ATHABASCA UNIVERSITY

MULTI-AGENT SYSTEMS (MAS) MODELLING ENVIRONMENT FOR HUNTING AND EVASION ALGORITHMS

BY

MARC PRINCE

A project submitted in partial fulfillment
Of the requirements for the degree of
MASTER OF SCIENCE in INFORMATION SYSTEMS

Athabasca, Alberta

September 2017

APPROVAL PAGE

This Master's thesis essay is approved.

DEDICATION

This research is dedicated to the NetLogo Community which may assist future generations of researchers in evolving our understanding of Autonomous Agents and Multi-Agent Systems.

ACKNOWLEDGEMENTS

The author would like to acknowledge Dr. Fuhua (Oscar) Lin of Athabasca University for his guidance in the research and to the author's family for their support throughout its completion.

ABSTRACT

While new hunting algorithms are being developed to model pack animal hunting behaviours, these are being assessed with statistical methods rather than with simulation methods. The deficiency with statistical methods is that they represent a centralized approach to evaluating hunting algorithms rather than the decentralized method that exists in reality with real hunting agents (such as wolves and lionesses). Put another way, there is no central agent that controls the behaviour of real hunters (they independently collaborate towards common goals), thus any attempt to simulate these through a centrally controlled algorithm is artificial. To address this deficiency, this research proposed, designed, and developed a Hunting Algorithm Performance Evaluation Environment (HAPEE) using Multi-Agent System (MAS). Based on the Belief, Desire, and Intent architecture, the MAS were developed with the Netlogo agent-based modelling environment which includes the World Agent. The World Agent was customized to a limited number of two-dimensional scenarios with both Hunting Agents (HA) and Prey Agents (PA) interacting with each other and with the obstacles within the scenarios. The Hunting Agents and Prey Agents were developed with the flexibility to instantiate many types of hunters and prey. The differences in both the HAs and PAs was with their skills (communications, perception, speed, etc.) and their cognitive abilities. Finally, the HAPEE was used to evaluate two hunting algorithms: the Lion Optimization Algorithm (LOA) and the Grey Wolf Optimazation (GWO) algorithm. The experimental results show that the LOA is more resilient to obstacles than is the GWO. In the presence of obstacles, the lionesses were more reliable in completing joint convergence onto the prey (with the Alpha attacking first followed by the Beta). While the wolves had a lower convergence rate, they displayed an ability to recover from the confusion caused by obstacles to finally complete convergence (though it took much longer). Thus the lionesses either succeeded or failed to converge, whereas the wolves could initially fail

to converge, but then re-engage and succeed to converge at a later attempt. So while the lionesses were fairly binary in success or failure to converge on the prey, the wolves displayed a broader tolerance to recover from failure to finally converge onto the prey.

PREFACE

This research was conducted to determine the feasibility of developing a comprehensive hunting environment within which to test search, hunt, and evasion algorithms. The NetLogo Agent-Based-Modelling environment was chosen as it is open-source and very widely used by the academic community. By restructuring the NetLogo code into the Belief, Desire, Intent (BDI) framework, it is hoped that this research will enable future academics to develop increasingly complex hunting environments and help learn and test hunting algorithms.

This thesis paper should be read in conjunction with the NetLogo model which was programmed using NetLogo version 6.0. The NetLogo model can be downloaded from the NetLogo Modelling Commons at http://modelingcommons.org/account/login under the title of Hunting Algorithm Performance Evaluation Environment (HAPEE).

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES AND ILLUSTRATIONS

Page

LIST OF SYMBOLS, ABBREVIATIONS AND NOMENCLATURE

AA	Autonomous Agent
BDI	Belief, Desire, Intent
GA	Generic Agent
GWO	Grey Wolf Optimization
HA	Hunting Agent
HAPEE	Hunting Algorithm Performance Evaluation Environment
LOA	Lion Optimization Algorithm
MAS	Multi-Agent Systems
NetLogo	MAS development environment
PA	Prey Agent
WA	World Agent

CHAPTER 1

INTRODUCTION

Research Problem

The study of hunting techniques has long been the pre-occupation of mankind. Fundamental to its survival, humanity's skill in this field has distinguished it above all other life on Earth. The study of these techniques over the millennia has evolved into well-defined techniques for searching, hunting, and evasion. Examples of these techniques have been captured in literature such as *The SAS Handbook of Tracking & Navigation* (Wilson, 2002). Employed in various environmental conditions, each corresponding algorithm offers strengths and weaknesses depending on both the skills of the hunter and that of the prey being hunted.

Though humanity has evolved hunting techniques well beyond melee weapons, the essence of all hunting in the animal kingdom remains that of hand-to-hand combat. While modelling the attack of one hunter on one prey is straight forward, the modelling becomes quite complicated once there is more than one hunter and they are cooperating in the hunt for the prey. The same is true if there are more than one prey cooperating to defeat the hunters. And while many hunting algorithms are being proposed to model pack animals (wolves, lions, etc.), the evaluation of their effectiveness is being determined by computational (i.e. mathematical) methods only. These computational methods are also difficult to adapt to consider complex hunting environments (involving obstacles) which have an impact on the effectiveness of the hunting algorithm under evaluation.

Of real benefit would be a Multi-Agent system (MAS) simulation environment which would provide a cost-effective alternative to the existing centralized computing methods to further evaluate these search, hunting, and evasion algorithms. Also, a MAS simulation environment

would more accurately represent the decentralized thinking and communication that occurs with pack animals in a hunting scenario. In addition, this MAS environment could be more easily adapted to include physical obstacles which interfere with hunting, thus provide a better understanding of the performance of the hunting algorithms under various environmental conditions.

Research Uniqueness

This research is unique because it proposes to develop a customizable multi-agent environment which can be dynamically expanded (i.e., scaled) to include a variety of collaborative agent types, each skilled at various search, hunting, and evasion algorithms. As well, obstacles can be included in the hunting environment to test the effectiveness of these algorithms under less than ideal conditions; these ideal conditions form the basis of the current computational research. In addition, this research proposes to adapt the NetLogo – Agent Oriented Programming environment to a Belief, Desire, and Intent (BDI) architecture, which currently does not exist in the NetLogo community.

The Purpose of Research

The purpose of this research is to explore the feasibility of developing a simulation environment with MAS which can be used to assess the effectiveness of search, hunting, and evasion algorithms. This development environment, known as the MAS Hunting Algorithm Performance Evaluation Environment (HAPEE), would consist of a World Agent (WA), Hunter Agents (HA) and Prey Agents (PA), where the last two agent types would have access to search, hunting, and evasion algorithms. The Hunter Agents and the Prey Agents would interact with each other and

with the environment itself (i.e., the World Agent). The World Agent would be a dynamic environment that can be adapted to represent the conditions that exist in real world hunting environments. The HAPEE would be programmed in NetLogo and thus provide the tools to evaluate the performance of search, hunting, and evasion algorithms. The hunting algorithms will consist of the Grey Wolf Optimizer algorithm (Mirjalili, Mirjalili, & Lewis, 2014) and the Lion Optimizer Algorithm (Yazdani & Jolai, 2015). All three of the agents (WA, PA, and HA) will be structured as closely as possible to the Belief-Desire-Intent (BDI) architecture as detailed in Intelligent Agents (Wooldridge, 1999). The completed HAPEE model would then be submitted to the Netlogo development group for inclusion in the Models Library.

Hypothesis

The hypothesis postulates that NetLogo MAS Development Environment, structured in a modified BDI framework, provides a feasible MAS Hunting Algorithm Performance Evaluation Environment thus providing an alternative means to test Hunting Algorithms.

Organization of the Remaining Chapters

The remaining chapters are organized as follows. Chapter II reviews the existing literature and current related research. Chapter III describes the methodology of implementing the simulation environment with NetLogo and using it to evaluate the effectiveness of the various search, hunt, and evasion algorithms. Chapter IV examines the test results of the Grey Wolf Optimizer algorithm and the Lion Optimizer Algorithm. Chapter V discusses the findings of the research and the results of the algorithm testing. The last chapter concludes the research effort and proposes recommendations for further research within this field.

CHAPTER II

REVIEW OF RELATED LITERATURE

Related Work

The related work in this field can be divided into three main groups as displayed in the Concept Map of the Literature in Figure 1 below:

a. Autonomous Agent Design Theory (denoted by Green Bubbles).

b. Hunting and Prey Behavioural Algorithms (denoted by Blue Bubbles).

c. Autonomous Agent and MAS Design Environment (denoted by Black Bubbles).

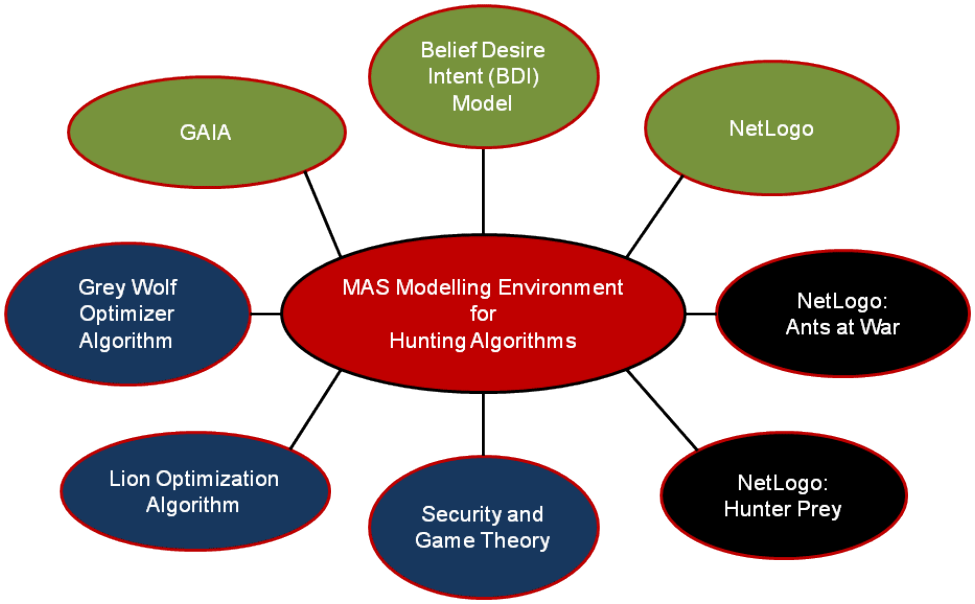

Figure 1: Concept Map of the Literature

The various areas of related work are described in more detail in the following sub-sections.

Autonomous Agent Design Theory and Architectures. While many design models exist for Autonomous Agents, the three considered for this research were Belief-Desire-Intent (BDI), GAIA, and NetLogo.

GAIA. The GAIA methodology is driven by requirements which are then grouped into two main models: the roles model and the interactions model [Wooldridge, 2000]. The GAIA architecture was considered because the Hunter Prey NetLogo Model, which was designed to evaluate a specific aspect of a hunter-prey environment (specifically the Contract Net Protocol), was programmed with this model as a framework.

NetLogo. The NetLogo Agent model is structured along the framework of "setup procedures" and "runtime procedures" where the agents' procedures are grouped under these main headings [Wilensky, 1999]. The NetLogo Agent model was considered as it was the native model for the NetLogo MAS development environment.

Belief-Desire-Intent. BDI has its origins in the Procedural Reasoning System (PRS) and has "proved to be the most durable agent architecture developed to date" [Wooldridge, 2009]. The architecture is structured around the beliefs, desires, and intentions of the Autonomous Agents. This AA architecture was considered because of its current relevance in the MAS field of study.

MAS Design Environment. Many MAS design environments exist within which to implement the agents and some of these environments are aligned with a corresponding architecture. Environments of these types are usually purpose built around a specific AA architecture. Examples of these include AgentSpeak [Machado & Bordini, 2003], V-Rep [Coppelia Robotics, 2017], and NetLogo. Two key aspects need to be considered in selecting a MAS Design Environment: the adaptability of the environment to the chosen AA architecture and the existence and maturity of the World Agent which controls the

environment. AgentSpeak [Machado & Bordini, 2003] is a design environment built specifically for a BDI architecture which has limited World Agents (these are custom built to a specific MAS implementation). V-Rep [Coppelia Robotics, 2017] is a design environment based on existing, commercially available robots which is not structured on any particular AA architecture. It has a three-dimensional World Agent which controls a limited surface area. Because it runs in 3 dimensions, it is computationally demanding and can support only a limited number of simultaneous Autonomous Agents. Though NetLogo has its native MAS architecture [Wilensky & Rand, 2015], it appears to be architecturally agnostic as seen by Wilhelmy, R. et al. (2014) who successfully adapted it to the GAIA architecture. It has a dynamic World Agent which can be programmed in two (2) and three (3) dimensions. It can also be interfaced with the Arduino robotic platforms to extend the AA and MAS into the physical world.

Search, Hunting and Evasion Algorithms. Many search pattern algorithms exist with respect to searching for objects or for prey. These patterns can be for an individual to execute by themselves or for groups to execute in a collaborative effort. Examples of search patterns include the expanding square search pattern, the expanding series of concentric circles pattern, and the track line pattern (see Figure 2 below).

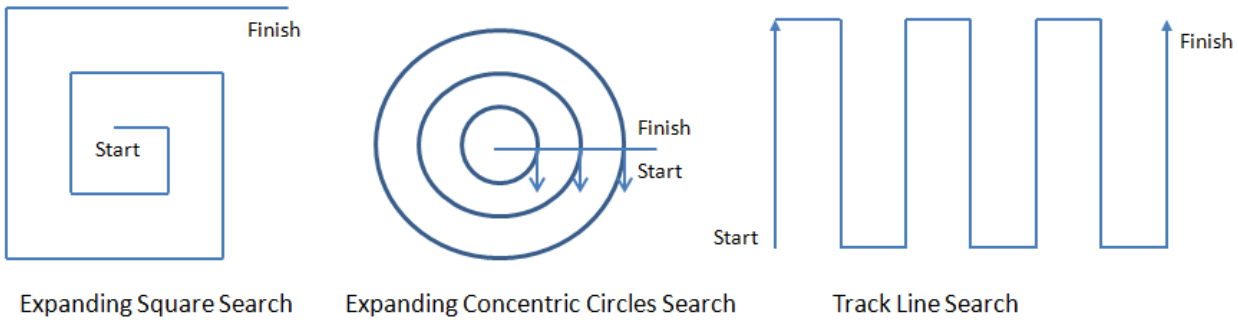

Figure 2: Search Patterns

Much literature exists on the hunting techniques of animals and many of these have been captured in videos and are available in television programming and most recently on YouTube. For example, Mech et al. (2015), have documented the hunting behaviours of wolves with respect to various prey types. Another facet of hunting techniques exists in modern warfare such as found in the Navy with surface and sub-surface (i.e., submarine) hunting methods. The development of evasion techniques has been studied as methods to avoid capture, and these are documented in the same manuals that study hunting techniques. For modern day land warfare, Wilson (2002) documents the evasion techniques used by the Special Air Service (SAS) to avoid capture while being pursued.

Current Research

Autonomous Agent Design Theory and Architectures. Research in Autonomous Agents has led to several design architectures, amongst them is GAIA, NetLogo, and Belief-Desire-Intent (BDI). While each has strengths and weaknesses, one of the major challenges of any AA architecture is in its implementation.

GAIA. Using NetLogo, Wilhelmy et al. (2014) structured the code for their MAS environment (hunter-prey scenario) according to the GAIA architecture. The result was lists of procedures roughly organised under individual agent types (hunter and prey) and functions (such as establishing the Contract Net). While the GAIA architecture worked for a small MAS environment (ie. hundreds of lines of code), it is questionable how scalable it is to larger code sets (ie. thousands of lines of code).

NetLogo. Much of the current research using NetLogo are but hundreds of lines of code and utilize the native architecture proposed by Wilensky & Rand (2015), but as with GAIA, given thousands of lines of code, the MAS would likely become unmanageable.

Belief-Desire-Intent. The BDI framework proposed by Wooldridge (1999), offers a structure that can perhaps be adapted to the NetLogo MAS design environment and used to organize thousands of lines of code.

Also, structured around the BDI architecture, Bordini and Hubner (2005) documented the processes within AgentSpeak and correlated them. In their model, Actions taken by the AA affect the environment including driving new *Perceptions*. While memory has been allocated for *Beliefs*, and the *Intentions* section hold the plans as they are selected from the *Plans Library*, there is no overt memory or process to account for the determination of desires.

The challenge then will be to see if the BDI architecture as proposed by Bordini and Hubner (2005) can easily be adapted to the NetLogo MAS design environment.

MAS Design Environments. In step with the evolution of AA architectures, the development of MAS design environments has also evolved over the last decade. Research continues to evolve in each of the three previously identified MAS environments: AgentSpeak, V-Rep, and NetLogo. Each of these design environments also have expanding libraries of reference material. Current research into expanding AgentSpeak has been in applicable World Agents and agent communication protocols (over the Internet). V-Rep continues to expand its library of physical robots as well as the

growing array of sensors that can be integrated into the robotic platforms. Current research in NetLogo over the last several years has led to significant expansion of the design environment. This has been fuelled by the fact that NetLogo is highly suitable for many academic disciplines ranging from the physical sciences (ex. physics) to the social sciences (ex. behavioural psychology). Because NetLogo appeals to such a broad range of scientists, it has been designed with simplicity in mind, to be used by non computer science researchers. One of the simplicities of its design is the fact that NetLogo is structured as a set of procedural calls. While vary basic, this structure is very easy to understand and enables code reuse between the various AAs. Several hunting projects have been recently completed in NetLogo such as the "Hunter-Gatherer Model" [Hosford, 2013] and "Ants at War" [Thomas, 2017].

Search, Hunting and Evasion Algorithms. Several hunting algorithms have been developed that postulate the hunting patterns of the followers (known as Betas, Deltas, and Omegas). An example is Oftadeh et al. (2010) who proposes a meta-heuristic optimization algorithm to converge the pack of hunters onto the prey. While these extend to land and sea based hunters, the land based algorithms in consideration for this research are the Lion Optimization Algorithm (LOA) [Yazdani & Jolai, 2015] and the Grey Wolf Optimizer (GWO) [Mirjalili et al., 2014] Algorithm.

Pack Leadership. The leadership within the pack is important as the hunting algorithms are often based on the movements of the pack leader. Selection of the pack leadership is critical for hunting animals as their very survival often depends on the skill of the leaders. The research conducted by Wilhelmy et al (2014) had

as a goal the establishment of hunting groups, and consequently pack leadership, through the Contract Net Protocol.

This research, which is focused on pack animals, assumes dominant leadership, and thus loyalty to the pack leader. Thus, when the pack leader summons the pack to hunt, there is no question of loyalty (or contract negotiation). Thus, the determination of pack leadership is out-of-scope of this research and, while important, is left to future development.

