token = “none”	COMPARE with	Food Object Food-Status = “none”
FFD Argument 2 Search token = “i_am_free”	COMPARE with	Food Object Food-Status = “i_am_free”

An ant’s first response to an FFD command, is to choose a food object which is free to pick up. Once the chosen food object is selected by using the comparisons like the ones above, then there is an option to reserve it by marking it as such, even before the ant has set off to find it. This is achieved by passing Argument 3 of the FFD task to the selected food object Food-Status.

Reservation at the Start of the FFD task:

The example below shows how an ant can reserve a food object by passing any word loaded in the FFD Argument 3 to the Food-Status. In this example, the token word "taken", is transferred to the Food-Status variable.

Once this has been done, then no other ant will be able to select this food object, unless it's FFD Argument 2 (Search token) happens to be "taken".
The side effect of this FFD reservation is that the ant will not be allowed to choose any other food object in the meanwhile, until it has disposed of the one which it has reserved. (It is not free to change it's choice)

No Reservation at the Start of the FFD task:

The example below shows how to avoid a reservation by passing specifically, the word "none" in the FFD Argument 3 to the Food-Status of the food object.

This has the same effect as not passing the token at all and the food object remains free for selection by other ants providing their FFD Argument 2 (Search token) is "none".
The side effect of this FFD missing reservation is that the ant does not need to keep to it's original selection but is free to choose alternative food objects. Until the selected food object is actually picked up, it remains available to all other ants.

Reservation at the Start of the PUF task:

When an ant is sufficiently near and aligned with it's selected food object then it can pick it up by using the PUF (Pick Up Food) task. This task should only be used after an FFD task but not necessarily directly after.
At this point, the food object Food-Status is loaded with argument 1 (Reservation token) of the PUF task.

The example below shows the passing of PUF Argument 1 to the food object Food-Status. If the previous state of the Food-Status was "none" then it is at this point that this food object now becomes unavailable and of no interest to other ants.

If, however, the token of "none" is passed to Food-Status at this point then technically the food object is still of interest to other ants.

Reservation at the Start of the PUF task:

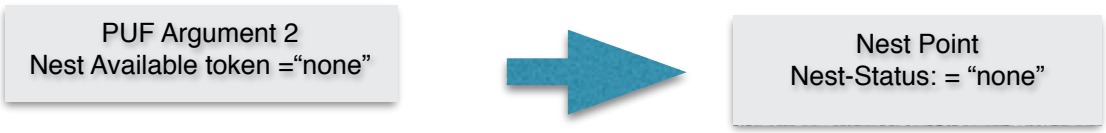
By keeping the food object's Food-Status token as "none" by loading the PUF Argument 1 with "none" will keep other ants interested and they may even chase the food object while in the jaws of another. They will not, however, be able to pick it up (PUF) or snatch from the jaws of another.

Changing Nest Status at the Start of the PUF task:

A food object, that is about to be picked up, may have been previously put down on a nest point. If this is the case, then an opportunity to change the nest point Nest-Status can be implemented at this point. This gives the opportunity for an ant picking up the food object, to make the nest point available for further use by other ants.

A nest point has the following variable:-
Nest-Status: Indicates the current status (Examples could be “none”, “Occupied”)

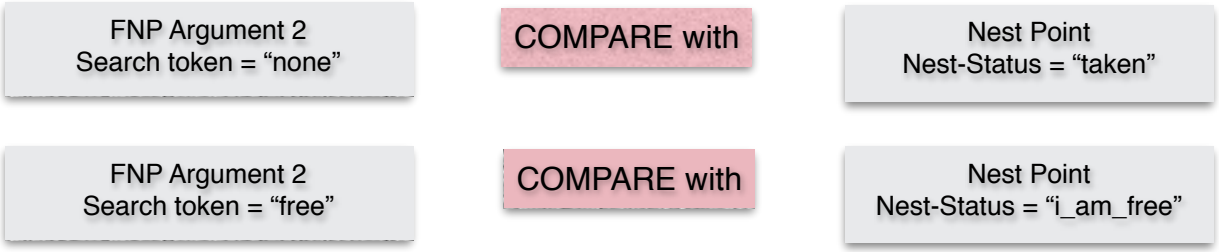
In this case the following change is made to the nest point Nest-Status indicating that the nest point is available again and can allow another food object to be put down at this point.



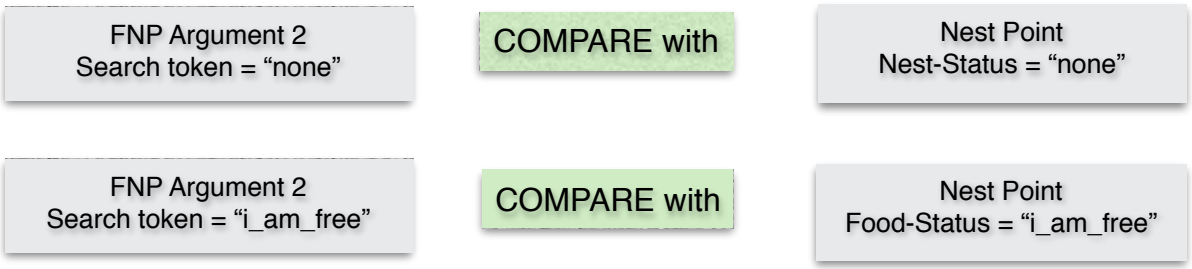
Token exchange for FNP (Finding Nest Point):

An ant that is trying to find a nest point, so that the food object its carrying can be placed their, must match Argument 2 of the FNP task, the Search token, with the nest point’s Nest-Status. Any word or phrase will do providing there are no spaces. The word “none” however, has special significance and signals that the token in Nest-Status has not been changed.