Pack Leader Hunting Techniques. Also of current research is that of the hunting algorithms, which have seen recent development, in the last ten years, within the scientific community. The research has been focused on the hunting algorithms that concern the follower hunters, and not with the pack leaders themselves. This is an important distinction because, while these algorithms propose the movement of the follower hunters, they do not suggest how the alpha hunter leads the pack to hunt their prey. The insight into the lead hunter's hunting techniques is left to other means of resolution. So, while the algorithm determining the follower hunters movement are given, the hunting strategy of the lead hunters are left to be developed. To address this critically important issue, the HAPEE must include some basic algorithms that can be used by the alpha hunter in leading their pack through a successful hunt.

Evaluation of Current Hunting Algorithms. The evaluation of current hunting algorithms is currently based on "solving complex optimization problems with metaheuristic algorithms...inspired by various phenomenon of nature" [Yazdini and Jolai, 2015]. Yasdini and Jolai (2015), also state that "Some algorithms

provide better solution for some particular problems compared with others. Therfore, pursuing for new optimization techniques is an open problem". The author agrees that further optimization techniques should be developed but also proposes that an alternative method for evaluating these algorithms be developed. In effect these current optimization techinques / algorithms establish a central control for the agents under observation, rather than the decentralized cooperation that exists in reality (with any hunting animal such as lionesses and wolves). As these evaluation environments are run through centrally controlled algorithms, a requirement exists to develop an evaluation environment that is decentralized, more accurately representing a real hunting environment, where hunting algorithms can be tested with cooperating Autonomous Agents (ie. within a Multi-Agent System).

Definition of Terms

Autonomous Agents (AA). These are computer agents with sensors that react to input based on the type and level of the sensory input and on their programming.

Belief-Desire-Intent (BDI). This is an AA design theory and architecture where the AAs are structured as having Beliefs, Desires, and Intentions.

Contract Net Protocol. This is a negotiation protocol where AAs decide which AA will be the leader for the session activity (ex. Hunt).

GAIA. This is an AA design theory and architecture where the AAs are structured as requirements grouped into the roles model and the interactions model.

Generic Agent (GA). Not a true AA as it is not instantiated, but is instead employed to enable code reuse amongst the other agents (PA and HA).

Grey Wolf Optimization (GWO). A hunting algorithm that predicts the convergence of omega wolves on a prey.

Hunting Agent (HA). This is an AA which hunts Prey Agents as a food source.

Lion Optimization Algorithm (LOA). A hunting algorithm that predicts the convergence of a beta lioness on a prey.

Multi-Agent Systems (MAS). This is a grouping of AAs which interact amongst each other, typically to accomplish a common goal.

Observer. This is the human that is establishing the World Agent parameters and running the MAS environment.

Prey Agent (PA). This is an AA which hunts stationary objects (such as apple trees) as a food source.

World Agent (WA). This is an AA which takes input from the Observer, establishes the MAS environment, and brokers the interactions between the various PAs and HAs throughout the runtime.

Summary

A review of the literature gave indications for the establishment of selection criteria for both the design architecture and environment in the evaluation of existing search, hunt, and evasion algorithms. The selection of an appropriate AA architectural model and MAS design environments is important because the HAPEE is going to need to be highly scalable. Also important is the ability to reuse code within the design environment in the instantiation of different types of agents. The ability to reuse code is also key to the scalability of the HAPEE model otherwise the code size could become repetitive, large, and unmanageable. The existing research also identified that a design environment should be chosen that can handle any type of

search, hunting, and evasion algorithms. With regards to selecting algorithms for pack leadership, the current research indicated that this could be addressed if we instantiate basic hunting procedures for the pack leaders.

CHAPTER III

METHODOLOGY

Using the NetLogo MAS design environment, the autonomous agents were programmed within the BDI architecture. Five types of AAs were developed: The World Agent (WA), the Patch Agent, the Generic Agent (GA), the Prey Agent (PA), and the Hunting Agent (HA).

BDI Architecture for NetLogo

To structure the NetLogo code within the BDI framework, adjustments to the BDI architecture as established by AgentSpeak were required, which introduced the Deliberation Cycle. Without a deliberation cycle, updates to beliefs did not consistently cascade through the autonomous agent. In a sense the Deliberation Cycle represents the AA's awareness, in that it is awake and constantly evaluating its environment through a cyclical process.

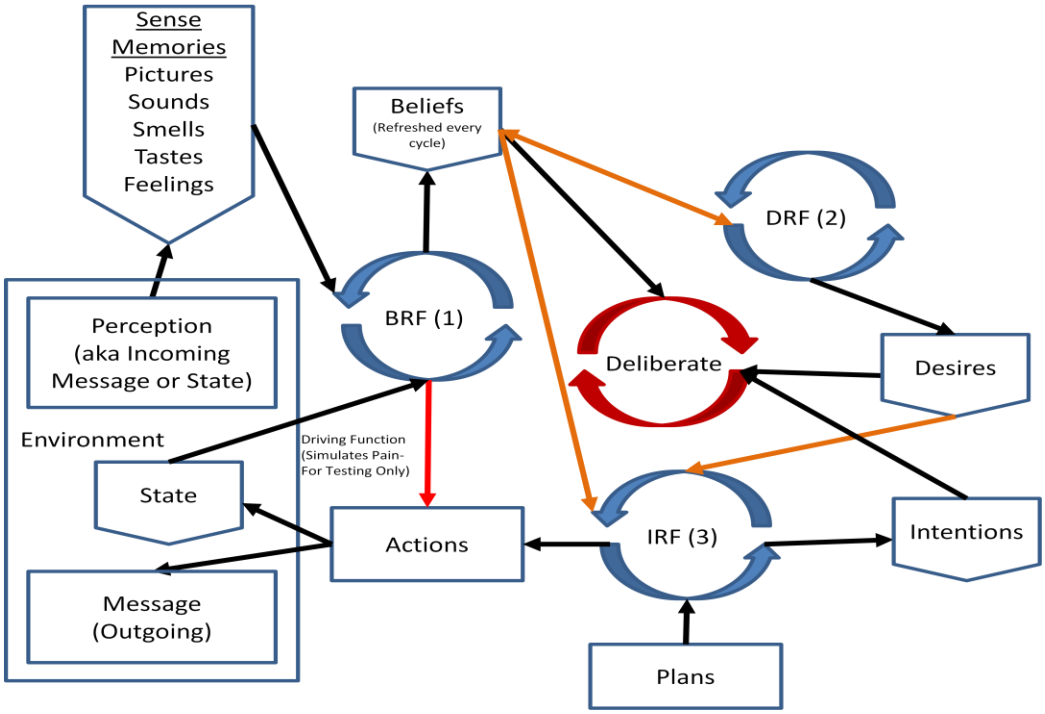

Figure 3: BDI Architecture for NetLogo

Perceptions and Sense Memory. There are two ways in which perceptions can be processed by the autonomous agent: as a driving function or through cyclical analysis. A

driving function not only updates the AA's beliefs but it then forces a deliberation cycle (aka the "filter function" in BDI) to conduct a review of intentions and thus desires. In a sense, it simulates pain. With analogy to a bee sting, the perception of the bee stinging updates the belief that the AA is under attack, which then forces a deliberation cycle, resulting in the change to intention and thus desire (to crush the bee). Cyclical analysis is driven by the Deliberation Cycle, representing a cycle of awareness which refreshes the Beliefs, based on current perceptions, through the Belief Review Function. The Hunting Environment utilizes both types of perception processing (driving function and cyclical analysis) to influence the AAs. Perceptions are stored in a very short-term memory called the Sense Memory which is updated with every Deliberation Cycle. Thus, it is highly ephemeral.

Beliefs and the Belief Review Function (BRF). Beliefs represent a longer-term memory of the AA's environment and these are updated either through a driving function (originating from Perceptions) or from a Belief Review Function (originating from a Deliberation Cycle). Beliefs are internal to the AA and they cannot be perceived within the World Agent by other AAs. As shown in Figure 3, Beliefs have influence in the Deliberation Cycle, DRF, and IRF.

Deliberation Cycle. Representing a cycle of awareness, the Deliberation Cycle takes input from the Beliefs, Desires, and Intentions, to then drive the Believe Review Function, the Desire Review Function, and the Intention Review Function through a review cycle. The Deliberation Cycle is intrinsically built into the NetLogo design environment and is activated (one cycle at a time) through the interface to the World Agent.

Desires and the Desire Review Function (DRF). Like a Belief, a Desire is a state which drives the AAs behaviour. A Desire is a highly persistent state that only changes once a need has been met. In the case of Prey Agents and Hunter Agents, the Desire to eat food drives their hunting behaviour (either for apples in the case of PAs and for deer in the case of HAs). Desires influence the Deliberation Cycle and the IRF.

Intentions and the Intention Review Function (IRF). An AA's Intentions are the courses of action that it intends to take in order to fulfill its desires. While these could be handled as a memory state, for the HAPEE, the AA's Intentions were procedural calls (to a given Intention) within which were further calls to one or more Plans.

Plans. Each Plan details a series of Actions that the AA will take that change the State of the AA or that of the World Agent.

Actions and States. The AA takes actions on itself or on the World Agent. Changing "heading" is an example of an Action an AA takes on itself while "eating" apples from a fruit tree is an example of an Action taken on the World Agent (through a Patch Agent). These Actions change the State of the AA or to the World Agent. Though more ephemeral, a State is also a message (such as a sound) as it moves through the World Agent.

NetLogo Design Environment

Perceptions. Four senses were developed for the Prey and Hunting Agents: visual, auditory, olfactory, and touch. Only the visual and auditory senses were fully enabled, touch was simply enabled through proximity, and while the World Agent was programmed to track odor types and levels, the ability to detect these was not enabled for

the HAs. Thus, only the visual, auditory, and touch senses were used in evaluating the hunting algorithms.

Sight. The sense of sight was enabled with two visual scanning algorithms which were developed for the agents. The first was a reactionary algorithm and the second was an analytical algorithm. They were both originally developed to enable the prey to find their plant food source (ex: apple tree), they were then adapted to the hunters. Both algorithms, described in PA_perceive_scan1 and PA_perceive_scan2, use the NetLogo commands of patch-ahead, patch-left-and-ahead, and patch-right-and-ahead to sequentially scan the ground in front (and left and right) of the Agent until a visual limit is reached. As shown in Figure 4 below, the patch-left-and-ahead establishes the left arc of the visual field and the patch-right-and-ahead establishes the right arc. The scan starts in the middle (from near to far distance) and sweeps left (from middle) and then right (from middle) by incrementing first by 5 degrees and then 10 degrees (for the remainder). The algorithms stop scanning along a heading if it encounters an object (such as a rock feature or tree feature) that the agent should not be able to see through. The advantage of the algorithm is that it is simple to implement, the disadvantage is that it covers the squares (closest to the agent) several times while missing squares at distance (thus creating blind spots).

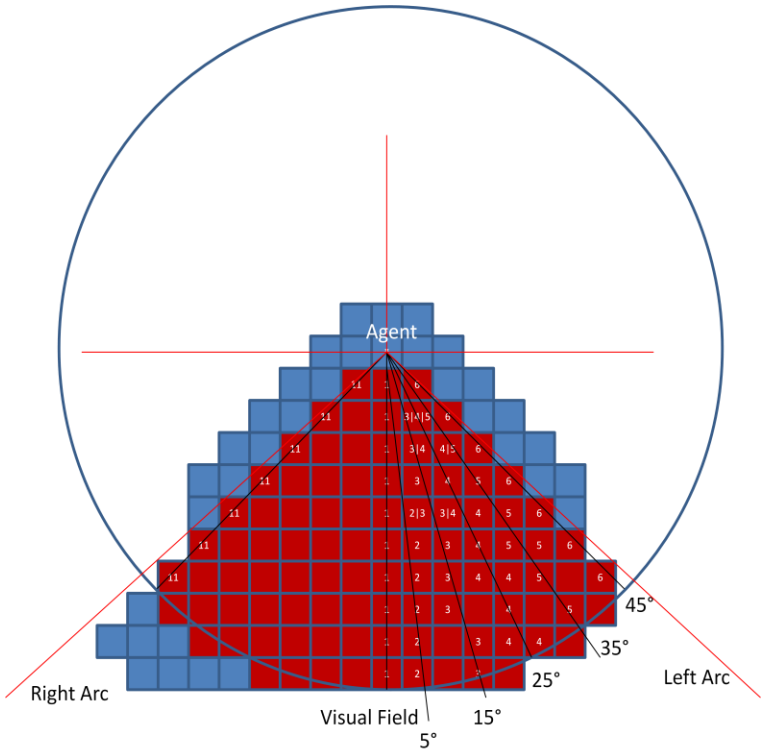

Figure 4: Visual Scanning Algorithm

The analytical scanning algorithm saves the objects perceived within the visual field of the agent and stores these in a perception array. The data in the perception array is then analysed later through any Belief Review Function (in this case PA_brf_food within the PA_brf_cycle) which uses it to identify patches of a certain colour.

Sound. The sense of hearing was enabled in the HAs through the HA_action_talk-prey_found procedure which would start the cycle of sound propagation with the WA procedure WA_brf_sounds-prey_found (which stores the sound in a State array that the WA initializes during setup through WA_brf_sounds-current-initialize and WA_brf_sounds-previous-initialize). The WA updates the sounds every Deliberation Cycle through the WA_brf_sounds-update_Current-to-Previous procedure and brokers these sounds with the Patch

Agents through the WA_Patch_brf_sounds procedure. This last procedure determines the residual sound level at the patch on which a PA or HA are occupying and lets them know whether a sound can be heard. The sound attenuation is given in Figure 5 below where a Level 1 sound (representing a whisper) can be heard on an adjecent square and a Level 7 sound (very loud) can be heard 200 square away. Note that this sound propagation algorithm can be adjusted to conform to any type of physical environment (as deserts and jungles have very different sound propagation properties).

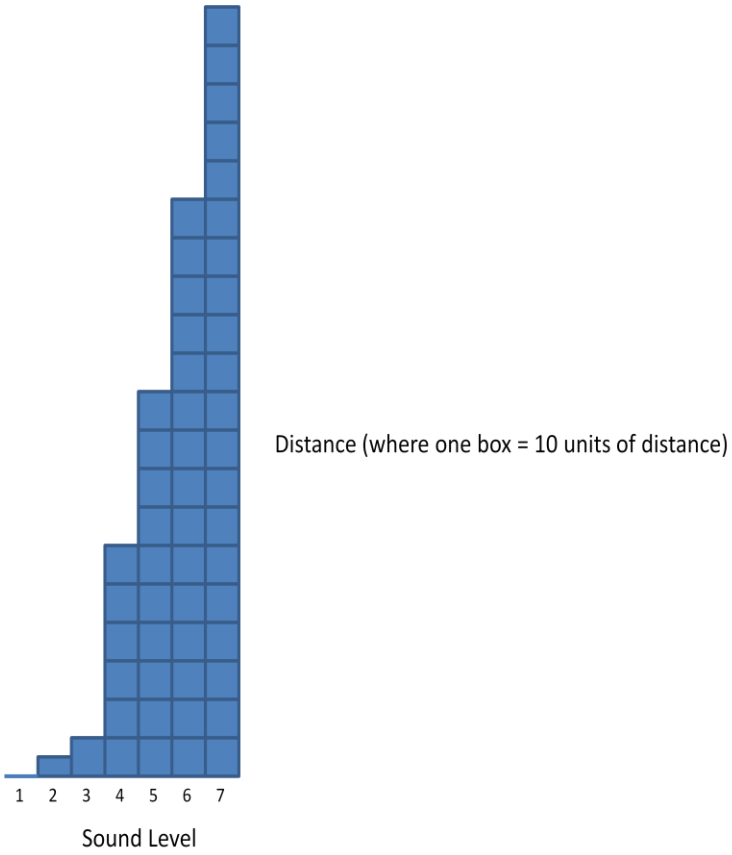

Figure 5: Graph of Sound Level vs Distance.

Smell. Each PA (through the GA) is given a unique scent (with GA_State_Scent-Type) and a scent level (with GA_State_Scent-Level). As a PA moves around the

WA, they leave their scent on the Patch Agents (through WA_Patch_perception_scent-initialize and WA_Patch_brf_scent_initialize). The level of smell gradually fades with WA_Patch_brf_scent_fade. An HA can detect these residual smells by asking the Patch (which it occupies) for any smells it holds. As long as the odor has not faded below the threshold of its olfactory capability, the HA will be able to detect that the PA had previously been on that same patch. As shown in Figure 6 below, the HA can then move in a circular pattern to determine the adjacent squares that also contain the odor and also determine their intensity level.

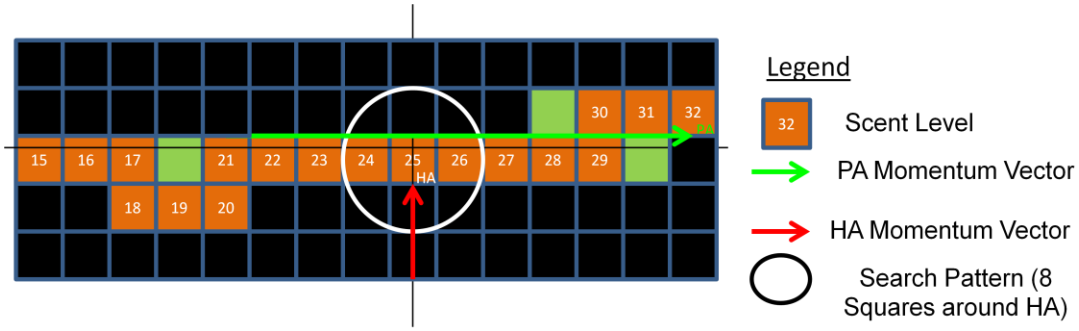

Figure 6: Hunting Agent Olfactory Sense.

As the smell fades with time, the HA will be able to determine the vector of the PA from lowest smelling patch (first patch occupied by the PA) to the highest smelling patch (last patch occupied by the PA). Once the HA determines the PA's vector, it simply follows the trail of odor (which is increasing) towards the current location of the PA.

Touch. The sense of touch was enabled with PAs and HAs through the occupation of the same patch. In the case of an HA and PA occupying the same patch, the call to procedure WA_Patch_brf_kill-prey set the patch to colour 19 thus denoting that the two Agents were collocated.

World Agent. The Human Operator (ie. Observer) interacts with the HAPEE through the World Agent's interface (shown in Figure 7). The buttons under Terrain (under the Environmental Setup Options (Perceptions) heading) are used to establish and test the density of the terrain features in the HAPEE. The buttons under the Weather heading are not active. The buttons under the heading Agent Setup Options (Perceptions) are used to establish the number of Prey Agents (and types) and Hunter Agents (and types) that will be instantiated in the HAPEE. The buttons under the heading World Agent Controls (Deliberation) are used to setup or reset the HAPEE or to cycle the HAPEE either once or continuously. As their title describes, the procedures that these buttons control are either located in the Perceptions or the Deliberation areas of the World Agent's code.

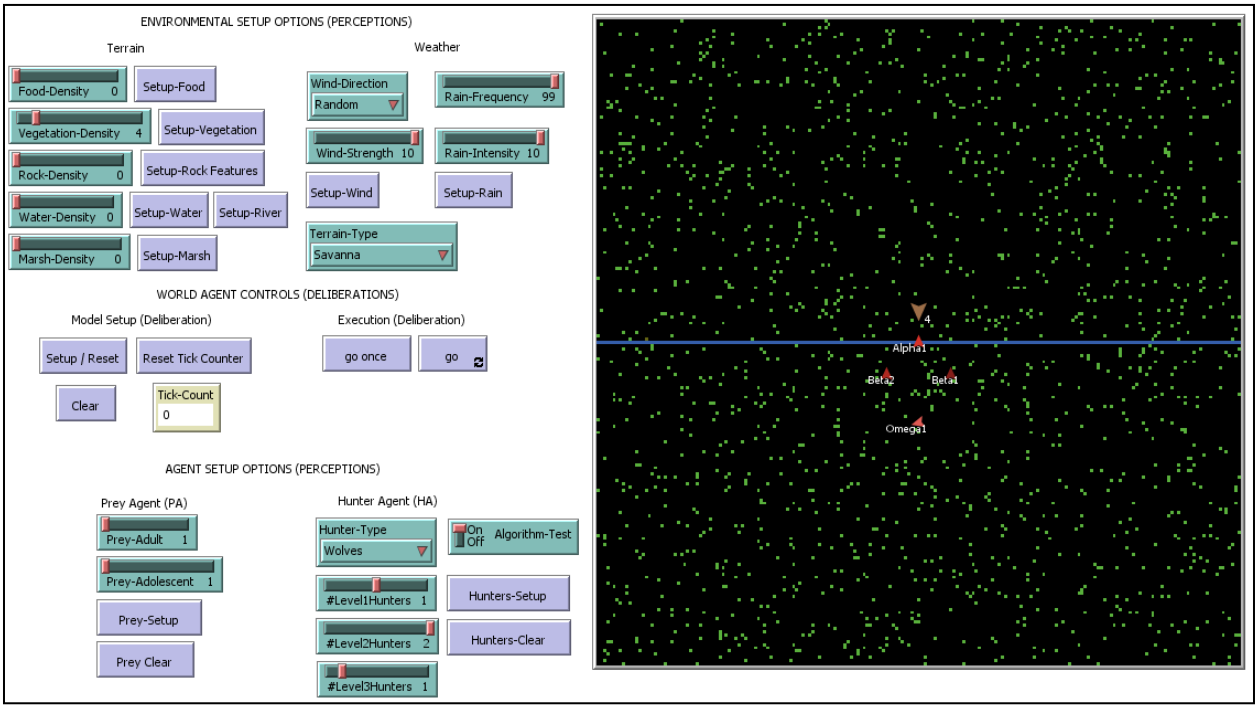

Figure 7: World Agent Interface.

The sequence of the setup of the World Agent is important and is shown in Figure 8. It is activated by pressing the Setup/Reset button. First the HAPEE is fully cleared of any

remnant Agents which would otherwise cause significant interference. Next the selections on the interface buttons are transferred to their equivalent WA variables. Then the sound arrays are initialized.

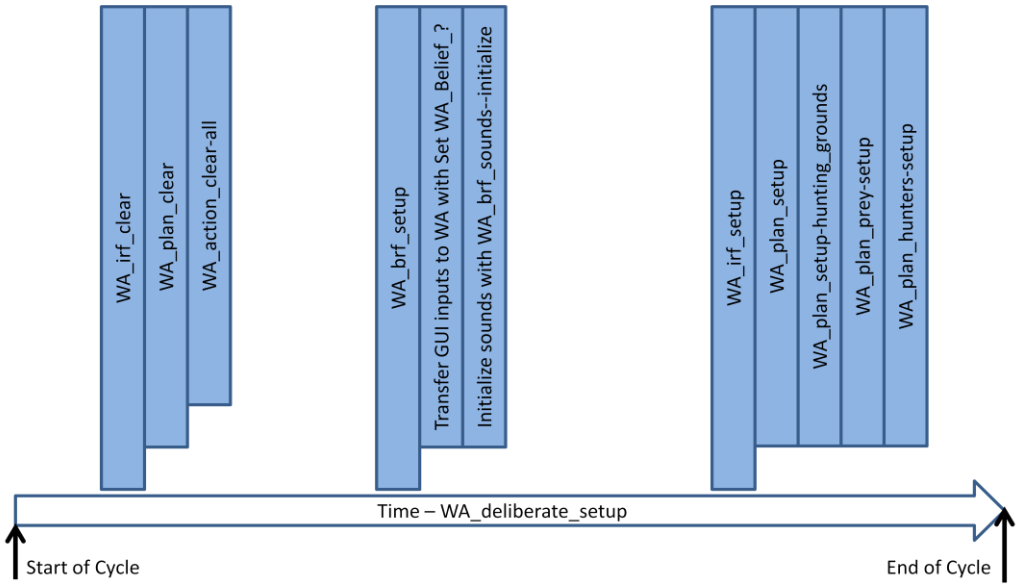

Figure 8: World Agent – Setup.

Next the Hunting Ground is initialized with the terrain feature densities selected prior to pressing the Setup/Reset button. Finally, the Prey Agents are instantiated followed by the Hunter Agents.

The World Agent Deliberation Cycle is activated once by pressing the “Go Once” button on the interface or continuously activated by pressing the “Go” button. At the start of its cycle it updates the patches through the WA_Patch_deliberate_cycle-start procedure which in turn: updates the sounds on patches, changes the colour of any patches with an HA occupying it, fades scents on patches, and grows food on those patches that are fruit trees. Each Prey Agent then steps through its Deliberation Cycle followed by a change in patch colour to that of the PAs.

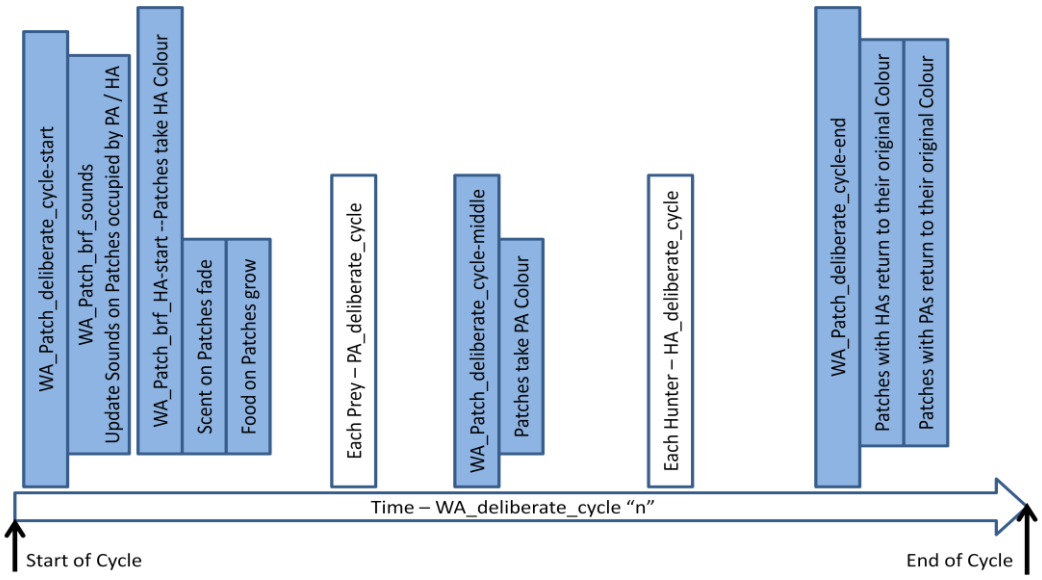

Figure 9: World Agent Deliberation Cycle.

All the HAs then step through their Deliberation Cycles followed by a return of all patches to their original colours. This cyclical process is detailed in Figure 9.

Generic Agent. The GA was developed to reuse code that is shared across all AAs regardless if they are PAs or HAs. While structured as a BDI AA, it is never instantiated in HAPEE. The GA is used to establish Design states as well as Belief states related to normal animal functioning (hunger, thirst, energy levels, etc.)

Prey Agent. The PA was designed around a simple Rest-Search-Eat cycle (Figure 10) which is activated by the Desire for food or water (GA_State_Energy-Current and GA_State_Water-Current).

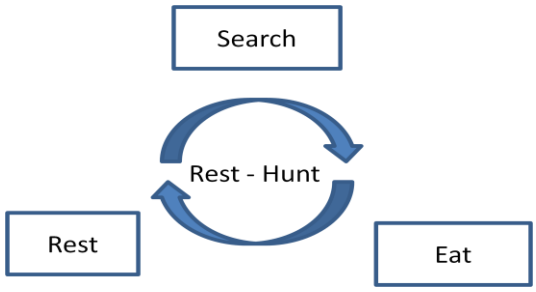

Figure 10: PA Activity Cycle.

While the PA has knowledge of the direction of the river, it searches for food based on a circular pattern of movement (like that of a partridge). When a PA encounters an HA, its evasion technique is simply to turn 180 degrees away from the HA and run away until exhausted.

Hunter Agent. The HA has a more complicated activity cycle as seen in Figure 11. Starting from Rest, the HA becomes hungry and begins the search for food. If it hears another wolf has found prey, then it will converge on that wolf. If it sees prey then it will alert the wolves as to its presence. Once the wolves converge on the Alpha wolf, they will encircle and then approach the prey. Once they get close enough, they will strike the prey and kill it. Eating terminates the cycle and the wolves return to a state of rest.

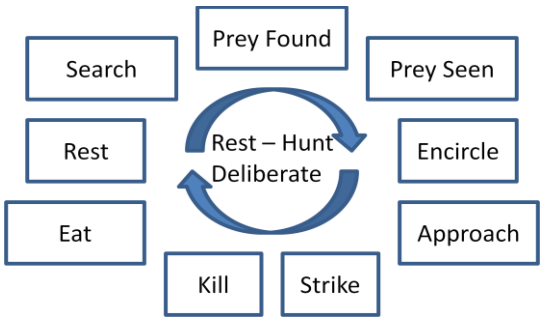

Figure 11: HA Activity Cycle.

Applied to one wolf, this cycle may seem simple. However, it is complicated by the fact that it controls the cohesion and the sequencing of the pack. This is particularly true for the encircle, approach, and strike phases of the hunt. To synchronize the activities of the wolves (or the lionesses), the code was developed around a hunting state (HA_Belief_Hunt-Mode) which is used to control all the HAs involved in the hunt. This detailed control process is depicted in Figure 12. While HAs can be at different states, the transition from one state to the other is dependent on the disposition of one HA with respect to the other (particularly the Alpha).

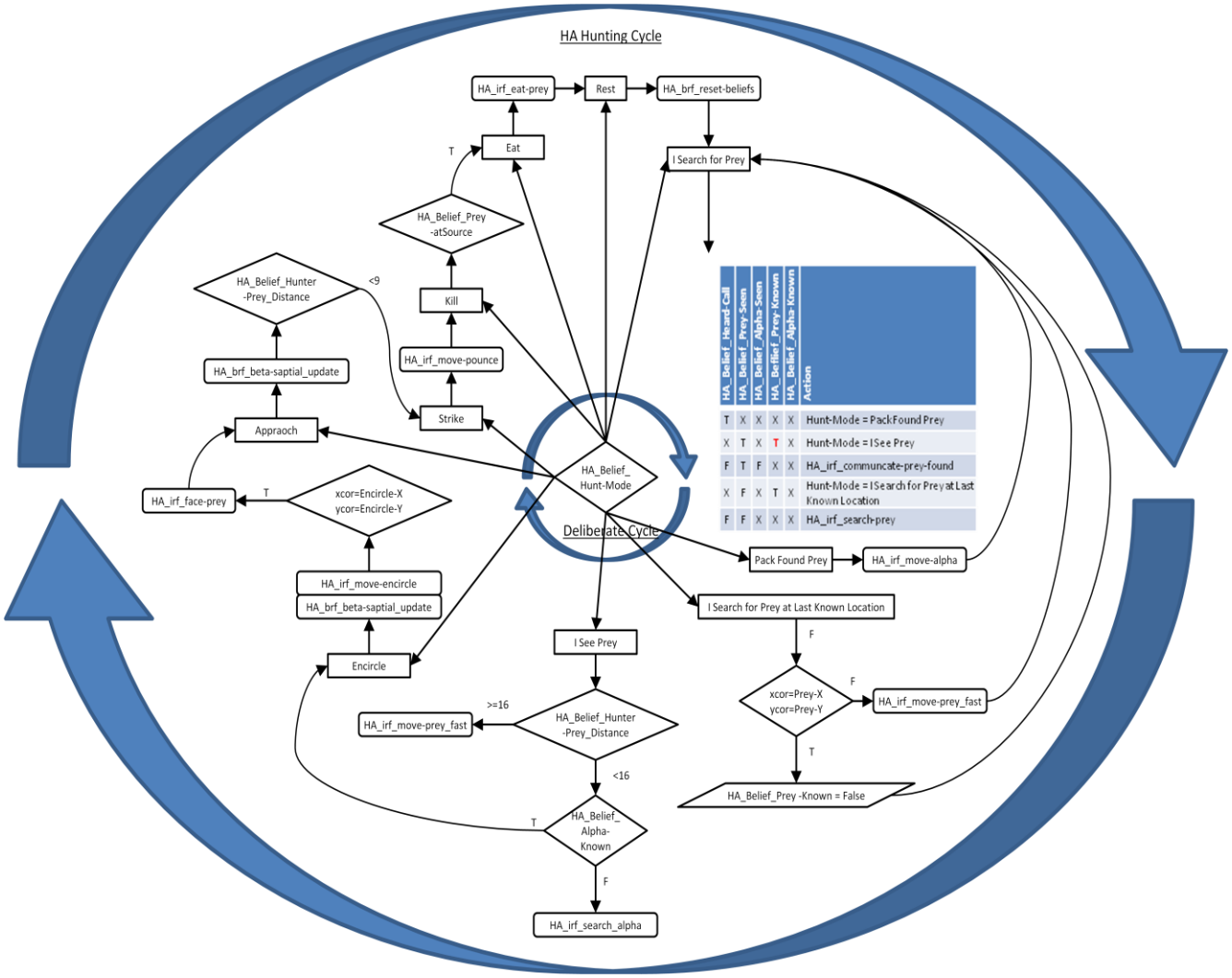

Figure 12: HA State-Based Hunting Cycle.

For instance, the wolves (or Beta lioness) will not transition to the strike phase unless the Alpha is seen striking the prey first. As discussed later, the Alpha typically strikes the prey from behind, thus initiating the strike from the other wolves (or Beta lioness). So, the Alpha roughly controls the pack, not through the use of messages, but through a state based process which controls the phasing of the overall hunting cycle.

Algorithms

As search algorithms lead the hunter to its prey, they form key building block as a precursor to hunting algorithms. Whether this is a wolf hunting a deer or a deer hunting an apple tree, the search algorithms are based on a specific sense (visual, hearing, smell/taste) and a corresponding movement pattern. In addition, a search algorithm can be completed alone or as a group (thus requiring communication).

Pack Leader Hunting Techniques. As discussed in Chapter 2, the hunting technique of the alpha hunter is critical to the execution of the hunting algorithms of the members of the hunting pack. While these are not given, they can in fact be deduced from the hunting algorithm of the follower (beta and omega) hunters, as these give strong indications of the alpha's hunting technique.

Alpha Lioness. For the Lion Optimizer Algorithm, the Beta hunter relies on the linear progression of the Alpha towards the prey. We also know that the Beta hunter strikes the prey in the face (by biting their snout) thus the Alpha's strike is 180 degrees from the prey's heading. The Beta only strikes once the Alpha lioness pounces on the prey's back end, sinking its claws into the rump of the animal and attempting to bite into the prey's lower spine (with intent to paralyse its back legs). We can therefore deduce that the Alpha hunter approaches the prey linearly from behind at 0 degrees offset from the prey's heading. This deduction is applied to the Alpha lioness in establishing her hunt algorithm.

Alpha and Beta Wolves. The Alpha wolf has a similar approach to the Alpha lioness in that they attack their prey from behind; this occurs naturally during the pursuit where the Alpha wolf is chasing the prey. Thus, it seemed a reasonable

deduction to mirror the Alpha wolf's hunting algorithm to that of the Alpha lioness. With the Alpha at the rear of the prey, the two Beta wolves take up position in the front and on either side of the prey. Prior to attacking, the two Beta wolves, Alpha wolf, and the prey form a "Y" with the prey at the junction point.

Lion Optimization Algorithm. The Lioness Hunting Algorithm controls the Beta's movement during the hunting cycle. Simple in its design and easily executed, the Beta lioness simply gets to a position in front of the prey which is inline with the Alpha lioness. As the Alpha approaches the prey from behind, the Beta lioness distracts the prey by approaching it from the front, matching the Alpha's distance from the prey. Once the Alpha pounces on the back of the prey, the Beta pounces on the front, sinking her claws into the shoulders of the prey, and biting its snout (in what appears to be an attempt to limit the prey's air intake).

Grey Wolf Optimizer Algorithm. The Grey Wolf Optimizer algorithm controls the Omega's movement and is in relation to the positions of the Alpha and two Beta wolves. With the Alpha directly behind the prey, the algorithm essentially places the Omega wolf directly in front of the prey which serves to distract it. Assuming that the Omega wolf has perfect visual targeting, the value of "A" is set to one (1) and the value of "C" is set to zero (0). This greatly simplifies the GWO to its fundamental structure by removing any ambiguity in the precise location of objects (whether prey, another wolf, or obstacles such as trees or rocks). It effectively localizes objects to the patch (square) on which they are located. This works well in the HAPEE because this is essentially how the visual perception is designed to function; where visual ambiguity is controlled by the visual perception algorithm and there is no additional requirement to further control the visual

ambiguity through the GWO. Thus, the GWO (equation 3.7) sums each of the position vectors from the Omega to the Alpha and Betas and then divides this by a factor of 3.

CHAPTER IV

RESULTS

BDI Architecture for NetLogo

With over four thousand lines of code for this project, the architecture provides a structured framework which is highly scalable. Organized into five major sections, additional sections could easily be added. As per Appendix-B, the code started with the declaration of variables for each of the five agent types: globals (for the World Agent), patches-own (for the Patch Agent), turtles-own (for the Generic Agent), hunted-own (for the Prey Agent), and hunters-own (for the Hunter Agent). The second section was the World Agent, the third section the Generic Agent, the fourth section the Prey Agent, and the final section was the Hunter Agent.

Each agent was sub-structured along its BDI functions where these were ordered alphabetically to simplify look-up. The sub-sections for each agent were structured as follows:

a. Actions;

b. Belief Review Function (BRF);

c. Deliberation;

d. Desire Review Function (DRF);

e. Intention Review Function (IRF);

f. Perceptions; and

g. Plans. Note that this section was only used for the World Agent. The GA, PA, and HA called Actions directly from the IRF.

The BDI Architecture proved conducive to supporting code reuse through the Generic Agent.