Like the FFD tokens the two examples below show that the Search token does NOT match the status of the nest point and therefor the nest point is NOT free to receive a food object. Usually it is because there is a food object already allocated to this point.



Like the FFD tokens the two examples below show that the Search token matches the status of the nest point and therefor the nest point is free to receive a deposit of a food object by this ant.



Similar to the FFD task an ant's first response to an FNP command, is to choose a nest point which is free to put down on. Once the chosen nest points selected by using the comparisons like the ones above, then there is an option to reserve it by marking it as such, even before the ant has set off to find it. This is achieved by passing Argument 3 of the FNP task to the selected nest point Nest-Status.

Reservation at the Start of the FNP task:

The example below shows how an ant can reserve a nest point by passing any word loaded in the FNP Argument 3 to the Nest-Status. In this example, the token word "occupied", is transferred to the Nest-Status variable.

Once this has been done, then no other ant will be able to select this nest point, unless it's FNP Argument 2 (Search token) happens to be "occupied".
The side effect of this FNP reservation is that the ant will not be allowed to choose any other nest point in the meanwhile, until it has used the one which it has reserved.

No Reservation at the Start of the FNP task:

The example below shows how to avoid a reservation by passing specifically, the word "none" in the FNP Argument 3 to the Nest-Status of the nest point.

This has the same effect as not passing the token at all and the next point remains free for selection by other ants providing their FNP Argument 2 (Search token) is "none".
The side effect of this FNP missing reservation is that the ant does not need to keep to it's original selection but is free to choose alternative nest points. Until the selected nest point is actually used to receive a food object then it remains available to all other ants.

Changing the Nest Status at the Start of the PDF task:

When an ant is sufficiently near and aligned with it's selected nest point then it can put it down using the PDF (Put Food Down) task. This task should only be used after a FND task but not necessarily directly after.
At this point, the food object Nest-Status is loaded with argument 2 (Reservation token) of the PDF task.

The example below shows the passing of PDF Argument 2 to the food object Nest-Status. If the previous state of the Nest-Status was "none" then it is at this point that this nest point now becomes unavailable and of no interest to other ants.

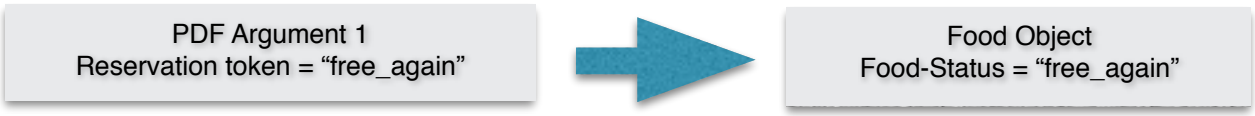
If, however, the token of "none" is passed to Nest-Status at this point then technically the nest point is still of interest to other ants.

Not changing the Nest Status at the Start of the PDF task:

By keeping the food object's Nest-Status token as "none" by loading the PDF Argument 2 with "none" will allow other ants to deposit their food objects there as well.

Changing the Food Status at the Start of the PDF task:

At the point when the food object is about to be put down then there is a last opportunity for it's Food-Status to be changed. What ever a Reservation token is loaded into the PDF Argument 1 is transferred to the food object's Food-Status.



Summery of Ownership Management:

The complete fetch & carry tasks are carried out in the order listed below
(Intervening tasks can sometimes be included).

Find Food Where? Find Nest Point Where?

FFD Task Label (MyFoodGroup,Arg2,Arg3,Arg4.Arg5)
PUF Task Label (Arg1,Arg2,Arg3)
FNP Task Label (MyNestGroup,Arg2,Arg3,Arg4.Arg5)
PDF Task Label (Arg1,Arg2,Arg3)

FFD: Can find a food objects in the group 'MyFoodGroup'.
FNP: Can find a nest points in the group 'MyNestGroup'.

The Search Tokens:-

FFD Task Label (Arg1,Available,Arg3,Arg4.Arg5)
PUF Task Label (Arg1,Arg2,Arg3)
FNP Task Label (Arg1,none,Arg3,Arg4.Arg5)
PDF Task Label (Arg1,Arg2,Arg3)

FFD: I can find and pick up any food object that has a Food-Status of 'Available'.
FNP: I can find and use any nest point that has a Nest-Status of 'none'.
Please not: All food Food-Status and all nest Nest-Status are preloaded with 'none'.