NetLogo MAS Design Environment

As we can see from Appendix-B, the result of programming the HAPEE was the generation of roughly 150 procedures. The adherence to nomenclature rules to distinguish the procedure from each other was very important. Table 1 below denotes the types and corresponding number of procedures used to program the HAPEE.

AA	Actions	BRF	Deliberate	DRF	IRF	Perceptions	Plans	Total
WA	15	16	4	0	3	11	5	54
Patch Agent	0	10	3	0	0	3	0	16
GA	8	5	1	2	1	0	0	17
PA	3	5	1	2	8	1	0	20
HA	13	11	5	0	11	4	0	44
Total	39	47	14	4	23	19	5	151

Table 1: Procedures Summary for the HAPEE.

Note that the there is no DRF for the HAs and this was purposely done to ensure these were always hungry and thus motivated to hunt.

Algorithm Evaluation

The results of the algorithm evaluation are documented at Appendix – B. The Lion Optimization Algorithm was tested first followed by the Grey Wolf Optimization algorithm.

Lion Optimization Algorithm

The following table is a summary of the results of the evaluation of the Lion Optimization Algorithm documented in Appendix – B.

(A) Trial Group	(B) Vegetation Density (%)	(C) Percentage of Convergences (%)	(D) Average Number of ΔTicks / Convergence
1	0	100	0
2	1	90	-0.1
3	2	90	+1.2
4	3	60	+0.2
5	4	50	+0.8

Table 2: Summary of LOA evaluation results

The results of the lionesses show that as the vegetation density increases, the percentage of convergences decrease with only a slight increase in the amount of time to complete the convergence.

Grey Wolf Optimizer Algorithm

The following table is a summary of the results of the evaluation of the Grey Wolf Optimization algorithm documented in Appendix – B.

(A) Trial Group	(B) Vegetation Density (%)	(C) Percentage of Convergences (%)	(D) Average Number of ΔTicks / Convergence
1	0	100	0
2	1	70	+1.4
3	2	50	+0
4	3	40	+14.0
5	4	30	+26.3

Table 3: Summary of GWO evaluation results

The results of the grey wolves show that as the vegetation density increases, the percentage of convergences decrease but with a significant increase in the time to complete the convergence.

CHAPTER V

DISCUSSION

For the discussion, the sections on BDI architecture and the NetLogo MAS design environment are treated together as the HAPEE is a combination of these two main functions.

BDI Architecture for NetLogo MAS Design Environment

Though the code at Appendix – B is lengthy, the adapted BDI Architecture for NetLogo has worked well to structure it and make it manageable. One of the means to keeping the code manageable was to ensure that states were only modified within their functional zones. This meant that Beliefs could only be modified in a BRF procedure, Desires in a DRF procedure, Intentions in an IRF procedure, and States within an Action procedure. While the occasional exception arose, the change of a state within a procedure not functionally related to that state had to occur through a call from a procedure that was functionally related. That way the state change was contained within the functionally related procedure (though directly executed by the embedded non-functionally related procedure). The BDI architecture simplified debugging as the state changes were contained to their functional areas of the code. Adapting the BDI Architecture to NetLogo also generated the following analysis during implementation.

Beliefs and Desires. While a belief and a desire are essentially the same in that they are both state functions (held in memory), it is the change in these states that reveals their differences. A change in a belief drives changes in desires (one or more), whereas a change in a desire drives a change in intentions. In a sense intentions are a function of desires which themselves are a function of beliefs. Intentions are also a direct function of beliefs and available plans. This is captured in the equation:

$$\text{intention} = (\text{desire}(\text{belief}), \text{belief}, \text{plan}) \quad (1)$$

which effectively summarizes the arrows pointing into the IRF in Figure 3.

Perceptions and Beliefs. Three methods were found of handling the way perceptions updated beliefs. The first method had two distinct functions, the Perceptions and the Belief Review Function, with a clear delineation between the functions. In this case changes to variables occurred within their respective functions but were copied to the variables in the other function to process them (ie. modify them). The second method combines the Perception variables directly into the BRF. The third method, opposite to the second method, has the Belief variables being directly updated within the Perception functions. The advantage of the first method is that code is separated and evolution of the perception functions can occur independently from the evolution of the belief review functions. Another advantage is that the results of a perception cycle can be used by several BRFs. The disadvantage is a requirement for additional memory and computational inefficiency due to the belief review functions having to examine essentially the same information an additional time to then change the beliefs; this increases the amount of code required and introduces additional latency (due to increased processing requirements). The second and third methods intertwine the variables of the Perception and Belief Review Functions, which has the advantage of being more computationally efficient and requiring less code. The disadvantage of these two methods is that the intertwined code becomes more difficult to evolve as they must be advanced together (there is no longer a clean interface between the two). Another disadvantage is that the results of a perception cycle cannot be used by more than one BRF; each BRF needs to run its own perception cycle. With a lot of BRFs, this would also become highly inefficient.

Examined from the perspective of an animal, these three methods of handling Perceptions and Beliefs are the difference between a learning animal and a reflex animal. A visual animal with high learning ability, will take a picture of its environment and then apply several BRFs to analyse it (and consequently update its beliefs based on the analysis). A visual animal with high reflex abilities, takes no picture of its environment, it turns perceptions directly into beliefs and then reacts accordingly. This simple code structuring gives indications as to the natural selection of higher intelligence vice faster reflexes. An animal of higher intelligence will have the ability to take one perception and analyse it to determine different conclusions (but at the cost of reaction time) while an animal of higher reflex will use a single perception to derive a single conclusion.

The method implemented in the HAPEE was that of a clear separation of the Perception and BRF functions. The perception data was brokered through the GA variable GA_Perceive_Visual-Scan which enabled the reuse of the visual scan data with the advantage that it permitted many Belief Review Functions to search that data set to make modifications to the Agent's beliefs (it was thus very efficient for the HAs as they had several BRF that needed to access the scan). The disadvantage was an increased time lag between the scan and its analysis by a BRF, so if the PA was in danger, then it took it more time to perceive the danger and react accordingly. Thus, the PAs had a lower reaction time than if a visual perception model had been selected that modified Beliefs in the Perception function.

Intentions, Plans, and Actions. While Beliefs and Desires have states, an AA's Intentions were simply procedural calls (to an IRF) which then further called plans, or in the case of the HA and PA, just called Action procedures. The use of plans was found to

be superfluous and the absence of these worked in structuring the PA and HA for the HAPEE because each Deliberation Cycle formed a unit of time (Tick) and each IRF was set to execute in that one cycle. This design decision assumed that the Actions executed in any IRF procedures (of both PAs and HAs) were temporally equivalent. Thus, in programming the HAPEE, care must be taken to ensure temporal equivalence of IRF procedures. If temporal equivalence is not maintained, then the Intentions would need to be states that regulate the execution of Actions in fixed periods of time.

Actions, States, and Perceptions. All Actions taken by the WA, Patch Agents, PAs or HAs (whether executed or not through the GA procedures) changed the States of one of these AAs through a call to one of their Perception procedures. Thus, the cycle of Action – Perception was completed as originally intended in the BDI Framework of Figure 3.

State Based Function Cycles. The design of the hunting cycle as a state controlled process, which is activated by the desire for food (GA_Desire_Food), prompted an extrapolation to the wider life cycle. As displayed in Figure 13, desires could be used to put AAs into additional functional cycles (other than the Hunting Cycle) such as a Pack Cycle, Reproduction Cycle, Home Building Cycle, or a Migration Cycle. Developing these cycles could evolve our understanding of evasion algorithms (for instance) as many animals use their homes as a defensive strategy to avoid capture.

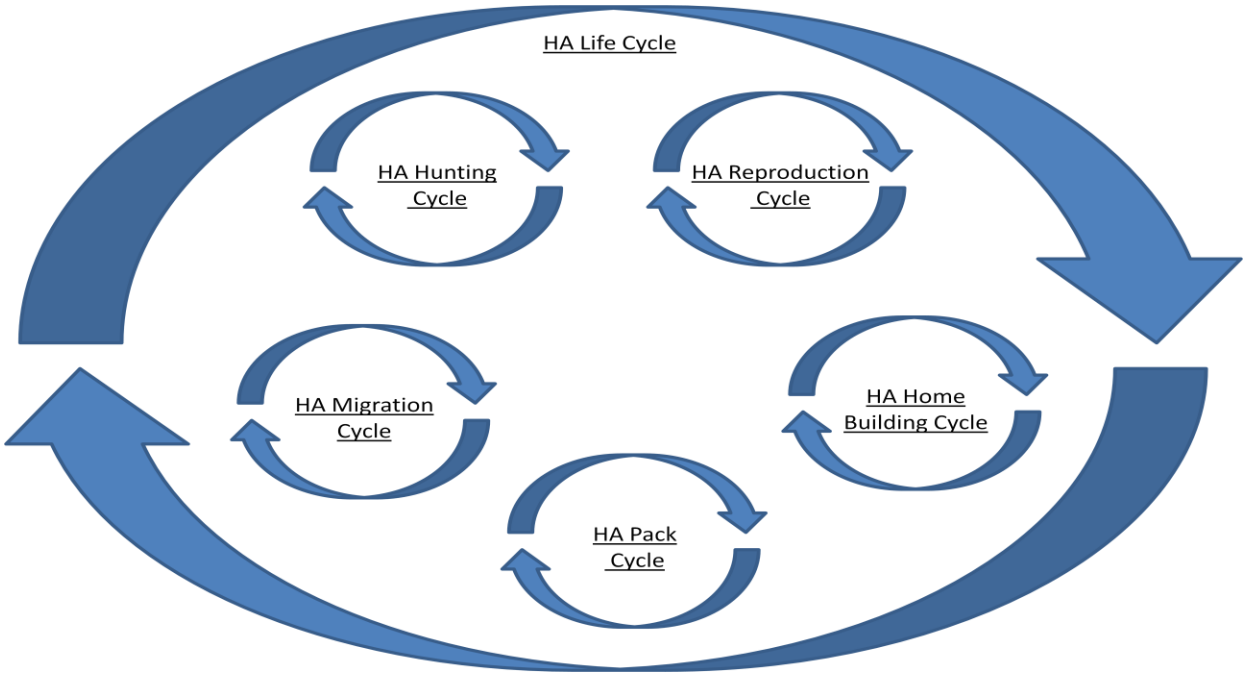

Figure 13: Functional Cycles within the Overall Life Cycle.

State Based Hunting, Task Focusing, and Hunting Algorithm Resilience. One of the observations of this study was that a state based hunting process (or algorithm) induced a task focusing where locking into a given state might trap the AA in that state. In a sense, state based hunting simulates the problem of gun-fighter vision narrowing (or task focus) where the HA becomes focused on the completion of a single task with little consideration to other tasks or other activities in its environment. A real concern with any tactical police force, the challenges of vision narrowing are addressed through repeated training where the gun-fighter practices a relaxation technique which draws him out of the vision narrowing, effectively bringing him from a state of "encircle", "approach" or "strike" back to a state of "search". Remaining in the task focused states of "approach" or "strike" are not only taxing on the gun-fighter (as they represent heightened states of alertness due to danger) but they will also lead to mistakes (often resulting in the injury of

innocent bystanders due to errors in adversarial target selection). The HAPEE then could be used to increase the resilience of HAs to quickly transition in and out of "encircle", "approach", and "strike" states back to "search" states. This will also enable an HA to quickly reacquire its Prey should it be lost during the "encircle, approach, or strike" phases.

Deliberate Cycle Timing and Persistent Memory. The NetLogo World Agent handles the interactions of the AAs within it in defined rounds (where each round is a Tick). At the start of a new round, two design options were available: the first option was to execute actions in play from last round and then conduct a new analysis (i.e. action-decision cycle where the round ends with the completion of the new analysis) or the round begins with a new analysis and then is finished at the completion of an action (i.e. decision-action cycle). While either design can work, the latter one is simpler to implement in coding as the decision–action cycle is contained within one temporal reference frame (round) rather than bridging two of them (decision being made in one round and the action being taken in the next round). Thus, the design decision was to start each round with an analysis of the current situation and then take an action according to the analysis. The use of persistent memory (memory that persists beyond the Deliberation Cycle) is then limited to the state based cycles (such as the Hunting Cycle).

Algorithm Evaluation

Using MATLAB, the graphs Trial Group vs. Convergence Percentages (%) were generated and then fitted with a linear curve.

Lion Optimization Algorithm. Graph 1 plots both the results of the LOA and GWO and fits the linear curve to the LOA. The equation of the fitted LOA curve is calculated to be y = -13x + 117.

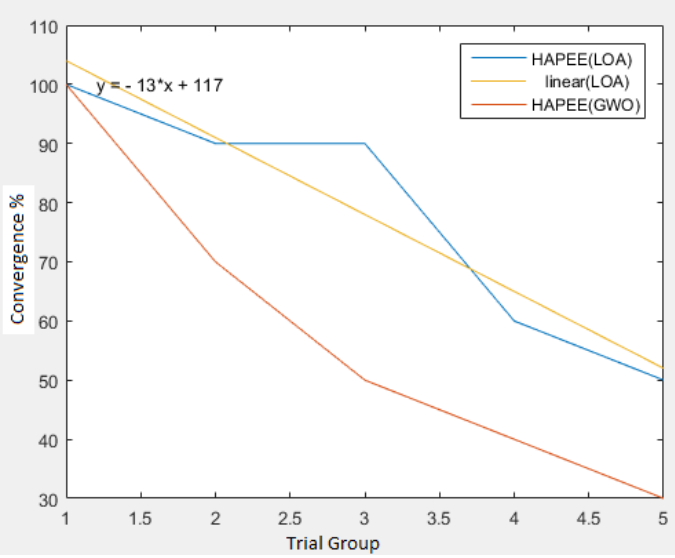

Figure 14: LOA fitted with a Linear Curve.

Grey Wolf Optimizer. Graph 2 plots both the results of the LOA and GWO and fits the linear curve to the GWO. The equation of the fitted LOA curve is calculated to be y = -17x + 109.

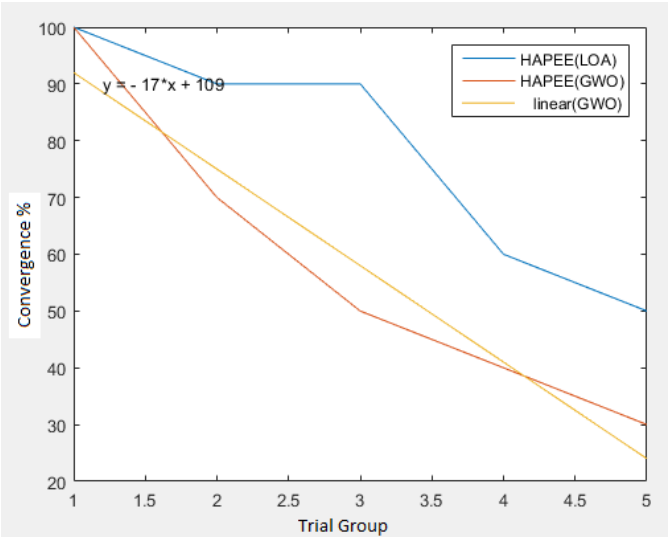

Figure 15: GWO fitted with a Linear Curve.

The GWO's steeper slope of -17x vise that of the LOA's at -13x gives strong indications that the GWO loses convergence faster in the presence of obstacles. This can be explained as we would expect the pack of wolves to have visual reference to each other in order to collaborate together. Even a small number of obstacles will disrupt their visual perception abilities to see each other. As the omega wolf needs to see the alpha and the betas to determine its position, it can easily become confused if it cannot see all the alpha and beta wolves. Because they work in pairs, the lionesses need only see each other, and there is no dependencies on other lions.

In evaluating the results of column D from Table 1, we can see that the lionesses either always converge on the prey (with little time variance (the average ΔTick < 1 Tick)) or they fail to converge. From column D of Table 2, with ΔTicks as high +26 Ticks, we see that the wolves may initially fail to convergence but are able to recover and re-engage the prey. This can be explained as the number of wolves which are surveying the hunting grounds are more numerous (by a factor of 2) than the number of lionesses. Thus, if the prey disappears behind an obstacle, it is far more probable for one of the wolves to have visual sight of it than it is for a lioness.

Perceptions and Search Algorithms. As search algorithms lead the hunter to its prey, they form a key building block as a precursor to hunting algorithms. Whether this is a wolf hunting a deer or a deer hunting an apple tree, the search algorithms are based on a specific sense (visual, hearing, smell/taste) and a corresponding movement pattern (whether, body, head, or just the eyes). The visual search algorithm implemented in HAPEE, based on a near-to-far radial scan, worked well in the near to mid distances but created blind spots in the far distance. Also, this visual search algorithm was inefficient in

the near field as it passed over the same square several times. A more thorough visual scan could be developed that inversed the pattern to that of a radial scan that then incremented from near-to-far (ie. a radial scan would occur at a fixed distance and then restarted at an incremented distance).

CHAPTER VI

CONCLUSION AND RECOMMENDATION

Conclusion

This research found that a BDI architecture enabled the NetLogo environment to effectively generate a HAPEE where two algorithms (LOA and GWO) were evaluated with the LOA being found to be more resilient to obstacles than the GWO.

BDI Architecture

As hypothesized, the Modified BDI Architecture could provide significant structuring to the NetLogo code thus making it feasible for a large-scale MAS environment.

NetLogo MAS Design Environment

Also, hypothesized, with the structuring provided by the Modified BDI Architecture, the NetLogo environment has proven to be able to establish a large-scale MAS Hunting Algorithm Performance Evaluation Environment (HAPEE) which is flexible and scalable.

Algorithms

The HAPEE found that the Lion Optimization Algorithm (LOA), in the presence of obstacles, was more effective at hunter convergence than the Grey Wolf Optimization algorithm. It also found that the Grey Wolf Optimization algorithm, after having lost sight of the prey, was more capable of re-acquiring and converging on the prey than was the Lion Optimization Algorithm. The LOA either succeeded or failed, whereas the GWO had a larger variance where it could recover from failure. Thus, as hypothesized, the HAPEE can be used to provide an alternative means to test Hunting Algorithms.

Suggestions for Further Research

Several areas of further research were alluded throughout this thesis paper. While the BDI Architecture is addressed in step with the NetLogo MAS Design Environment, the Algorithms are dealt with separately.

BDI Architecture and NetLogo MAS Design Environment

Learning vs. Dexterity. The simple code structuring of Perceptions vice Beliefs gives indications as to the natural selection of higher intelligence vice faster reflexes. As an analogy, the analytical algorithm favours a learning agent (with a scaling number of BRFs) while a reactionary algorithm favours an agent with high dexterity (aka agility). This simple design choice perhaps gives some indication as to the construct of the human mind. Humans with higher analytical and learning abilities often have lower dexterity than those humans that are very good at sports (high dexterity), but face challenges with their analytical thinking and abilities to learn diverse subjects. For further study, animals, that are not constantly under the threat of being hunted, are naturally selected to evolve their minds from those of high reflex ability to those that have a higher learning ability. Also, a third visual scanning algorithm could exist which would be a hybrid of the analytical and reactionary algorithms, this could be developed as a follow-on activity.