The Food Object Reservation Tokens:-

FFD Task Label (Arg1,Arg2,MINE,Arg4.Arg5)
PUF Task Label (mineNow,Arg2,Arg3)
FNP Task Label (Arg1,Arg2,Arg3,Arg4.Arg5)
PDF Task Label (atNest,Arg2,Arg3)

FFD: I have already marked the %foodStatus as 'MINE' at the very outset. Unless any other ant is using the search token 'MINE' then this food object is mine to find and pick up.
PUF: On picking up the food object I am marking the %foodStatus with 'mineNow'. Unless any other ant is using the search token 'mineNow' then this food object is of no interest to any other ant.
PDF: When I put down the food object I will be marking the %foodStatus with 'atNest'. Unless any other ant is using the search token 'atNest' then this food object is of no interest to any other ant.

The Nest Point Reservation Tokens:-

FFD Task Label (Arg1,Arg2,Arg3,Arg4.Arg5)
PUF Task Label (Arg1,Arg2,Arg3)
FNP Task Label (Arg1,Arg2,myNest,Arg4.Arg5)
PDF Task Label (Arg1,occupied,Arg3)

FNP: I have already marked the nestStatus as 'myNest at the very outset. Unless any other ant is using the search token 'myNest' then this nest object is mine to find and use.
PDF: When I put down the food object I will be marking the nest Nest-Status with 'occupied'. Unless any other ant is using the search token 'occupied' then this nest point is of no interest to any other ant.

ADDITIONAL TASK DEFINITIONS and their ARGUMENTS:

TASK: Follow Pheromone Trail:

FPT (Arg 1, Arg 2, Arg 3, Arg 4);
Arg 1 = string PT Member Prefix
Arg 2 = string 'Start' or 'Nearest' ('Start' = at first Ph marker. 'Nearest' at nearest Ph marker)
Arg 3 = int Trail length
Arg 4 = string The next Task Label to go to when this one is completed.

An Example of a FPT:-

FPT (RouteA_, Start, 10, stopTask);

Follow a pheromone trail by moving to the first pheromone marker called 'RouteA_1' then follow the trail by moving to the next marker called 'RouteA_2'. Continue to visit each marker until the final one(marker 10) is reached then go to task 'stopTask' for the next instruction.

TASK: Wait For Food:

WFF (Arg 1, Arg 2, Arg 3, Arg 4);
Arg 1 = string Food Group Name
Arg 2 = string Search token (a match with Food-Status signifies availability).
Arg 3 = int Stop waiting when this number of Food Objects become available.
Arg 4 = string The next Task Label to go to when the waiting has finished.

An Example of a WFF:-

WFF (Food_GrpA, freeForAny, 3, nextTask);

Wait on the spot until at least three Food Objects become available within the food group "Food_GrpA" by comparing argument 2, "freeForAny" with the Food-Status variable. When at least three Food Objects become available go to task 'nextTask' for the next instruction.

TASK: Go to Muster Point:

GMP (Arg 1);
Arg 1 = string The next Task Label to go to when this one is completed .

An Example of a GMP:-

GMP (nextTask);

Go to the personal muster point and proceed to 'nextTask' for the next instruction.

TASK: Comment:

CMT (Arg 1, Arg 2);
Arg 1 = string The Comment
Arg 2 = string The next Task Label to go to when this one is completed.

An Example of a CMT:-

CMT (This is an example of a comment,Task6);

The programmer can leave a note, reminder or comment.
The next task ID is executed immediately. This task has no effect on the ants or other objects.

TASK: Ant Allocation Fork:

AAF (Arg 1, Arg 2, Arg 3, Arg 4, Arg 5, Arg 6);
Arg 1 = string Name of first ant.
Arg 2 = string Name of second ant OR 'none'.
Arg 3 = string Name of third ant OR 'none'.
Arg 4 = string Name of fourth ant OR 'none'.
Arg 5 = string The label of Task Label to go to for the next instruction if you are NOT listed in previous arguments.
Arg 6 = string The label of Task Label to go to for the next instruction if you ARE listed in previous arguments.

An Example of an AAF:-

AAF (Ant_1, Ant_2, Ant_3,none, Task6, Task17);

Instruct Ant_1, Ant_2 and Ant_3 (up to four) to get their instructions NOT from the beginning of task line list, but instead, start from task Label 'Task17'. Ants that are not mentioned in the argument list will go to to 'Task6'.
If the split instruction is to less than four ants, then put 'none' in the unwanted arguments.
If the split instruction is for more than four ants then use a second AAF.

TASK: Steal If Possible:

SIP (Arg 1, Arg 2, Arg 3, Arg 4, Arg 5);
Arg 1 = string Search token (a match with Food-Status signifies availability).
Arg 2 = string Reservation token (placed in Food-Status to signify ownership).
Arg 3 = string The next Task Label to go to if nothing stolen (usually an PUF task).
Arg 4 = string The next Task Label to go to if steal successful (usually an FNP task).
Arg 5 = string The next Task Label to go to if stolen from (usually an FFD task)

An Example of a SIP:-

SIP (none, mine, Ant_3,Task6, Task8, Task1);

Steal if possible a food object which is being carried by another but only if the food object's Food-Status reads 'none'. When stolen, change the food object Food-Status to 'mine'. Go to task 'Task6' for the next instruction if nothing stolen (usually an PUF task). If a steal has been successful then go to 'Task8' for the next instruction (usually an FNP task) and then instruct the stolen from (victim) to go to 'Task1' for their next instruction (usually an FFD task).
The SIP Task Line must only be placed between FFD and PUF

TASK: Wait For a While:

WFW (Arg 1, Arg 2);
Arg 1 = int The number of seconds of waiting time.
Arg 2 = string The next Task Label to go to when the waiting has finished.