Temporal Equivalence. The BDI Architecture does not overtly address the temporal equivalence of a Deliberation Cycle amongst AAs (or within a MAS). There are good reasons for this, mainly because the BDI Architecture is meant for the design of AAs within the real World (not a World Agent) where time is real

and not artificially controlled by the World Agent. Thus, the temporal nature of Intentions, Plans, and Actions could be further studied to ensure equivalence across all the AAs within the HAPEE.

Weather. While the HAPEE was intended to handle simple weather conditions of wind and rain, the buttons under the Weather heading are not active and remain to be developed. These will be particularly important in the enabling of the olfactory sense of both PAs and HAs which is particularly influenced by the movement of air or the presence of air born water.

Functional Cycles within the AA Life Cycle. The HAPEE in its current state has only developed the Hunting Cycle and marginally developed the Pack Cycle (in instantiation). Further functional cycles remain to be developed within the life cycle of the AA.

Hunting Algorithm Resiliency. The ability of an AA to recover from vision narrowing (or task focus) caused by a state phase lock such as "encircle, approach, or strike" represents an increase in hunter algorithm resilience. Further research remains to continue to evolve the resiliency of the state based hunting process to be able to quickly transition in and out of the "encircle, approach, and strike" states back to a "search" (or prey acquisition) state.

Algorithms

Search Algorithms. While the HAPEE generated a visual search algorithm and basic (circular) physical search algorithm, many more remain to be developed and tested.

Visual Search Scan. A more thorough visual scan could be developed that inversed the pattern to that of a radial scan that then incremented from near-to-far. The design of other visual scans is highly encouraged.

Hunting Algorithms. While the LOA and GWO were tested with the HAPEE, it is hoped that the scientific community will evolve this environment to test other hunting algorithms.

Evasion Algorithms. A simple "turn and run" evasion algorithm was developed and utilised for this initial research. This area of research remains untapped and even a slight focus in this field could have significant results to organization such as police, special forces, and detentions (in the recovery of prisoners when they escape).

REFERENCES

Bordini, R. H., & Hubner, J. F. (2005). Computational Logic in Multi-Agent Systems, *3259*(April). http://doi.org/10.1007/b104175.

Coppelia Robotics. (2017). Virtual Robot Experimentation Platform (V-REP) version 3.4.0. Retrieved 21 February 2017 from www.coppeliarobotics.com

Hosford, A. (2013). NetLogo: Hunter-Gatherer Model Final Project. Retrieved from http://modelingcommons.org/browse/one_model/3782. Center for Connected Learning and Computer-Based Modeling, Northwestern University, Evanston, IL.

Machado, R., & Bordini, R.H. (2003). Running AgentSpeak(L) Agents on SIM AGENT. DOI: 10.1007/3-540-45448-9_12. Retrieved on 10 July 16 from https://www.researchgate.net/publication/2910734_Running_AgentSpeakL_Agents_on_SIM_AGENT

Mech, D. L., Smith, W. S., & MacNulty, D. R. (2015). Wolves on the Hunt: The Behavior of Wolves Hunting Wild Prey. The University of Chicago Press, Chicago, Ill.

Mirjalili, S., Mirjalili, S.M., and Lewis, A. (2014). Grey Wolf Optimizer. Advances in Engineering Software, vol. 69, pp. 46-61. http://dx.doi.org/10.1016/j.advengsoft.2013.12.007

Oftadeh, R., Mahjoob, M.J., & Shariatpanahi, M. (2010). A novel meta-heuristic optimization algorithm inspired by group hunting of animals: Hunting search. *Comput. Math. with Appl.*, vol. 60, no. 7, pp. 2087–2098.

Thomas, J. (2017). NetLogo: Ants At War. Retrieved from http://modelingcommons.org/browse/one_model/4828#model_tabs_browse_info. Center for Connected Learning and Computer-Based Modeling, Northwestern University, Evanston, IL.

Wilhelmy, R. et al. (2014). Multiagent modeling of a hunter-prey scenario using ContractNET. *Researchgate.Net*, (June 2015). Retrieved from http://www.researchgate.net/publication/248380879_Multiagent_modeling_of_a_hunter-prey_scenario_using_ContractNET/file/5046351ddc0017a1d3.pdf

Wilensky, U. (1999). NetLogo. http://ccl.northwestern.edu/netlogo/. Center for Connected Learning and Computer-Based Modeling, Northwestern University, Evanston, IL.

Wilensky, U., & Rand, W. (2015). An Introduction to Agent-Based Modelling: Modeling Natural, Social, and Engineered Complex Systems with NetLogo. The MIT Press: Massachusetts, USA.

Wilson, N. (2002). The SAS Handbook of Tracking & Navigation. Amber Books Ltd: London, UK.

Wooldridge, M. (1999). Intelligent Agents. In G. Weiss (Ed.), *Multiagent systems: A modern approach to distributed artificial intelligence* (pp. 27-79). Cambridge, MA: The MIT Press.

Wooldridge, M., Jennings, N. R., & Kinny, D. (2000). The Gaia Methodology for AgentOriented Analysis and Design. Autonomous Agents and Multi-Agent Systems 3(3): 285–312, 285–312.

Wooldridge, M. (2009). An introduction to MultiAgent Systems – 2nd Edition. Chichester, West Sussex: John Wiley and Sons, Ltd.

Yazdani, M., & Jolai, F. (2015). Lion Optimization Algorithm (LOA): A Nature-Inspired Metaheuristic Algorithm. *J. Comput. Des. Eng.*, vol. 3, no. 1, pp. 1–14.

APPENDIX – A

ALGORITHM TEST RESULTS

Lioness Hunting Results

Trial Number	Vegetation Density (%)	Tick-Count until Convergence	Convergence	dT
0	0	20	Y	n/a
1	1	20	Y	0
2	1	19	Y	-1
3	1	20	Y	0
4	1	20	Y	0
5	1	20	Y	0
6	1	20	Y	0
7	1	20	Y	0
8	1	-	N	-
9	1	20	Y	0
10	1	20	Y	0
1	2	26	Y	+6
2	2	15	Y	-5
3	2	20	Y	0
4	2	26	Y	+6
5	2	-	N	-
6	2	19	Y	-1
7	2	19	Y	-1
8	2	20	Y	0
9	2	26	Y	+6
10	2	20	Y	0
1	3	-	N	-
2	3	-	N	-
3	3	-	N	-
4	3	-	N	-
5	3	20	Y	0
6	3	22	Y	+2
7	3	20	Y	0
8	3	19	Y	-1
9	3	20	Y	0
10	3	20	Y	0
1	4	-	N	-
2	4	19	Y	-1
3	4	-	N	-
4	4	19	Y	-1
5	4	19	Y	-1
6	4	-	N	-
7	4	-	N	-
8	4	27	Y	+7
9	4	-	N	-
10	4	20	Y	0

Wolf Hunting Results

Trial Number	Vegetation Density (%)	Tick-Count until Convergence	Convergence	dT
0	0	19	Y	n/a
1	1	-	N	-
2	1	19	Y	0
3	1	-	N	-
4	1	29	Y	+10
5	1	19	Y	0
6	1	19	Y	0
7	1	19	Y	0
8	1	-	N	-
9	1	19	Y	0
10	1	19	Y	0
1	2	-	N	-
2	2	16	Y	-3
3	2	-	N	-
4	2	20	Y	+1
5	2	-	N	-
6	2	20	Y	+1
7	2	19	Y	0
8	2	-	N	-
9	2	-	N	-
10	2	20	N	+1
1	3	18	Y	-1
2	3	-	N	-
3	3	-	N	-
4	3	-	N	-
5	3	-	N	-
6	3	33	Y	+14
7	3	-	N	-
8	3	62	Y	+43
9	3	19	Y	0
10	3	-	N	-
1	4	-	N	-
2	4	-	N	-
3	4	75	Y	+56
4	4	-	N	-
5	4	29	Y	+10
6	4	-	N	-
7	4	32	Y	+13
8	4	-	N	-
9	4	-	N	-
10	4	-	N	-