An Example of a FPT:-

WFW (20.0, nextTask);

Wait on the spot for 20 seconds than go to task 'nextTask' for the next instruction.

EXAMPLES OF SEQUENCES

All the sequences below are available to try and can be automatically loaded into the Demonstration and Experimentation Window with one click.

What is going on?:

25 Task Lines

This is dealt with later in this section.

Pheromone Trail 1: An example of a recycling Trail

A single Task Line:

```
task08, FPT(Ph_TA,Start,12,task08);
```

Follow the Pheromone Trail defined by the Trail title 'Ph_TA' starting at the beginning and visiting all 12 markers in order.
When finished, go back to the beginning and do this task again by going to ‘task08’.

You can see the trail by clicking on the on-screen 'reveal PHT A' button which reveals it.

Pheromone Trail 2: An example of two connected Trails.

Two Task Lines:

```
task08, FPT(Ph_TC,Nearest,10,task09);
task09, FPT(Ph_TC,Start,12,task09);
```

Follow the Pheromone Trail defined by the Trail title 'Ph_TC' starting at the nearest marker and visit all markers until number 10 is reached then go to ‘task09’.
Follow the Pheromone Trail defined by the Trail title 'Ph_TC' starting at the beginning and visit all markers until number 12 is reached then repeat this by going back to ‘task09’.

You can see the trail by clicking on the on-screen 'reveal PHT C' button which reveals the trail.

Finding Food 1: Using the Reservation token

A single Task Line:

```
task01, FFD(leaf_A_grp,none,mine,task01,task01);
```

The ants insist on KEEPING THEIR FIRST CHOICE of food object.

Find Food (the nearest food object) in the group parent named 'leaf_A_grp' whose status matches the search token 'none'. Reserve the nearest food object at the beginning of this task by placing the reserve token 'mine' in the food object's status.
When finished, go back to ‘task01’,the beginning of this task

Press the on-screen 'Go To Drag View' button, drag the leaves (food objects) around with your cursor and see how the ants react.

Finding Food 2: Using no Reservation

A single Task Line:

```
task01, FFD(leaf_A_grp,none,none,task01,task01);
```

The ants will TAKE ANY food object and WILL CHANGE THEIR MIND.

Find Food (the nearest food object) in the group parent named 'leaf_A_grp' whose status matches the search token 'none'. Do not reserve the nearest food object at the beginning of this task by placing the reserve token 'none' in the food object's status.
When finished, go back to the beginning of this task

Press the on-screen 'Go To Drag View' button, drag the leaves (food objects) around with your cursor and see how the ants react. The ants don't care about keeping their original choice of food object.

Fetch and Carry 1: The original function of the algorithm.

A Five Line Sequence:

```
task01, FFD(leaf_A_grp,none,mine,task02,task05);
task02, PUF(mine,nest_free,task03);
task03, FNP(NestA_grp, none,mine,task04,task05);
task04, PDF(at_nestA,filled,task01);
task05, GMP(task05);
```

A Complete Fetch and Carry Sequence:
Find Food (the nearest food object) in the group parent named 'leaf_A_grp' whose status matches the search token 'none'. Reserve the nearest food object at the beginning of this task by placing the reserve token 'mine' in the food object's status. If there is no food left to find then got to task labelled 'task05'.
When this task is finished go to 'task02'.
Pick Up the Food object and place in the reserve token 'mine' in the food object's status. If the food object had been associated with a nest position then place in the reserve token 'nest_free' to the nest position status.
When this task is finished go to 'task03'.
Find Nest Position (the nearest nest position) in the group parent named 'NestA_grp' whose status matches the search token 'none'. Reserve the nearest nest position at the beginning of this task by placing the reserve token 'mine' in the nest position's status. If there is no nest position left to find then got to task labelled 'task05'.
When this task is finished go to 'task04'.
Put Down the Food object and place in the reserve token 'at_nestA' in the food object's status. Place in the reserve token 'filled' in the nest position status.
When this task is finished go to 'task01' (back to the beginning). The last task is 'task05' which is Go to Muster Point and repeat this task for ever.
You can see the Nest group by clicking on the on-screen 'reveal Nests' button which reveals ALL nest points.

Fetch and Carry 2: Using no reservations on Nest Points
A Five Line Sequence:

```
task01, FFD(leaf_A_grp,none,none,task02,task05);
task02, PUF(mine,nest_free,task03);
task03, FNP(NestA_grp, none,none,task04,task05);
task04, PDF(at_nestA,none,task01);
task05, GMP(task05);
```

Another version of the Fetch and Carry Sequence:
This is the same as the last fetch and carry sequence but unlike the last one where reservation tokens were loaded at the outset of the 'Find Food' (FFD) task and at the point when the food object was picked up (PUF), no reservations are made in this sequence including ones for the nest points.
This, in theory, means that any food object is available until it is actually picked up and any nest point available throughout the sequence.
Like the last sequence, it ends on Go to Muster Point. You can see the muster points by clicking on the on-screen 'reveal MUS P' button which reveals the points. Muster points (and every ant has his/her own individual one) also demands that the ants are correctly orientated.

Other Task Definitions !: Go to Muster Point & Wait For a While
A Four Line Sequence:

```
task00, FPT(Ph_TA,Nearest,12,task01);
task01, FPT(Ph_TA,Start,12,task02);
task02, GMP(task03);
task03, WFW(10,task00);
```

This demonstrates the use of 'Wait For a While' coupled with 'Go to Muster Point'.
Follow the Pheromone Trail defined by the Trail title 'Ph_TA' starting at the nearest marker and visit all 12 markers in turn then go to 'task01'.
Follow the Pheromone Trail defined by the Trail title 'Ph_TA' starting at the beginning and visit all markers until number 12 is reached then go to 'task02'.
Proceed to your Muster Point then go to 'task03'.
Wait here for 10 seconds then go to back to 'task00'.

Other Task Definitions 2: Ant Allocation Fork & Wait For Food.
A twelve Line Sequence:

```
task00, AAF(ant_2,ant_4,ant_6,ant_8,task01,task06);

task01, FFD(leaf_A_grp,none,mine,task02,task05);
task02, PUF(mine,none,task03);
task03, FNP(NestA_grp, none,done,task04,task05);
task04, PDF(At_NestA,filled,task01);
task05, GMP(task05);

task06, FNP(NestB_grp,none,none,task07,task05);
task07, WFF(leaf_A_grp,At_NestA,8,task08);
task08, FFD(leaf_A_grp,At_NestA,mine,task09,task05);
task09, PUF(mine,done,task10);
task10, FNP(NestC_grp, none,none,task11,task05);
task11, PDF(At_NestC,filled,task08);
```

This demonstrates the use of 'Ant Allocation Fork' coupled with 'Wait For Food'.
The first line of this sequences instructs ant_2,ant_4,ant_6 and ant_8 to go to the task line labelled with 'task06'. The rest of the ants which are not mentioned go to 'task01'. This effectively splits a team of eight ants into two teams of four. Each team, therefor, has a different sequence to perform.
Ant_1, ant_3, ant_5 and ant_7 begin by transporting food objects in 'leaf_A_grp' over to 'NestA_grp'. When they reach each nest point they mark the food status with 'At_nestA'. When they have finished the whole of this task they go to 'task05' Go to Muster Point.
The second team of ant_2, ant_4, ant_6 and ant_8 start by visiting 'NestB_grp'. They wait here until there are 8 food objects with the availability token of 'At_nestA" (left by the other team). Once 8 food objects are available they proceed to pick up and carry the food objects back to where they came from which is 'NestC_grp'. When they too, have finished the whole of this task they go to 'task05' Go to Muster Point.

An Example of a Parameter Line: How to change ant parameters.
Load Ant Speed:
A two Line Sequence:

```
task00, LAS (ant_1, none, none, none, 0.15, 6.0, task01);
task01, FPT(Ph_TA,Start,12,task01);
```

This demonstrates the use of 'Load Ant Speed' which transfers the value 0.15 to the distance increment which will slow ant_1 down by a third compared with the other ants. It also transfers the value 6.0 to the bearing increment which affects how quickly ant_1 will turn. 'task01 is implemented next which is an endless Pheromone Trail. See how frustrated all other ants are when trying to overtake ant_1 (Parameter Line Commands are coved in the next chapter).

An Example of a Monitor Line: How to monitor ant variables.

Monitor RFP, RCS, RIC 1:

A eight Line Sequence:

```
task00, RFP (all,task0a);
task0a, RCS (ant_1,task0b);
task0b, RIC (ant_2,task01);
task01, FFD(leaf_A_grp,none,mine,task02,task05);
task02, PUF(mine,done,task03);
task03, FNP(NestA_grp, none,done,task04,task05);
task04, PDF(At_Nest,filled,task00);
task05, GMP(task00);
```

This demonstrates the use of RFP (Read Food Placed) which monitors all ants. The RCS (Read Collisions encountered) monitors 'ant_1' and the RIC (Read Imminent Collisions) monitors ant_2.
The placement of these three Monitor Lines will yield constant monitoring throughout the sequence.

If just a final readout is needed at the end of the sequence then they can placed towards the end like in the example below:-

Monitor RFP, RCS, RIC 2:

```
task01, FFD(leaf_A_grp,none,mine,task02,task00);
task02, PUF(mine,done,task03);
task03, FNP(NestA_grp, none,done,task04,task00);
task04, PDF(At_Nest,filled,task01);
task00, RFP (all,task0a);
task0a, RCS (ant_1,task0b);
task0b, RIC (ant_2,task05);
task05, GMP(task05);
```

An Example of a Fork Line: How to react to variables.