APPENDIX – B

HAPEE – NETLOGO CODE

```
;@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@;
;@@@@ MAS HUNTING ALGORITHM PERFORMANCE EVALUATION ENVIRONMENT @@@@;
;@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@;
;@@@@                                              @@@@;
;@@@@  Copyright 2017 Marc Prince.                           @@@@;
;@@@@  See Info tab for full copyright and license.           @@@@;
;@@@@                                              @@@@;
;@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@;
;@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@;

; CODE FOR TROUBLESHOOTING
; show [(word "("pxcor ", " pycor")")] of self


extensions [rnd]

globals [ ; These are detailed in the Beliefs, Desires, Intentions of the World Agent
  WA_Belief_Food-Density          WA_Belief_Vegetation-Density       WA_Belief_Rock-Density            WA_Belief_Water-Density     WA_Belief_Open-Terrain
  WA_Belief_Marsh-Density         WA_Belief_Wind-Direction           WA_Belief_Wind-Strength           WA_Belief_Rain-Frequency
  WA_Belief_Rain-Intensity        WA_Belief_Prey-Adult_Number
  WA_Belief_HA-Alpha_Quantity       WA_Belief_HA-Beta_Quantity        WA_Belief_HA-Omega_Quantity        WA_Belief_HA-Type           WA_Belief_HA-Algorithm_Test
  WA_State_Sounds-Current_Cycle    WA_State_Sounds-Previous_Cycle
  WA_State_Agents-List
  WA_State_Patches-List
]

patches-own [ ; These are detailed in the World Agent under Patch Agent
  WA_Patch_State_Type              WA_Patch_State_Grow-Time
  WA_Patch_State_Colour            WA_Patch_State_Colour-Temp
  WA_Patch_State_Agent-Number      WA_Patch_State_Agent-Heading       WA_Patch_State_Agent-Speed
  WA_Patch_State_Scent-Type        WA_Patch_State_Scent-Level
  WA_Patch_State_Sounds
]

turtles-own [ ; These are detailed in the Generic Agent's Beliefs, Desire, Intentions.
  GA_State_Agent-Type    GA_State_Agent-Class    GA_State_Exhaustion        GA_State_Scan-Distance
  GA_State_Scent-Type    GA_State_Scent-Level    GA_State_Energy-Current    GA_State_Speed-Current     GA_State_Water-Current     GA_State_Voice-Level
  GA_Design_Speed-Max    GA_Design_Energy-Max    GA_Design_Hearing-Level-Min  GA_Design_Vocal-Level-Max  GA_Design_Scent-Level-Min  GA_Design_Vision-
          Max
  GA_Belief_Hunger-Level  GA_Belief_Water-Level    GA_Belief_Water-Bearing     GA_Belief_Patch-Colour     GA_Belief_Patch-Type
  GA_Belief_Water        GA_Belief_Water-Heading  GA_Belief_Water-Distance    GA_Belief_Water-Sources    GA_Belief_Water-atSource
  GA_Desire_Food         GA_Desire_Water          GA_Desire_Evade
  GA_Perceive_Visual-Scan
]

breed [ hunted prey ]
hunted-own [ ; These are detailed in the Prey Agent's Beliefs, Desire, Intentions.
  PA_Belief_Food         PA_Belief_Food-Heading    PA_Belief_Food-Distance    PA_Belief_Food-Sources    PA_Belief_Food-atSource
  PA_Belief_Hunter       PA_Belief_Hunter-Heading  PA_Belief_Hunter-Distance  PA_Belief_Hunter-Sees-Me  PA_Belief_Hunter-atSource
  PA_Desire_Heading-Next  PA_Desire_Hunger-Thirst
]

breed [ hunters hunter ]
hunters-own [ ; These are detailed in the Hunter Agent's Beliefs, Desire, Intentions.
  HA_State_Hunter-Type          HA_State_Hunter-Class         HA_State_Hunter-Level          HA_Belief_Hunter-Leader
  HA_Belief_Hunt-Mode
  HA_Belief_Prey-Seen           HA_Belief_Prey-Known          HA_Belief_Hunter-Prey_Heading    HA_Belief_Hunter-Prey_Distance
  HA_Belief_Prey-X              HA_Belief_Prey-Y              HA_Belief_Prey-Heading          HA_Belief_Prey-Speed
  HA_Belief_Prey-atSource       HA_Belief_Prey-Sources
  HA_Belief_Alpha               HA_Belief_Alpha-Seen          HA_Belief_Alpha-Known           HA_Belief_Hunter-Alpha_Heading    HA_Belief_Hunter-Alpha_Distance
  HA_Belief_Alpha-X             HA_Belief_Alpha-Y             HA_Belief_Alpha-Heading         HA_Belief_Alpha-Speed
  HA_Belief_Alpha-Prey_Distance  HA_Belief_Alpha-Prey_Heading   HA_Belief_Alpha-Prey_Attack
  HA_Belief_Beta1               HA_Belief_Beta1-Seen          HA_Belief_Beta1-Known           HA_Belief_Hunter-Beta1_Heading    HA_Belief_Hunter-Beta1_Distance
  HA_Belief_Beta1-X             HA_Belief_Beta1-Y             HA_Belief_Beta1-Heading
  HA_Belief_Beta1-Prey_Distance  HA_Belief_Beta1-Prey_Heading   HA_Belief_Beta1-Encircle-X      HA_Belief_Beta1-Encircle-Y
  HA_Belief_Beta2               HA_Belief_Beta2-Seen          HA_Belief_Beta2-Known           HA_Belief_Hunter-Beta2_Heading    HA_Belief_Hunter-Beta2_Distance
  HA_Belief_Beta2-X             HA_Belief_Beta2-Y             HA_Belief_Beta2-Heading
  HA_Belief_Beta2-Prey_Distance  HA_Belief_Beta2-Prey_Heading   HA_Belief_Beta2-Encircle-X      HA_Belief_Beta2-Encircle-Y
  HA_Belief_Omega1              HA_Belief_Omega1-Seen         HA_Belief_Omega1-Known
  HA_Belief_Omega2              HA_Belief_Omega2-Seen         HA_Belief_Omega2-Known
  HA_Belief_Heard-Call          HA_Belief_Heard-Alpha
  HA_Belief_Encircle-X          HA_Belief_Encircle-Y
  HA_Perceive-Sounds            HA_Perceive-Scents
]

links-own [_range]

;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;---------------------;;
;;;;; World Agent ;;;;;;;;
;;---------------------;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;

;;;;;;;;; Actions ;;;;;;;;;;;;;
; Nomenclature: WA_action_subject_verb (refactoring still needs to be completed)
```

```
to WA_action_clear-all
  clear-all
  reset-ticks
end

to WA_action_clear-ticks
  reset-ticks
end

to WA_action_hunters-die
  ask hunters [die]
end

to WA_action_hunters-setup

  if WA_Belief_HA-Type = "Lionesses" [
    HA_belief_instantiate-lioness_alpha
    HA_belief_instantiate-lioness_beta
  ]

  if WA_Belief_HA-Type = "Wolves" [
    let i WA_Belief_HA-Alpha_Quantity
    while [ WA_Belief_HA-Alpha_Quantity != 0 ] [
      HA_belief_instantiate-wolf_alpha
      set WA_Belief_HA-Alpha_Quantity (WA_Belief_HA-Alpha_Quantity - 1)
    ]
    set WA_Belief_HA-Alpha_Quantity i

    set i WA_Belief_HA-Beta_Quantity
    while [ WA_Belief_HA-Beta_Quantity != 0 ] [
      HA_belief_instantiate-wolf_beta
      set WA_Belief_HA-Beta_Quantity (WA_Belief_HA-Beta_Quantity - 1)
    ]
    set WA_Belief_HA-Beta_Quantity i

    set i WA_Belief_HA-Omega_Quantity
    while [ WA_Belief_HA-Omega_Quantity != 0 ] [
      HA_belief_instantiate-wolf_omega
      set WA_Belief_HA-Omega_Quantity (WA_Belief_HA-Omega_Quantity - 1)
    ]
    set WA_Belief_HA-Omega_Quantity i
  ]

end

to WA_action_prey-die
  ask hunted [die]
end

to WA_action_prey-setup

  let i 0
  while [ (i < WA_Belief_Prey-Adult_Number) ] [
    PA_belief_instantiate-hunted_prey_adult
    set i i + 1
  ]
end

to WA_action_setup-axes
  ;; draw x-axis & y-axis as light gray
  ask patches with [pxcor = 0 or pycor = 0]
    [ set pcolor gray - 3 ]
end

to WA_action_setup-food
  let random-value random 100
  let x-value min-pxcor
  let y-value min-pycor

  while [ (x-value <= max-pxcor) ] [
    set y-value min-pycor
    while [ (y-value <= max-pycor)] [
      set random-value random 100
      if random-value <= WA_Belief_Food-Density [
        ask patches with [ pxcor = x-value and pycor = y-value] [set pcolor orange]
      ]
      set y-value (y-value + 5)
    ]
    set x-value  (x-value + 5)
  ]
end

to WA_action_setup-marsh
  let random-value random 100
  let x-value 0
  let y-value 0
  ask patches with [ pcolor = blue ] [
    set random-value random 100
    if random-value <= WA_Belief_Marsh-Density [
      set x-value pxcor
      set y-value pycor
      if random-value < 51
```

```
      [ask patches with [ pxcor = x-value and pycor = (y-value + 1)] [set pcolor 42]]
    if (33 < random-value) and (random-value < 67)
      [ask patches with [ pxcor = x-value and pycor = (y-value - 1)] [set pcolor 42]]
    if random-value > 66 [
      ask patches with [ pxcor = x-value and pycor = (y-value - 1)] [set pcolor 42]
      ask patches with [ pxcor = x-value and pycor = (y-value + 1)] [set pcolor 42]
      ]
   ]
  ]
  ask patches with [ pcolor = sky ] [
   set random-value random 100
   if random-value <= WA_Belief_Marsh-Density [
     set x-value pxcor
     set y-value pycor
     if random-value < 34 [
       ask patches with [ pxcor = x-value and pycor = (y-value + 1)] [set pcolor 42]
     ]
     if (33 < random-value) and (random-value < 67) [
       ask patches with [ pxcor = x-value and pycor = (y-value + 1)] [set pcolor 42]
       ask patches with [ pxcor = x-value and pycor = (y-value - 1)] [set pcolor 42]
     ]
     if random-value > 66 [
       ask patches with [ pxcor = x-value and pycor = (y-value - 1)] [set pcolor 42]
       ask patches with [ pxcor = x-value and pycor = (y-value + 1)] [set pcolor 42]
       ask patches with [ pxcor = (x-value - 1) and pycor = y-value] [set pcolor 42]
       ask patches with [ pxcor = (x-value + 1) and pycor = y-value] [set pcolor 42]
     ]
   ]
  ]
end

to WA_action_setup-river
;  let random-value random 100
;  let x-value min-pxcor
;  let y-value 0
  ask patches [
    if pycor = 0 [set pcolor blue     set WA_Patch_State_Type "Stream"     set WA_Patch_State_Colour blue ]
  ]

;  while [ x-value <= max-pxcor] [
;    ask patch x-value y-value [set pcolor blue]
;    ask patch x-value (y-value - 1) [set pcolor blue]
;
;    set x-value  (x-value + 1)
;    set random-value random 100
;    ifelse random-value < 51
;      [ set y-value (y-value - 1) ]
;      [ set y-value (y-value + 1) ]
;  ]
end

to WA_action_setup-statistical_features
  let m []
  let feature_probability (list m)
  set feature_probability but-first feature_probability

  set m lput orange m
  set m lput WA_Belief_Food-Density m
  set feature_probability lput m feature_probability
  set m []

  set m lput green m
  set m lput WA_Belief_Vegetation-Density m
  set feature_probability lput m feature_probability
  set m []

  set m lput grey m
  set m lput WA_Belief_Rock-Density m
  set feature_probability lput m feature_probability
  set m []

  set m lput sky m
  set m lput WA_Belief_Water-Density m
  set feature_probability lput m feature_probability
  set m []

  set WA_Belief_Open-Terrain (100 - WA_Belief_Food-Density - WA_Belief_Vegetation-Density - WA_Belief_Rock-Density - WA_Belief_Water-Density)
  set m lput black m
  set m lput WA_Belief_Open-Terrain m
  set feature_probability lput m feature_probability
  set m []

  ask patches [
    set pcolor first rnd:weighted-one-of-list feature_probability [ [p] -> last p]
    WA_Patch_brf_instantiate
  ]
end

to WA_action_setup-rock_features
  let random-value random 100
  let x-value min-pxcor
  let y-value min-pycor
```

```
  while [ (x-value <= max-pxcor) ] [
    set y-value min-pycor
    while [ (y-value <= max-pycor)] [
      set random-value random 100
      if random-value <= WA_Belief_Rock-Density [
        set random-value random 100
        ask patches with [ pxcor = x-value and pycor = y-value] [set pcolor grey]
        if (0 < random-value) and (random-value < 26) [
          ask patches with [ pxcor = x-value and pycor = (y-value + 1)] [set pcolor grey]
          ask patches with [ pxcor = x-value and pycor = (y-value - 1)] [set pcolor grey]
        ]
        if (25 < random-value) and (random-value < 51) [
          ask patches with [ pxcor = (x-value - 1) and pycor = y-value] [set pcolor grey]
          ask patches with [ pxcor = (x-value + 1) and pycor = y-value] [set pcolor grey]
        ]
        if (50 < random-value) and (random-value < 76) [
          ask patches with [ pxcor = x-value and pycor = (y-value + 1)] [set pcolor grey]
          ask patches with [ pxcor = x-value and pycor = (y-value - 1)] [set pcolor grey]
          ask patches with [ pxcor = (x-value - 1) and pycor = y-value] [set pcolor grey]
          ask patches with [ pxcor = (x-value + 1) and pycor = y-value] [set pcolor grey]
        ]
        if (75 < random-value) and (random-value < 101) [
          ask patches with [ pxcor = x-value and pycor = (y-value - 1)] [set pcolor grey]
          ask patches with [ pxcor = x-value and pycor = (y-value + 1)] [set pcolor grey]
          ask patches with [ pxcor = (x-value - 1) and pycor = y-value] [set pcolor grey]
          ask patches with [ pxcor = (x-value + 1) and pycor = y-value] [set pcolor grey]
          ask patches with [ pxcor = (x-value - 1) and pycor = y-value - 1] [set pcolor grey]
          ask patches with [ pxcor = (x-value + 1) and pycor = y-value - 1] [set pcolor grey]
          ask patches with [ pxcor = (x-value - 1) and pycor = y-value + 1] [set pcolor grey]
          ask patches with [ pxcor = (x-value + 1) and pycor = y-value + 1] [set pcolor grey]
        ]
      ]
      set y-value (y-value + 4)
    ]
    set x-value  (x-value + 4)
  ]
end

to WA_action_setup-vegetation
  let random-value random 100
  let x-value min-pxcor
  let y-value min-pycor

  while [ (x-value <= max-pxcor) ] [
    set y-value min-pycor
    while [ (y-value <= max-pycor)] [
      set random-value random 100
      if random-value <= WA_Belief_Vegetation-Density [
        ask patches with [ pxcor = x-value and pycor = y-value] [set pcolor green]
        set random-value random 100
        if (19 < random-value) and (random-value < 36) [
          ask patches with [ pxcor = x-value and pycor = (y-value + 1)] [set pcolor green]
          ask patches with [ pxcor = x-value and pycor = (y-value - 1)] [set pcolor green]
        ]
        if (35 < random-value) and (random-value < 51) [
          ask patches with [ pxcor = (x-value - 1) and pycor = y-value] [set pcolor green]
          ask patches with [ pxcor = (x-value + 1) and pycor = y-value] [set pcolor green]
        ]
        if (50 < random-value) and (random-value < 76) [
          ask patches with [ pxcor = x-value and pycor = (y-value + 1)] [set pcolor green]
          ask patches with [ pxcor = x-value and pycor = (y-value - 1)] [set pcolor green]
          ask patches with [ pxcor = (x-value - 1) and pycor = y-value] [set pcolor green]
          ask patches with [ pxcor = (x-value + 1) and pycor = y-value] [set pcolor green]
        ]
        if (75 < random-value) and (random-value < 101) [
          ask patches with [ pxcor = x-value and pycor = (y-value - 1)] [set pcolor green]
          ask patches with [ pxcor = x-value and pycor = (y-value + 1)] [set pcolor green]
          ask patches with [ pxcor = (x-value - 1) and pycor = y-value] [set pcolor green]
          ask patches with [ pxcor = (x-value + 1) and pycor = y-value] [set pcolor green]
          ask patches with [ pxcor = (x-value - 1) and pycor = y-value - 1] [set pcolor green]
          ask patches with [ pxcor = (x-value + 1) and pycor = y-value - 1] [set pcolor green]
          ask patches with [ pxcor = (x-value - 1) and pycor = y-value + 1] [set pcolor green]
          ask patches with [ pxcor = (x-value + 1) and pycor = y-value + 1] [set pcolor green]
        ]
      ]
      set y-value (y-value + 5)
    ]
    set x-value  (x-value + 5)
  ]
end

to WA_action_setup-water
  let random-value random 100
  let x-value min-pxcor
  let y-value min-pycor

  while [ (x-value <= max-pxcor) ] [
    set y-value min-pycor
    while [ (y-value <= max-pycor)] [
      set random-value random 100
      if random-value <= WA_Belief_Water-Density [
        ask patches with [ pxcor = x-value and pycor = y-value] [set pcolor sky]
      ]
```

```
    set y-value (y-value + 3)
   ]
   set x-value  (x-value + 3)
  ]
end

to-report WA_action_tickCount
  report ticks
end

;;;;;;;;;;;;;;; Belief Revision Function (BRF) ;;;;;;;;;;;;;;;;;

;--Beliefs--;

;  WA_Belief_Setup-State              Belief: True/False - If true then the World Agent has been reset. Not Used Because No Tick to Associate Cycle...Look to Delete.
;  WA_Belief_Food-Density             Belief: The World Agent's density of food sources (apple trees, etc).
;  WA_Belief_Vegetation-Density       Belief: The World Agent's density of vegetation.
;  WA_Belief_Rock-Density             Belief: The World Agent's density of rock features.
;  WA_Belief_Water-Density            Belief: The World Agent's density of water (ponds, lakes, etc).
;  WA_Belief_Marsh-Density            Belief: The World Agent's density of marshland.
;  WA_Belief_Open-Terrain             Belief: The World Agent's density of open terrain. This value is calculated and is approximate.
;  WA_Belief_Wind-Direction           Belief: The direction of the wind.
;  WA_Belief_Wind-Strength            Belief: The strength of the wind.
;  WA_Belief_Rain-Frequency           Belief: The frequency of rain.
;  WA_Belief_Rain-Intensity           Belief: The intensity of rain.
;  WA_Belief_Prey-Adult_Number        Belief: Number of Prey.
;  WA_Belief_Hunters-Alpha_Number     Belief: Number of Alpha Hunters.
;  WA_Belief_Hunters-Beta_Number      Belief: Number of Beta Hunters.
;  WA_Belief_Hunters-Omega_Number     Belief: Number of Omega Hunters.
;  WA_State_Sounds                    State: List that holds the sounds that are in the environment during that cycle.
;  WA_Belief_HA-Type                  Belief: The type of hunters.

;--BRF--;
; Nomenclature: WA_perceive_subject_verb (refactoring still needs to be completed)

to WA_brf_hunters-instantiate; This is designed as a driving function and is utilized for testing
  set WA_Belief_HA-Alpha_Quantity #Level1Hunters
  WA_action_hunters-setup
end

to WA_brf_hunters-die; This is designed as a driving function and is utilized for testing
  WA_action_hunters-die
end

to WA_brf_prey-instantiate; This is designed as a driving function and is utilized for testing
  set WA_Belief_Prey-Adult_Number Prey-Adult
  WA_action_prey-setup
end

to WA_brf_prey-die; This is designed as a driving function and is utilized for testing
  WA_action_prey-die
end

to WA_brf_setup
  set WA_Belief_Food-Density Food-Density
  set WA_Belief_Vegetation-Density Vegetation-Density
  set WA_Belief_Rock-Density Rock-Density
  set WA_Belief_Water-Density Water-Density
  set WA_Belief_Marsh-Density Marsh-Density

  set WA_Belief_Wind-Direction Wind-Direction
  set WA_Belief_Wind-Strength Wind-Strength

  set WA_Belief_Prey-Adult_Number Prey-Adult
  set WA_Belief_HA-Type Hunter-Type
  set WA_Belief_HA-Alpha_Quantity #Level1Hunters
  set WA_Belief_HA-Beta_Quantity #Level2Hunters
  set WA_Belief_HA-Omega_Quantity #Level3Hunters
  set WA_Belief_HA-Algorithm_Test Algorithm-Test

  WA_brf_sounds-initialize

end

to WA_brf_setup-food; This is designed as a driving function and is utilized for testing
  set WA_Belief_Food-Density Food-Density
  WA_action_setup-food
end

to WA_brf_setup-vegetation; This is designed as a driving function and is utilized for testing
  set WA_Belief_Vegetation-Density Vegetation-Density
  WA_action_setup-vegetation
end

to WA_brf_setup-rock_features; This is designed as a driving function and is utilized for testing
  set WA_Belief_Rock-Density Rock-Density
  WA_action_setup-rock_features
end

to WA_brf_setup-water; This is designed as a driving function and is utilized for testing
  set WA_Belief_Water-Density Water-Density
  WA_action_setup-water
end
```

```
to WA_brf_setup-river; This is designed as a driving function and is utilized for testing
  WA_action_setup-river
end

to WA_brf_setup-marsh; This is designed as a driving function and is utilized for testing
  set WA_Belief_Marsh-Density Marsh-Density
  WA_action_setup-marsh
end

to WA_brf_sounds-initialize
  WA_brf_sounds-current-initialize
  WA_brf_sounds-previous-initialize
end

to WA_brf_sounds-current-initialize
   set WA_State_Sounds-Current_Cycle []

  let m []
  set m lput 0 m                    ; sender's agent number
  set m lput "name-curr-init" m     ; sender's label
  set m lput 0 m                    ; xcor of sender
  set m lput 0 m                    ; ycor of sender
  set m lput 0 m                    ; sound-level of sender  ********this does not seem to be updating
  set m lput "Curr-Initialize" m        ; sender's message

  set WA_State_Sounds-Current_Cycle lput m WA_State_Sounds-Current_Cycle
;  show "sounds-current-initialize"
end

to WA_brf_sounds-previous-initialize

  set WA_State_Sounds-Previous_Cycle []

  let m []
  set m lput 0 m                   ; sender's agent number
  set m lput "name-prev-init" m    ; sender's label
  set m lput 0 m                   ; xcor of sender
  set m lput 0 m                   ; ycor of sender
  set m lput 0 m                   ; sound-level of sender
  set m lput "Prev-Initialize" m       ; sender's message

  set WA_State_Sounds-Previous_Cycle lput m WA_State_Sounds-Previous_Cycle
;  show "sounds-previous-initialize"
end

to WA_brf_sounds-prey_found

  let m []
  set m lput [who] of self m
  set m lput [label] of self m
  set m lput [xcor] of self m
  set m lput [ycor] of self m
  set m lput [GA_State_Voice-Level] of self m
  set m lput "WA_brf_sounds-prey_found-1" m

  set WA_State_Sounds-Current_Cycle lput m WA_State_Sounds-Current_Cycle
;  show "WA_brf_sounds-prey_found"

;  foreach WA_State_Sounds-Current_Cycle [ [n] ->
;    let sound-level (item 3 n)
;    let message (item 4 n)
;    show [(word "("message", "sound-level ")")] of self
;  ]
end

to WA_brf_sounds-update_Current-to-Previous
  set WA_State_Sounds-Previous_Cycle []
    let n []
;    let sender 0
;    let name "name"
;    let sound-x 0
;    let sound-y 0
;    let sound-level 0
;    let message "W"

  foreach WA_State_Sounds-Current_Cycle [ [m] ->
    set n[]
    set n lput (item 0 m) n       ; sender's agent number
    set n lput (item 1 m) n       ; sender's label
    set n lput (item 2 m) n       ; sender's xcor
    set n lput (item 3 m) n       ; sender's ycor
    set n lput (item 4 m) n       ; sender's sound level
    set n lput (item 5 m) n       ; sender's message

    set WA_State_Sounds-Previous_Cycle lput n WA_State_Sounds-Previous_Cycle
  ]

;  foreach WA_State_Sounds-Previous_Cycle [ [o] ->
;    let sound-level1 (item 3 o)
;    let message1 (item 4 o)
;    show ( word message1 "test" sound-level1)
;  ]
```

```
  set WA_State_Sounds-Current_Cycle [] ; clears the list
;  show "WA_brf_sounds-update_Current-to-Previous" ; [(word "brf_sounds - ("message")")] of self
;  set WA_State_Sounds-Current_Cycle but-first WA_State_Sounds-Current_Cycle
end

;;;;;;;;;;;;;;;; Deliberation ;;;;;;;;;;;;;;;;
; Because the World Agent cannot automatically react to new perceptions, the BRF must be forced in the Deliberation process
; to ensure that the Beliefs are updated prior to the Desires, Plans, and Intentions are re-examined.
; Nomenclature: WA_perceive_verb_subject (refactoring still needs to be completed)

to WA_deliberate_clear
  WA_irf_clear
end

to WA_deliberate_clear-ticks
  WA_irf_clear-ticks
end

to WA_deliberate_cycle
  ;; stop if the lead turtle is about to go out of the world
  ;if [patch-at 1 0] of hunters = nobody [ stop ]
 ; if ((remainder ticks 5) = 0) [(set ROx posx) (set ROy posy)]
  tick

  WA_Patch_deliberate_cycle-start

  ask hunted [
    PA_deliberate_cycle
  ]

  WA_Patch_deliberate_cycle-middle

  ask hunters [
    if WA_Belief_HA-Type = "Lionesses" [
      if HA_State_Hunter-Class = "Alpha" [
        HA_deliberate_lioness-alpha
      ]
      if HA_State_Hunter-Class = "Beta" [
        HA_deliberate_lioness-beta
      ]
    ]

    if WA_Belief_HA-Type = "Wolves" [
      if HA_State_Hunter-Class = "Alpha" [
        HA_deliberate_wolf-alpha
      ]
      if HA_State_Hunter-Class = "Beta" [
        HA_deliberate_wolf-beta
      ]
      if HA_State_Hunter-Class = "Omega" [
        HA_deliberate_wolf-omega
      ]
    ]
  ]

  WA_Patch_deliberate_cycle-end

  WA_brf_sounds-update_Current-to-Previous
  ask turtle 0 [show[(word "("pxcor ", " pycor")")] of self]
  ask turtle 1 [show[(word "("pxcor ", " pycor")")] of self]
  ask turtle 2 [show[(word "("pxcor ", " pycor")")] of self]
;  ask turtle 3 [show[(word "("pxcor ", " pycor")")] of self]
;  ask turtle 4 [show[(word "("pxcor ", " pycor")")] of self]
  show "***End of Turn"
end

to WA_deliberate_setup ; executed by clicking the Setup/Reset button
  WA_irf_clear
  WA_brf_setup
  WA_irf_setup
end

;;;;;;;;;;;;;;;; Desires ;;;;;;;;;;;;;;;;
; The World Agent has no desires of its own.

;;;;;;;;;;;;;;;; Desire Review Funtion (DRF) ;;;;;;;;;;;;;;;;
; The World Agent has no desires of its own.

;;;;;;;;;;;;;;;; Intention Review Function (IRF) ;;;;;;;;;;;;
; Nomenclature: WA_irf_verb_subject (refactoring still needs to be completed)

;--Intentions--;
; The World Agent has no state level intentions of its own.

;--IRF--;
to WA_irf_clear
  WA_plan_clear
end

to WA_irf_clear-ticks
  WA_action_clear-ticks
end
```

```
to WA_irf_setup
  WA_plan_setup
  WA_plan_setup-hunting_grounds
  WA_plan_hunters-setup
  WA_plan_prey-setup
end

;;;;;;;;;;;;; Plans ;;;;;;;;;;;;
; Nomenclature: WA_perceive_subject_verb (refactoring still needs to be completed)

to WA_plan_clear
  WA_action_clear-all
end

to WA_plan_setup
  WA_action_setup-axes
end

to WA_plan_setup-hunting_grounds ; The hunting ground is set up according to the inputs of the various slider buttons
  WA_action_setup-statistical_features
  WA_action_setup-river
  WA_action_setup-marsh
end

to WA_plan_prey-setup
  WA_action_prey-setup
end

to WA_plan_hunters-setup
  WA_action_hunters-setup
end

;;;;;;;;;;;;; Perceptions ;;;;;;;;;;
; This function consists of the selection boxes on the Interface GUI.
; With NetLogo, these objects broker variables which are handled by the BRF Function through a deliberation cycle.
; They are documented here for the purpose of completeness.
; Also, several procedures are coded here as they represent a forcing function which is activated through perception.
; Their primary function is for testing purposes.
; Nomenclature: WA_perceive_subject_verb (refactoring still needs to be completed)

; Vegetation-Density     Slider from 1-100 used to set the density of vegetation.
; Rock-Density           Slider from 1-25 used to set the density of rock features.
; Water-Density          Slider from 1-25 used to set the density of water sources.
; Marsh-Density          Slider from 1-100 used to set the density of marsh lands.
; Wind-Direction         Chooser (Random, North, North-East, East, South-East, South, South-West, West, North-West) used to select the wind direction.
; Wind-Strength          Slider from 1-10 used to set the strength of the wind.
; Rain-Frequency
; Rain-Intensity

to WA_perception_hunters-clear
  WA_brf_hunters-die
end

to WA_perception_hunters-setup
  WA_brf_hunters-instantiate
end

to WA_perception_prey-clear
  WA_brf_prey-die
end

to WA_perception_prey-setup
  WA_brf_prey-instantiate
end

to WA_perception_setup-food
  WA_brf_setup-food
end

to WA_perception_setup-vegetation
  WA_brf_setup-vegetation
end

to WA_perception_setup-rock_features
  WA_brf_setup-rock_features
end

to WA_perception_setup-water
  WA_brf_setup-water
end

to WA_perception_setup-river
  WA_brf_setup-river
end

to WA_perception_setup-marsh
  WA_brf_setup-marsh
end

to WA_perception_sounds-prey_found
  WA_brf_sounds-prey_found
end
```

```
;;----------------------;;
;;;;;;; Patch Agent ;;;;;;;
;;----------------------;;

;;;;;;;;;;; Belief Review Function (BRF) ;;;;;;;;;;;;;

; WA_Patch_State-Scent_Type       State: This variable holds the scent type that is on the patch. The types are neutral, hunter, and prey.
; WA_Patch_State-Scent_Level      State: This variable holds the scent level on the patch.
; WA_Patch_State-Grow_Time        State: This variable holds the growth rate of the vegetation patches.
; WA_Patch_State-Colour           State: This variable holds the patch's color when it was first instantiated.
; WA_Patch_State_Colour-Temp      State: This variable holds the patch's temporary color while it regrows.
; WA_Patch_State-Type             State: This variable holds the patch's type which is rock, vegetation, food, water, etc.
; WA_Patch_State-Sounds           State: This list holds the sounds (sound waves) that can be detected on the patch.
;                                   - Sound Level 0    No sound
;                                   - Sound Level 1    Whisper, only heard on the patch from which it originates and the adjoining patches.
;                                   - Sound Level 2    Low Voice, only heard within 10 patch radius.
;                                   - Sound Level 3    Normal Voice, heard within 50 patch radius.
;                                   - Sound Level 4    Loud Voice, heard within 100 patch radius.
;                                   - Sound Level 5    Yelling / barking, heard within 150 patch radius.
;                                   - Sound Level 6    Howling, heard within 200 patch radius.
; WA_Patch_State_Agent-Number     State: This variable holds the unique turtle number of the PA or HA currently on the patch agent.
; WA_Patch_State_Agent-Heading    State: This variable holds the heading of the PA or HA currently on the patch agent.
; WA_Patch_State_Agent-Speed      State: This variable holds the speed of the PA or HA currently on the patch agent.

;;;;;;;;;;; Instantiation ;;;;;;;;;;;

to WA_Patch_brf_instantiate
  if pcolor = orange [ set WA_Patch_State_Type "Food"         set WA_Patch_State_Colour orange    set WA_Patch_State_Colour-Temp  "None"]
  if pcolor = green  [ set WA_Patch_State_Type "Vegetation"   set WA_Patch_State_Colour green     set WA_Patch_State_Colour-Temp  "None"]
  if pcolor = grey   [ set WA_Patch_State_Type "Rock"         set WA_Patch_State_Colour grey      set WA_Patch_State_Colour-Temp  "None"]
  if pcolor = sky    [ set WA_Patch_State_Type "Water"        set WA_Patch_State_Colour sky       set WA_Patch_State_Colour-Temp  "None"]
  if pcolor = black  [ set WA_Patch_State_Type "Field"        set WA_Patch_State_Colour black     set WA_Patch_State_Colour-Temp  "None"]
end

;;;;;;;;;;; BRFs ;;;;;;;;;;;

to WA_Patch_brf_scent_fade

  ask patches [
    if (WA_Patch_State_Scent-Type = "prey") [
      ;show [(word "("pxcor ", " pycor" , "WA_Patch_State-Scent_Level")")] of self

      ifelse (WA_Patch_State_Scent-Level = 0) [
        set WA_Patch_State_Scent-Type 0
;       if pcolor = yellow [ set pcolor black]
      ]
      [set WA_Patch_State_Scent-Level (WA_Patch_State_Scent-Level - 1)]
    ]
  ]
end

to WA_Patch_brf_scent_initialize
  ask patch-here [
;    if (pcolor = black) [ set pcolor yellow ]
    set WA_Patch_State_Scent-Type [GA_State_Scent-Type] of myself
    set WA_Patch_State_Scent-Level [GA_State_Scent-Level] of myself
  ]
end

to WA_Patch_brf_food-eat
  ask patch-here [
    set pcolor lime
    set WA_Patch_State_Colour-Temp lime
    set WA_Patch_State_Grow-Time 350
  ]
end

to WA_Patch_brf_food-grow
  ask patches [
    if (WA_Patch_State_Colour-Temp = lime) [
      ifelse (WA_Patch_State_Grow-Time = 0) [
        set pcolor orange
        set WA_Patch_State_Colour-Temp "None"
      ][
        set WA_Patch_State_Grow-Time (WA_Patch_State_Grow-Time - 1)
      ]
    ]
  ]
end

;to WA_Patch_brf_HA-end
;  foreach WA_State_Patches-List [ [m] ->
;    ask patch (item 0 m) (item 1 m) [
;      if (pcolor = red) or (pcolor = 13) or (pcolor = 13) [ set pcolor WA_Patch_State_Colour ]
;    ]
;  ]
;end

to WA_Patch_brf_HA-start
  set WA_State_Patches-List []
  ask hunters [
    let n []
```

```
    ask patch-here [
      if pcolor != 19 [
        ;set WA_Patch_State_Colour-Temp pcolor
        set pcolor [color] of myself
;       show pcolor
        set WA_Patch_State_Agent-Number [who] of myself
        set WA_Patch_State_Agent-Heading [heading] of myself
        set WA_Patch_State_Agent-Speed [ GA_State_Speed-Current ] of myself

        set n lput pxcor n
        set n lput pycor n
        set n lput WA_Patch_State_Agent-Number n
        set n lput WA_Patch_State_Agent-Heading n
        set n lput WA_Patch_State_Agent-Speed n
        ask myself [set WA_State_Patches-List lput n WA_State_Patches-List]
      ]
    ]
  ]
end

to WA_Patch_brf_kill_prey
  ask patch-here [
;     set WA_Patch_State_Colour 19
     set pcolor 19
     show "HA and PA on same Patch"
  ]
end

;to WA_Patch_brf_PA-end
;  foreach WA_State_Patches-List [ [m] ->
;     ask patch (item 0 m) (item 1 m) [
;       if (pcolor = brown) [ set pcolor WA_Patch_State_Colour ]
;    ]
;  ]
;end

to WA_Patch_brf_PA/HA-end
  foreach WA_State_Patches-List [ [m] ->
    ask patch (item 0 m) (item 1 m) [
      if (pcolor = red) [
        set pcolor WA_Patch_State_Colour
        if (WA_Patch_State_Colour-Temp = lime) [set pcolor WA_Patch_State_Colour-Temp]
      ]

      if (pcolor = 13) [
        set pcolor WA_Patch_State_Colour
        if (WA_Patch_State_Colour-Temp = lime) [set pcolor WA_Patch_State_Colour-Temp]
      ]

      if (pcolor = 14) [
        set pcolor WA_Patch_State_Colour
        if (WA_Patch_State_Colour-Temp = lime) [set pcolor WA_Patch_State_Colour-Temp]
      ]

      if (pcolor = 16) [
        set pcolor WA_Patch_State_Colour
        if (WA_Patch_State_Colour-Temp = lime) [set pcolor WA_Patch_State_Colour-Temp]
      ]

      if (pcolor = 19) [
        set pcolor WA_Patch_State_Colour
        if (WA_Patch_State_Colour-Temp = lime) [set pcolor WA_Patch_State_Colour-Temp]
      ]

      if (pcolor = brown) [
        set pcolor WA_Patch_State_Colour
        if (WA_Patch_State_Colour-Temp = lime) [set pcolor WA_Patch_State_Colour-Temp]
      ]

;       if (pcolor = red) or (pcolor = 13) or (pcolor = 14) or (pcolor = 16) or (pcolor = 19) or (pcolor = brown) [
;        ifelse (WA_Patch_State_Colour = lime) [
;          set pcolor WA_Patch_State_Colour-Temp
;       ][
;         set pcolor WA_Patch_State_Colour
;       ]
      ;]
    ]
  ]
end

to WA_Patch_brf_PA-start
;  set WA_State_Patches-List []
  ask hunted [
    let n []
    ask patch-here [
      if pcolor != 19 [
        ;set WA_Patch_State_Colour pcolor
        set pcolor [color] of myself
        show pcolor
        set WA_Patch_State_Agent-Number [who] of myself
        set WA_Patch_State_Agent-Heading [heading] of myself
        set WA_Patch_State_Agent-Speed [ GA_State_Speed-Current ] of myself
```

```
        set n lput pxcor n
        set n lput pycor n
        set n lput WA_Patch_State_Agent-Number n
        set n lput WA_Patch_State_Agent-Heading n
        set n lput WA_Patch_State_Agent-Speed n
        ask myself [set WA_State_Patches-List lput n WA_State_Patches-List]
      ]
    ]
  ]
end

to WA_Patch_brf_sounds
  let sender 0
  let name "name"
  let sound-x 0
  let sound-y 0
  let sound-level 0
  let message "none"
  let sound-distance 0

  let n []

  ask turtles [
    ask patch-here [
      set WA_Patch_State_Sounds []
      set n []
      foreach WA_State_Sounds-Previous_Cycle [ [m] ->
        set sender (item 0 m)
        set name (item 1 m)
        set sound-x (item 2 m)
        set sound-y (item 3 m)
        set sound-level (item 4 m)
        set message (item 5 m)

;           show [(word "brf_sounds - ("sound-level")")] of self

        set sound-distance (sqrt( ((pxcor - sound-x) ^ 2) + ((pycor - sound-y) ^ 2) ))
;       show [(word "sender - ("sender") and sound-distance - ("sound-distance") and initial sound-level - ("sound-level")")] of self

        if ((sound-distance >= 0) and (sound-distance <= 2)) and sound-level >= 1 [
          set n lput sender n
          set n lput name n
          set n lput sound-x n
          set n lput sound-y n
          set n lput sound-level n
          set n lput message n

          set WA_Patch_State_Sounds lput n WA_Patch_State_Sounds
;           show [(word "patch sound-level - ("sound-level")")] of self
        ]

        if ((sound-distance > 2) and ( sound-distance <= 5)) and sound-level >= 2 [ ;((sound-level > 1) and (sound-level <= 2)) [
          set sound-level (sound-level - 1)

          set n lput sender n
          set n lput name n
          set n lput sound-x n
          set n lput sound-y n
          set n lput sound-level n
          set n lput message n

          set WA_Patch_State_Sounds lput n WA_Patch_State_Sounds
;           show [(word "patch sound-level - ("sound-level")")] of self
        ]

        if ((sound-distance > 5) and ( sound-distance <= 10)) and sound-level >= 3 [ ;((sound-level > 2) and (sound-level <= 3)) [
          set sound-level (sound-level - 2)

          set n lput sender n
          set n lput name n
          set n lput sound-x n
          set n lput sound-y n
          set n lput sound-level n
          set n lput message n

          set WA_Patch_State_Sounds lput n WA_Patch_State_Sounds
          show [(word "patch sound-level - ("sound-level")")] of self
        ]

        if ((sound-distance > 10) and ( sound-distance <= 50)) and sound-level >= 4 [ ;((sound-level > 3) and (sound-level <= 4)) [
          set sound-level (sound-level - 3)

          set n lput sender n
          set n lput name n
          set n lput sound-x n
          set n lput sound-y n
          set n lput sound-level n
          set n lput message n

          set WA_Patch_State_Sounds lput n WA_Patch_State_Sounds
;           show [(word "patch sound-level - ("sound-level")")] of self
        ]
```

```
      if ((sound-distance > 50) and ( sound-distance <= 100)) and sound-level >= 5 [ ;((sound-level > 4) and (sound-level <= 5)) [
        set sound-level (sound-level - 4)

        set n lput sender n
        set n lput name n
        set n lput sound-x n
        set n lput sound-y n
        set n lput sound-level n
        set n lput message n

        set WA_Patch_State_Sounds lput n WA_Patch_State_Sounds
;         show [(word "patch sound-level - ("sound-level")")] of self
      ]

      if ((sound-distance > 100) and ( sound-distance <= 150)) and sound-level >= 6 [ ;((sound-level > 5) and (sound-level <= 6)) [
        set sound-level (sound-level - 5)

        set n lput sender n
        set n lput name n
        set n lput sound-x n
        set n lput sound-y n
        set n lput sound-level n
        set n lput message n

        set WA_Patch_State_Sounds lput n WA_Patch_State_Sounds
;         show [(word "patch sound-level - ("sound-level")")] of self
      ]

      if ((sound-distance > 150) and ( sound-distance <= 200)) and sound-level >= 7 [ ;((sound-level > 6) and (sound-level <= 7)) [
        set sound-level (sound-level - 6)

        set n lput sender n
        set n lput name n
        set n lput sound-x n
        set n lput sound-y n
        set n lput sound-level n
        set n lput message n

        set WA_Patch_State_Sounds lput n WA_Patch_State_Sounds
         show [(word "patch sound-level - ("sound-level")")] of self
      ]

      if ((sound-distance > 200) and ( sound-distance <= 250)) and sound-level >= 8 [ ;((sound-level > 7) and (sound-level <= 8)) [
        set sound-level (sound-level - 7)

        set n lput sender n
        set n lput name n
        set n lput sound-x n
        set n lput sound-y n
        set n lput sound-level n
        set n lput message n

        set WA_Patch_State_Sounds lput n WA_Patch_State_Sounds
;         show [(word "patch sound-level - ("sound-level")")] of self
      ]
    ]
;         foreach WA_Patch_State_Sounds [ [o] ->
;           set message (item 4 o )
;           show [(word "Patch_State_Sounds - ("message")")] of self
;         ]

  ]
 ]

end

;;;;;;;;;;; Deliberate ;;;;;;;;;;;

to WA_Patch_deliberate_cycle-end
  WA_Patch_brf_PA/HA-end
  ;WA_Patch_brf_HA-end
  ;WA_Patch_brf_PA-end
end

to WA_Patch_deliberate_cycle-middle

  WA_Patch_brf_PA-start
end

to WA_Patch_deliberate_cycle-start
  WA_Patch_brf_sounds
  WA_Patch_brf_HA-start
  WA_Patch_brf_scent_fade
  WA_Patch_brf_food-grow
end

;;;;;;;;;;;;;; Perceptions ;;;;;;;;;;
; Perceptions by a Patch Agent represent a forcing function of an action taken against the patch.

to WA_Patch_perception_food-eaten ; this is a forcing function initiated by a PA eating the food on the patch.
  WA_Patch_brf_food-eat
end
```

```
to WA_Patch_perception-kill_prey
  WA_Patch_brf_kill_prey
end

to WA_Patch_perception_scent-initialize
  WA_Patch_brf_scent_initialize
end

;to WA_Patch_perception_sounds
;  WA_Patch_brf_sounds
;end

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;-----------------------;;;
;;;;;; Generic Agent ;;;;;;
;;;-----------------------;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

;The Generic Agent defines states and procedures that are common to both Hunter and Prey Agents.
;As NetLogo does not yet have Super Classes (and thus Inheritance), the Generic Agent fills this role.
;It is used as a means to re-use code, increasing both coding efficiency and simplicity (in both reading the code and evolving it).

;;;;;;;;; Actions ;;;;;;;;;;;;;
; Nomenclature: GA_action_subject_verb     if the subject is the Agent, then it is omitted.

to GA_action_move-none
  show "Do Nothing"
  set GA_State_Energy-Current (GA_State_Energy-Current - 1)