If Greater Than:

A severn Line Sequence:

```
task01, FFD(leaf_A_grp,none,mine,task02,task05);
task02, PUF(mine,done,task03);
task03, FNP(NestA_grp, none,done,task04,task05);
task04, PDF(At_Nest,filled,task0A);
task0A, IF> (ant_1,RFP, 3,task01,task06);
task05, GMP(task05);
task06, FPT(Ph_TA,Start,12,task06);
```

When ant_1 has put down just three food objects, it decides to have change and follow the 'Ph_TA' pheromone trail.

F and C Pandemonium:

A sequence of fetch and carry with absolutely no reservations.
A five Line Sequence:

```
task01, FFD(leaf_A_grp,none,none,task02,task05);
task02, PUF(none,done,task03);
task03, FNP(NestA_grp, none,done,task04,task05);
task04, PDF(none,none,task01);
task05, GMP(task05);
```

Stealing from each other:

SOTMT:

The Supervision of Ownership Transference Mid Transit. (Supervised Stealing).
A severn Line Sequence:
The SIP (Steal If Possible) has been inserted in-between FFD and the PUF. It cannot be used in any other combination.
The ants can be very quick when stealing form each other so monitor Lines have been inserted to read back the number of steals, both stolen (the stealer) and stolen from (the victim).

```
task01, FFD(leaf_A_grp,none,none,task02,task06);
task02, SIP(none,mine,task03,task04,task01);
task03, PUF(none,nest_free,task04);
task04, FNP(NestA_grp, none,mine,task05,task06);
task05, PDF(at_nest,filled,task01);
task06, RSL (all,task07);
task07, RLS (all,task08);
task08, GMP(task08);
```

What is going on?, the Task Lines:

An explanation of the first demonstration.
A twenty five Line Sequence.

Can you guess what is going by either looking at the Lines in the sequence or at the animation?

```
task00, AAF(ant_1,ant_3,ant_5,ant_7,task0b,task09);
task0b, AAF(ant_9,ant_11,ant_13,ant_15,task01,task09);

task01, CMT(Ants 1 to 4 start here,task03);
task03, FFD(leaf_A_grp,none,reserved,task04,task71);
task04, PUF(reserved,none,task05);
task05, FNP(NestA_grp, none,picked,task06,task71);
task06, PDF(At_NestA,filled,task01);

task71, FFD(leaf_A_grp,At_NestB,reserved,task72,task90);
task72, PUF(reserved,none,task73);
task73, FNP(NestD_grp, none,picked,task74,task90);
task74, PDF(At_NestD,filled,task71);
task75, FPT (Ph_TA, Start, 12, task75);
task90, FNP(NestE_grp, none,none,task90,task90);

task09, CMT(Ants 5 to 8 start here,task10);
task10, WFF(leaf_A_grp,At_NestA,8,task11);
task11, FFD(leaf_A_grp,At_NestA,reserved,task12,task81);
task12, PUF(reserved,none,task13);
task13, FNP(NestB_grp, none,picked,task14,task81);
task14, PDF(At_NestB,filled,task11);

task80, FPT (Ph_TA, Start, 12, task81);
task81, FFD(leaf_A_grp,At_NestD,reserved,task82,task90);
task82, PUF(reserved,none,task83);
task83, FNP(NestC_grp, none,picked,task84,task90);
task84, PDF(none,filled,task11);
task85, FPT (Ph_TA, Start, 12, task85);
```

PARAMETER LINES and their ARGUMENTS:

Ant Parameter Lines feed directly into the parameters that control ant behaviour. A typical parameter may be the speed of the ant, whether it walks quickly or slowly.
Parameter Lines look like Task Lines but unlike them, there purpose is to change the values of behaviour variables.

The Format of Ant Parameter Line:

The line starts with the parameter label followed by the name of the command separated by a comma (,), followed by a list of arguments separated by commas (,) and encased in plain brackets.
Each parameter line must end with a semicolon (;).
The number of arguments can change depending the command but if the command is associated withs ants then the first four arguments are always the name of an ant and the last argument is always the name of the next parameter or Task Line command to be executed after this one.
If it is necessary to issue Parameter Commands to more than four ants then use additional Parameter Lines.

```
Param Label String, Command  (Arg 1, Arg 1 + N, Label of Next Line);
```

In this example of an Ant Parameter Line is 'LAS' (Load Ant Speed) the speed that an ant walks). It refers to Ant_1, Ant_2 and Ant_3 and passes two floating point arguments (required for AntSpeed). The last argument is the name of the next parameter or Task Line command.

```
thisFirst, LAS  (Ant_1, Ant_2, Ant_3, none, 0.45, 5.0, nextParam);
```

This example is refers to only one ant.

```
thisFirst, LAS  (Ant_1, none, none, none, 0.15, 3.0, nextParam);
```

This example is refers to seven ants.

```
thisFirst, LAS  (Ant_1, Ant_2, Ant_3, Ant_4, 0.15, 3.0, nextParam);
nextParam, LAS  (Ant_5, Ant_6, Ant_7, none, 0.15, 3.0, next23);
```

Parameter Line can be placed any where in the Task Line Sequence

Parameter: Load Ant Speed

LAS (Arg 1, Arg 2, Arg 3, Arg 4, Arg 5, Arg 6, Arg 7);
Arg 1 = string Name of first ant or ‘all’.
Arg 2 = string Name of second ant OR ‘none’.
Arg 3 = string Name of third ant OR ‘none’.
Arg 4 = string Name of fourth ant OR ‘none’.
Arg 5 = float Distance Increment.
Arg 6 = float Bearing Increment.
Arg 7 = string The label of next Task or Parameter Label.

An Example of an LAS:-

LAS (ant_1, ant_2, none, none, 0.15, 3.0, task14);

Load the Distance Increments with 0.15 and the Bearing Increments with 3.0 of ant_1 and ant_2.
Then go to task ‘task14’.

The Distance Increment is the distance travelled by the ant every frame.
The Bearing Increment is the angle of turn by the ant every frame.

Parameter: Load Think Time

LTT(Arg 1, Arg 2, Arg 3, Arg 4, Arg 5, Arg 6, Arg 7);
Arg 1 = string Name of first ant or ‘all’.
Arg 2 = string Name of second ant OR ‘none’.
Arg 3 = string Name of third ant OR ‘none’.
Arg 4 = string Name of fourth ant OR ‘none’.
Arg 5 = float Minimum Time to think about what to do in seconds.
Arg 6 = float Maximum Time to think about what to do in seconds.

An Example of an LTT:-

LTT (ant_1, ant_2, none, none, 0.1, 0.7, task14);

Load the Minimum Think Time with 0.1 and the Maximum Think Time with 0.7 of ant_1 and ant_2.
Then go to task ‘task14’.

The Minimum Think Time is the smallest amount of time the ant takes to make a decision.
The Maximum Think Time is the largest amount of time time the ant takes to make a decision.
The Actual Think Time is any random value between Minimum and Maximum.

Parameter: Load CollisionThink

LCT(Arg 1, Arg 2, Arg 3, Arg 4, Arg 5, Arg 6, Arg 7);
Arg 1 = string Name of first ant or 'all'.
Arg 2 = string Name of second ant OR 'none'.
Arg 3 = string Name of third ant OR 'none'.
Arg 4 = string Name of fourth ant OR 'none'.
Arg 5 = float Minimum Time to think about what to do when in collision in seconds.
Arg 6 = float Maximum Time to think about what to do when in collision in seconds.

An Example of an LCT:-

LCT (ant_1, none, none, none, 0.2, 0.6, task16);

Load the Minimum Collision Think Time with 0.2 and the Maximum Collision Think Time with 0.6 of ant_1. Then go to task 'task16'.

The Minimum Collision Think Time is the smallest amount of time the ant takes to make a decision about a collision.
The Maximum Think Collision Time is the largest amount of time time the ant takes to make a decision about a collision.
The Actual Collision Think Time is any random value between Minimum and Maximum.

Parameter: Load Collision Parameters

LCP(Arg 1, Arg 2, Arg 3, Arg 4, Arg 5, Arg 6, Arg 7);
Arg 1 = string Name of first ant or 'all'.
Arg 2 = string Name of second ant OR 'none'.
Arg 3 = string Name of third ant OR 'none'.
Arg 4 = string Name of fourth ant OR 'none'.
Arg 5 = float Collision Radius (Distance).
Arg 6 = float Collision Field Of View.

An Example of an LCP:-

LCP (ant_1, ant_2, ant_3, ant_4, 3.0, 50.0, task16);

Load the Collision Radius with 3.0, which, when compared with a distance away from an object that is equal or less than this distance, is then considered to be an obstruction.
Load the Collision Field Of View with 50.0, which is the angle either side of forward direction in which objects are considered as potential obstructions.
A Collision FOV gives a total forward coverage of 100 degrees.
Do this for ant_1,ant_2,ant_3 and ant_4

The default Collision Radius is 3.0 and the default Collision Field Of View is 50 degrees.

Parameter: Load Imminent collision Parameters

LIP(Arg 1, Arg 2, Arg 3, Arg 4, Arg 5, Arg 6, Arg 7);
Arg 1 = string Name of first ant or 'all'.
Arg 2 = string Name of second ant OR 'none'.
Arg 3 = string Name of third ant OR 'none'.
Arg 4 = string Name of fourth ant OR 'none'.
Arg 5 = float Imminent Collision Radius (Distance).
Arg 6 = float Imminent Collision Field Of View.

An Example of an LIP:-

LIP (all, none, none, none, 6.0, 20.0, task16);

Load the Imminent Collision Radius with 6.0, which, when compared with a distance away from an object that is equal or less than this distance, is then considered to be an obstruction.
Load the Imminent Collision Field Of View with 20.0, which is the angle either side of forward direction in which objects are considered as potential obstructions.
A Collision FOV gives a total forward coverage of 40 degrees.
Do this for all ants

The default Collision Radius is 6.0 and the default Collision Field Of View is 20 degrees.
Imminent Collisions help with the smooth running of the algorithm. They help to predict and avoid oncoming collisions.

Parameter: Increment General purpose Counter

IGC (Arg 1, Arg 2);
Arg 1 = string Name of ant or 'all'.
Arg 2 = string The next Task Label to go to when this one is completed.

An Example of an IGC:-

IGC (ant_2, task18);

Increment ant_2's general purpose counter (counter = counter + 1) then go to 'task18'.

MONITORING LINES and their ARGUMENTS

Monitor: Read Food objects Placed

RFP (Arg 1, Arg 2);
Arg 1 = string Name of ant or ‘all’.
Arg 2 = string The next Task Label to go to when this one is completed .

An Example of an RFP:-

RFP (all, task16);

Read back (monitor) the number of food objects PUT DOWN by all ants then go to Task Line ‘task16’ for the next instruction.

Monitor: Read number of CollisionS

RCS (Arg 1, Arg 2);
Arg 1 = string Name of ant or ‘all’.
Arg 2 = string The next Task Label to go to when this one is completed .

An Example of an RCS:-

RCS (ant_3, task10);

Read back (monitor) the number of collisions encountered by ‘ant_3’ then go to Task Line ‘task10’ for the next instruction.

Monitor: Read number of Imminent Collisions

RIC (Arg 1, Arg 2);
Arg 1 = string Name of ant or ‘all’.
Arg 2 = string The next Task Label to go to when this one is completed .

An Example of an RIC:-

RIC (ant_5, taskAA;

Read back (monitor) the number of imminent collisions encountered by ‘ant_5’ then go to Task Line ‘taskAA’ for the next instruction.

Monitor: Read number of food objects StoLen

RSL (Arg 1, Arg 2);
Arg 1 = string Name of ant or ‘all’.
Arg 2 = string The next Task Label to go to when this one is completed .

An Example of an RSL:-

RSL (ant_15, taskABoB;

Read back (monitor) the number of food objects stolen from others by ‘ant_15’ then go to Task Line ‘taskABoB’ for the next instruction.

Monitor: Read number of food objects LoSt

RLS (Arg 1, Arg 2);
Arg 1 = string Name of ant or ‘all’.
Arg 2 = string The next Task Label to go to when this one is completed .

An Example of an RLS:-

RLS (ant_13, mYtask;

Read back (monitor) the number of food objects stolen from ‘ant_13’ by others then go to Task Line ‘mYtask’ for the next instruction.

Monitor: Read General purpose Counter

RGC (Arg 1, Arg 2);
Arg 1 = string Name of ant or ‘all’.
Arg 2 = string The next Task Label to go to when this one is completed .

An Example of an RGC:-

RGC (ant_11, mYnext;

Read back (monitor) the general purpose counter of ‘ant_13’ then go to Task Line ‘mYnext’ for the next instruction.

FORK LINES and their ARGUMENTS

Variables that can be Monitored

These if commands have access to a choice of six variables:-

RFP = Read Food Placed
RCS = Read Collisions
RIC = Read Imminent Collisions
RSL = Read Stolen
RLS = Read Lost
RGC = Read General purpose Counter

IF: Fork If equals

IF= (Arg 1, Arg 2, Arg 3, Arg 4, Arg 5);
Arg 1 = string Name of ant or 'all'.
Arg 2= string Name of variable either RFP, RCS, RIC, RSL, RLS or constant integer.
Arg 3 = string Name of variable either RFP, RCS, RIC, RSL, RLS or constant integer.
Arg 4 = string The next Task Label to go to if the condition is not met.
Arg 5 = string The next Task Label to go to if the condition IS met.

An Example of an IF=:-

IF= (ant_2, RFP, 3, task09, task16)

Ant_2 to go to Task Line 'task16' for the next instruction if the value of RFP (food objects placed) is equal to 3. If not equal then go to 'task09' for the next instruction.

IF: Fork If greater than

IF> (Arg 1, Arg 2, Arg 3, Arg 4, Arg 5);
Arg 1 = string Name of ant or 'all'.
Arg 2= string Name of variable either RFP, RCS, RIC, RSL, RLS or constant integer.
Arg 3 = string Name of variable either RFP, RCS, RIC, RSL, RLS or constant integer.
Arg 4 = string The next Task Label to go to if the condition is not met.
Arg 5 = string The next Task Label to go to if the condition IS met.

An Example of an IF>:-

IF> (all, RCS, 12, task09, task16)

All ants to go to Task Line 'task16' for your next instruction if the value of RCS (collisions encountered) is greater than 12. If not greater then go to 'task09' for your next instruction.

IF: Fork If smaller than

IF< (Arg 1, Arg 2, Arg 3, Arg 4, Arg 5);
Arg 1 = string Name of ant or ‘all’.
Arg 2= string Name of variable either RFP, RCS, RIC, RSL, RLS or constant integer.
Arg 3 = string Name of variable either RFP, RCS, RIC, RSL, RLS or constant integer.
Arg 4 = string The next Task Label to go to if the condition is not met.
Arg 5 = string The next Task Label to go to if the condition IS met.

An Example of an IF<:-

IF< (all,6, RIC, task09, task16)

All ants to go to Task Line ‘task16’ for your next instruction if 6 is smaller than the value of RIC (imminent collisions encountered). If not smaller then go to ‘task09’ for your next instruction.