  set GA_State_Water-Current (GA_State_Water-Current - 1)
  WA_Patch_perception_scent-initialize
end

to GA_action_move-normal
  fd 1
  set GA_State_Energy-Current (GA_State_Energy-Current - 2)
  set GA_State_Water-Current (GA_State_Water-Current - 2)
  WA_Patch_perception_scent-initialize
end

to GA_action_move-quick
  fd 2
  ; fd 1
  ; fd 1
  set GA_State_Energy-Current (GA_State_Energy-Current - 3)
  set GA_State_Water-Current (GA_State_Water-Current - 3)
  WA_Patch_perception_scent-initialize
end

to GA_action_move-fast
  fd 3
  set GA_State_Energy-Current (GA_State_Energy-Current - 4)
  set GA_State_Water-Current (GA_State_Water-Current - 4)
  WA_Patch_perception_scent-initialize
end

to GA_action_move-run
  fd 5
  set GA_State_Energy-Current (GA_State_Energy-Current - 5)
  set GA_State_Water-Current (GA_State_Water-Current - 5)
  WA_Patch_perception_scent-initialize
end

to GA_action_move-sprint
  fd 7
  set GA_State_Energy-Current (GA_State_Energy-Current - 6)
  set GA_State_Water-Current (GA_State_Water-Current - 6)
  set GA_State_Exhaustion 20
  WA_Patch_perception_scent-initialize
end

to GA_action_move-pounce_prey
  move-to patch HA_Belief_Prey-X HA_Belief_Prey-Y
end

to GA_action_turn-around
  let heading-temp heading
  set heading (remainder int ((heading-temp) + 180) 360)
end

;;;;;;;;;;; Belief Review Function (BRF) ;;;;;;;;;;;;;
; The Beliefs of the Generic Agent are detailed under the Beliefs / Turtles-Own section of the World Agent.
; This occurs because NetLogo requires these variables to be declared at the beginning of the code (for reasons unknown).
; Though the class of generic agent variables are not formally declared at this location in the code, they are described below:

;--Beliefs--;
;  GA_Design_Speed-Max                Design: the Agent's maximum possible speed.
;  GA_State_Speed-Current             State: the Agent's current speed.
;  GA_Design_Energy-Max               Design: the maximum Energy that the Agent can store.
;  GA_State_Energy-Current            State: the Agent's current energy reserve.
;  GA_Belief_Hunger-Level             Belief: the Agent's hunger level. The level is dependent on the amount of energy in reserve. The lower the energy reserve, the higher the
                                              hunger level.
;  GA_Design_Hearing-Level-Min        Design: this is the minimum sound level that the Agent is able to perceive (measured at the Agent's location (ie: the Agent's ears)).
;  GA_Design_Vocal-Level-Max          Design: this is the sound level of the Agent's voice (measured at the Agent's location (ie: the Agent's mouth)).
```

```
;  GA_Design_Scent-Level-Min          Design: this is the minimum scent level that the Agent is able to perceive.
;  GA_State_Scent-Type                State: This is the type of scent that the Agent possesses. The types are either prey or hunter.
;  GA_State_Scent-Level               Belief: this is the Agent's scent level. The scent level increases with the Agent's age.
;  GA_Design_Vision-Max               Belief: the maximum distance that the Agent can perceive objects.
;  GA_State_Water-Current             State: the Agent's current water reserve.
;  GA_Belief_Water-Level              Belief: the Agent's thirst level.
;  GA_Belief_Water-Bearing            Belief: the bearing to the nearest water source.
;  GA_Belief_Patch-Colour             Belief: the colour of the patch (WA_Patch_State-Colour) that the Agent is occupying. This variable holds the patch colour at the start of the
                cycle and restores it at the end.
;  GA_Belief_Patch-Type               Belief: the type of patch (WA_Patch_State-Type) that the Agent is occupying.
;  GA_State_Voice-Level               State: This is the voice level of the Agent.
;                                  -  Sound Level 0     No sound
;                                  -  Sound Level 1     Whisper, only heard on the patch from which it originates and the adjoining patches.
;                                  -  Sound Level 2     Low Voice, only heard within 10 patch radius.
;                                  -  Sound Level 3     Normal Voice, heard within 50 patch radius.
;                                  -  Sound Level 4     Loud Voice, heard within 100 patch radius.
;                                  -  Sound Level 5     Yelling / barking, heard within 150 patch radius.
;                                  -  Sound Level 6     Howling, heard within 200 patch radius.
;  GA_State_Exhaustion                State: The Agent is exhausted - True/False.
;  GA_State_Agent-Type                State: The type of Agent: PA / HA.
;  GA_State_Agent-Class               State: The class of Agent: For PA (Adult or Adolescent). For HA (Alpha, Beta, Omega).
;  GA_State_Scan-Distance             Belief: this variable is the distance upon which the eyes focus...representing the control of the nerves upon the retina.
;                                  It is a state because it represents a physical state of the optical reception capability.
;  GA_Perceive_Visual-Scan            Perception: This variable holds the perceived data from the eyes.

;--BRF--;

to GA_brf_cycle
  set GA_Belief_Water false
  set GA_Belief_Water-atSource false
  set GA_Belief_Water-Distance 100
  set GA_State_Exhaustion (GA_State_Exhaustion - 1 )
end

to GA_brf_disposition
  ask patch-here [ ; determines the patch that the Agent is standing on
    ask myself [ set GA_Belief_Patch-Colour WA_Patch_State_Colour     set GA_Belief_Patch-Type WA_Patch_State_Type ]
  ]
end

to GA_brf_hunger-level
  if (0 < GA_State_Energy-Current) and (GA_State_Energy-Current < 25) [set GA_Belief_Hunger-Level 4]
  if (24 < GA_State_Energy-Current) and (GA_State_Energy-Current < 50) [set GA_Belief_Hunger-Level 3]
  if (49 < GA_State_Energy-Current) and (GA_State_Energy-Current < 75) [set GA_Belief_Hunger-Level 2]
  if (74 < GA_State_Energy-Current) and (GA_State_Energy-Current < 100) [set GA_Belief_Hunger-Level 1]
  if (99 < GA_State_Energy-Current) and (GA_State_Energy-Current < 150) [set GA_Belief_Hunger-Level 0]
end

to GA_brf_thirst-level
  if (0 < GA_State_Water-Current) and (GA_State_Water-Current < 25) [set GA_Belief_Water-Level 4]
  if (24 < GA_State_Water-Current) and (GA_State_Water-Current < 50) [set GA_Belief_Water-Level 3]
  if (49 < GA_State_Water-Current) and (GA_State_Water-Current < 75) [set GA_Belief_Water-Level 2]
  if (74 < GA_State_Water-Current) and (GA_State_Water-Current < 100) [set GA_Belief_Water-Level 1]
  if (99 < GA_State_Water-Current) and (GA_State_Water-Current < 151) [set GA_Belief_Water-Level 0]

end

to GA_brf_water
  let temp-heading 0
  let temp-distance 100
  let patch-colour 0

  let n first GA_Perceive_Visual-Scan
  set GA_Belief_Patch-Colour (item 0 n)

  ifelse (GA_Belief_Patch-Colour = blue) or (GA_Belief_Patch-Colour = sky) or (GA_Belief_Patch-Colour = 42)[
;    show "at water"
    set GA_Belief_Water-atSource true ][
    foreach GA_Perceive_Visual-Scan [ [m] ->
       set patch-colour (item 0 m)
       if (patch-colour = blue) or (patch-colour = sky) or (patch-colour = 42) [
;         show "found water"
         (set GA_Belief_Water true)
         (set temp-heading (item 1 m))
         (set temp-distance (item 2 m))

         if (temp-distance < GA_Belief_Water-Distance) [
           set GA_Belief_Water-Distance temp-distance
           set GA_Belief_Water-Heading temp-heading
         ]
       ]
     ]
   ]

 ; show [(word "( "GA_Belief_Water-Heading" , "GA_Belief_Water-Distance" )")] of self

  if (GA_Belief_Water = false)[
    ifelse (ycor >= 0)[set GA_Belief_Water-Heading 180 set GA_Belief_Water true show "water at 180"][set GA_Belief_Water-Heading 0 set GA_Belief_Water true show "water
                at 0"]
  ]

end
```

```
;;;;;;;;;;;;;;;; Deliberation ;;;;;;;;;;;;;;;;

to GA_deliberate_cycle
  GA_brf_hunger-level
  GA_brf_thirst-level
  GA_drf_food
  GA_drf_water
end

;;;;;;;;;;;;; Desire Review Function (DRF) ;;;;;;;;;;;;;;;;

;--Desires--;
;  GA_Desire_Food                  Desire: does the Agent desire food?
;  GA_Desire_Water true              Desire: does the Agent desire water?
;  GA_Desire_Evade                 Desire: Does the Agent desire to evade the situation. States are True or False.

;--DRF--;

to GA_drf_food
  if GA_Belief_Hunger-Level = 4 [set GA_Desire_Food true]
  if GA_Belief_Hunger-Level = 3 [set GA_Desire_Food true]
  if GA_Belief_Hunger-Level = 2 [set GA_Desire_Food true]
  if GA_Belief_Hunger-Level = 1 [set GA_Desire_Food true]
  if GA_Belief_Hunger-Level = 0 [set GA_Desire_Food false]
end

to GA_drf_water
  if GA_Belief_Water-Level = 4 [set GA_Desire_Water true]
  if GA_Belief_Water-Level = 3 [set GA_Desire_Water true]
  if GA_Belief_Water-Level = 2 [set GA_Desire_Water true]
  if GA_Belief_Water-Level = 1 [set GA_Desire_Water true]
  if GA_Belief_Water-Level = 0 [set GA_Desire_Water false]
end

;;;;;;;;;;;;;; Intention Review Function (IRF) ;;;;;;;;;;;;
; Nomenclature: GA_irf_verb_subject

to GA_irf_turn-and-run
  show "run-away!"
  GA_action_turn-around
  GA_action_move-sprint
;  GA_action_turn-around
end

;;;;;;;;;;;;; Perceptions ;;;;;;;;;;

to GA_perceive

end

;;;;;;;;;;;;; Plans ;;;;;;;;;;;;

;to GA_plan_turn-and-run
;  show "run-away!"
;  GA_action_turn-around
;  GA_action_move-sprint
;  GA_action_turn-around
;end

;;----------------------;;
;;;;;; Prey Agent ;;;;;;
;;----------------------;;

; The Prey Agent represents an adult or adolescent prey depending on how it is instantiated.
; The command "breed [ hunted prey]" is defined in the World Agent / Beliefs section as required by NetLogo.

;;;;;;;;; Actions ;;;;;;;;;;;;;
; Nomenclature: PA_action_subject_verb     if the subject is the Agent, then it is omitted.

to PA_action_eat-food
;  show "eating food"
  set GA_State_Energy-Current 150
  WA_Patch_perception_food-eaten
end

to PA_action_drink-water
;  show "drinking water"
  set GA_State_Water-Current 150
end

to PA_action_kill-prey
  WA_Patch_perception-kill_prey
end

;;;;;;;;;;;;; Belief Review Function (BRF) ;;;;;;;;;;;;;;;;

; The Beliefs of the Prey Agent are instantiated under the Beliefs / Hunted-Own section of the World Agent.
; This occurs because NetLogo requires these variables to be declared at the beginning of the code (for reasons unknown).
; Though the class of hunted variables are not formally declared at this location in the code, they are described below:

;--Beliefs--;
;  PA_Belief_Food                Belief: the Agent believes (true/false) that it has perceived food.
;  PA_Belief_Food-Heading          Belief: this is the heading of perceived food.
```

```
;  PA_Belief_Food-Distance          Belief: this is the distance of the food from the Agent.
;  PA_Belief_Food-Sources           Belief: this is the Agent's list of known food sources.
;  PA_Belief_Food-atSource          Belief: true/false, the Agent is geolocated with a food source.

;  PA_Belief_Water                 Belief: the Agent believes (true/false) that it has perceived water.
;  PA_Belief_Water-Heading          Belief: this is the heading of perceived water.
;  PA_Belief_Water-Distance         Belief: this is the distance of the water from the Agent.
;  PA_Belief_Water-Sources          Belief: this is the Agent's list of known water sources.
;  PA_Belief_Water-atSource         Belief: true/false, the Agent is geolocated with a water source.

;  PA_Belief_Hunter                Belief: the Agent believes (true/false) that it has perceived a hunter.
;  PA_Belief_Hunter-Heading         Belief: this is the heading of the perceived hunter.
;  PA_Belief_Hunter-Distance        Belief: this is the distance of the hunter from the Agent.
;  PA_Belief_Hunter-Sees-Me         Belief: the Agent believes (true/false) that it is being observed by the hunter. (Not Used)
;  PA_Belief_Hunter-atSource        Belief: true/false, the Prey Agent is geolocated with a Hunter Agent.

;  PA_Belief_Move-Next_Heading      Belief: this is the heading that the prey prefers to follow on its next movement. (Not Used)
;  PA_Belief_Move-Next_Distance     Belief: this is the number of squares that the Prey can move before it needs to reassess its next steps. (Not Used)

;--Instantiation--;
to PA_belief_instantiate-hunted_prey_adult

  create-hunted 1
    [ set color brown
      set size 7
      set shape "default"
      set pen-size 1

      setxy 0 10 ;random-xcor random-ycor
;      set heading 180
      ;pen-down
;      facexy 0 1

      set label who
      set label-color white
      set heading 180
      set GA_State_Energy-Current 75
      set GA_State_Water-Current 150
      set GA_Design_Speed-Max 5
      set GA_State_Speed-Current 0
      set GA_State_Scent-Type "prey"
      set GA_State_Scent-Level 100
      set GA_State_Exhaustion 0
      set PA_Belief_Hunter-atSource false

      set PA_Belief_Food-Sources []
      set GA_Belief_Water-Sources []
    ]
end

to PA_belief_instantiate-hunted_prey_adolescent
  create-hunted 1
    [ set color orange
      set size 3
      set shape "default"
      set pen-size 1

      setxy 25 25
      facexy 0 1

      set label who
      set label-color black

      set GA_Design_Speed-Max 5
      set GA_State_Speed-Current 0
    ]
end

;--BRF--;
to PA_brf_cycle
  set PA_Belief_Food false
  set PA_Belief_Food-atSource false
  set PA_Belief_Food-Distance 100
  set PA_Belief_Hunter false
  set PA_Belief_Hunter-Distance 100
  GA_brf_cycle

  PA_perception_visual-scan

end

to PA_brf_food
  let temp-heading 0
  let temp-distance 100
  let patch-colour 0

  let n first GA_Perceive_Visual-Scan
  set GA_Belief_Patch-Colour (item 0 n)

  ifelse (GA_Belief_Patch-Colour = orange)[
;    show "at food"
    set PA_Belief_Food-atSource true
  ][
```

```
    foreach GA_Perceive_Visual-Scan [ [m] ->
      set patch-colour (item 0 m)
      if (patch-colour = orange) [
;       show "found food"
        (set PA_Belief_Food true)
        (set temp-heading (item 1 m))
        (set temp-distance (item 2 m))

        if (temp-distance < PA_Belief_Food-Distance) [
          set PA_Belief_Food-Distance temp-distance
          set PA_Belief_Food-Heading temp-heading
        ]
      ]
    ]
  ]

end

to PA_brf_hunter
  let temp-heading 0
  let temp-distance 100
  let patch-colour 0

  let n first GA_Perceive_Visual-Scan
  set GA_Belief_Patch-Colour (item 0 n)
;  show [(word "("GA_Belief_Patch-Colour")")] of self

  ifelse (GA_Belief_Patch-Colour = red)[
    show "I'm being attacked!"
    set PA_Belief_Hunter-atSource true
  ][
    foreach GA_Perceive_Visual-Scan [ [m] ->
      set patch-colour (item 0 m)
      if (patch-colour = red) [
        show "found hunter"
        (set PA_Belief_Hunter true)
        (set temp-heading (item 1 m))
        (set temp-distance (item 2 m))

        if (temp-distance < PA_Belief_Hunter-Distance) [
          set PA_Belief_Hunter-Distance temp-distance
          set PA_Belief_Hunter-Heading temp-heading
;          show [(word "(" PA_Belief_Hunter-Heading ", " PA_Belief_Hunter-Distance")")] of self
        ]
      ]
    ]
  ]

end

;;;;;;;;;;;;;;;; Deliberation ;;;;;;;;;;;;;;;;

to PA_deliberate_cycle

  PA_brf_cycle

  PA_brf_hunter

  PA_drf_hunter

  if PA_Belief_Hunter-atSource = true [
    PA_irf_kill-prey
  ]

  if PA_Belief_Hunter-atSource = false [
    if GA_State_Exhaustion <= 0 [
      ifelse (GA_Desire_Evade = true) [
        PA_irf_evade-turn-and-run
      ][
        GA_deliberate_cycle
        PA_drf_food-or-water
;    show [(word "("GA_Desire_Food " , "GA_Desire_Water " , "PA_Desire_Hunger-Thirst " , "GA_State_Energy-Current " , "GA_State_Water-Current ")")] of self
        let heading-set false

        if (GA_Desire_Water = true) [
;      show "Water"
          GA_brf_water
;      show [(word "( "GA_Belief_Water" )")] of self
          if GA_Belief_Water-atSource = true [PA_irf_drink-water]
            if (GA_Belief_Water = true) and (GA_Belief_Water-atSource = false) [
              PA_irf_move-water
              set heading-set true
            ]
          ]

        if (GA_Desire_Food = true) and (PA_Desire_Hunger-Thirst = "Food")[
;      show "Food"
          PA_brf_food
;      show [(word "( "PA_Belief_Food" )")] of self
          if PA_Belief_Food-atSource = true [PA_irf_eat-food]
          if PA_Belief_Food = true and (PA_Belief_Food-atSource = false)[
            PA_irf_move-food
            set heading-set true
```

```
        ]
      ]

      if ((GA_Desire_Water = true) or (GA_Desire_Food = true) and (PA_Belief_Food = false) and (GA_Belief_Water = false) and (heading-set = false)) [PA_irf_move-
            search_circle]

      if ((GA_Desire_Food = false) and (GA_Desire_Water = false)) [PA_irf_relax]
    ]
  ]
 ]
end

;;;;;;;;;;;;;; Desire Review Function (DRF) ;;;;;;;;;;;;;;;;

;--Desires--;
; PA_Desire_Hunger-Thirst            Desire: Does the Agent desire food or water above the other. States are "Food" or "Water".

to PA_drf_food-or-water
  ifelse (GA_Desire_Food = true) and (GA_Desire_Water = true) [set PA_Desire_Hunger-Thirst "Water"] [ set PA_Desire_Hunger-Thirst "Food" ]
end

to PA_drf_hunter
  ifelse (PA_Belief_Hunter = true) [set GA_Desire_Evade true][set GA_Desire_Evade false]
end

;;;;;;;;;;;;;; Intention Review Function (IRF) ;;;;;;;;;;;;;
; Nomenclature: PA_irf_verb_subject

;--Intentions--;


;--IRF--;
to PA_irf_drink-water
  PA_action_drink-water
end


to PA_irf_eat-food
  PA_action_eat-food
end

to PA_irf_kill-prey
  PA_action_kill-prey
end

to PA_irf_move-food
  set heading PA_Belief_Food-Heading
  GA_action_move-normal
end

to PA_irf_move-search_circle
  set heading (heading + 10)
  GA_action_move-normal
end

to PA_irf_relax
  GA_action_move-none
end

to PA_irf_move-water
  set heading GA_Belief_Water-Heading
  GA_action_move-normal
end

to PA_irf_evade-turn-and-run
  GA_irf_turn-and-run
end

;;;;;;;;;;;;;; Plans ;;;;;;;;;;;;

;to PA_plan_food-moveto
;  set heading PA_Belief_Food-Heading
;  GA_action_move-normal
;end

;to PA_plan_food-water-search
;  set heading (heading + 10)
;  GA_action_move-normal
;end

;to PA_plan_water-moveto
;  set heading GA_Belief_Water-Heading
;  GA_action_move-normal
;end

;;;;;;;;;;;;;; Perceptions ;;;;;;;;;;

;  PA_Perceive_Visual-Scan            Perception: this is a list of perceived objects during a visual scan.

to PA_perception_visual-scan

  set GA_Perceive_Visual-Scan []
```

```
;;;;; Scan Agent's Patch;;;;;

  ask patch-here [ ; determines the patch that the Agent is standing on
    let m []
    set m lput pcolor m
    set m lput 0 m
    set m lput 0 m
    ask myself [set GA_Perceive_Visual-Scan lput m GA_Perceive_Visual-Scan]
  ]

;;;;; Center Scan;;;;;
  let i 1

  while [i <= 12] [
    ask patch-ahead i [ ; determines the patches in front of the Agent
      let h (remainder int ((towards myself) + 180) 360)
      let m []
      set m lput pcolor m
      set m lput h m
      set m lput i m
      ask myself [set GA_Perceive_Visual-Scan lput m GA_Perceive_Visual-Scan]
      if ((pcolor = green) or (pcolor = grey)) [ (set i 13) ]
    ]
    set i (i + 1)
  ]

;;;;; Left Scan;;;;;

  set i 1

  while [i <= 11] [
    ask patch-left-and-ahead 5 i [ ; determines the patches left by 5 degrees in front of the Agent
      let h (remainder int ((towards myself) + 180) 360)
      let m []
      set m lput pcolor m
      set m lput h m
      set m lput i m
      ask myself [set GA_Perceive_Visual-Scan lput m GA_Perceive_Visual-Scan]
      if ((pcolor = green) or (pcolor = grey)) [ (set i 12) ]
    ]
    set i (i + 1)
  ]

  set i 1

  while [i <= 10] [
    ask patch-left-and-ahead 15 i [ ; determines the patches left by 15 degrees in front of the Agent
      let h (remainder int ((towards myself) + 180) 360)
      let m []
      set m lput pcolor m
      set m lput h m
      set m lput i m
      ask myself [set GA_Perceive_Visual-Scan lput m GA_Perceive_Visual-Scan]
      if ((pcolor = green) or (pcolor = grey)) [ (set i 11) ]
    ]
    set i (i + 1)
  ]

  set i 1

  while [i <= 10] [
    ask patch-left-and-ahead 25 i [ ; determines the patches left by 25 degrees in front of the Agent
      let h (remainder int ((towards myself) + 180) 360)
      let m []
      set m lput pcolor m
      set m lput h m
      set m lput i m
      ask myself [set GA_Perceive_Visual-Scan lput m GA_Perceive_Visual-Scan]
      if ((pcolor = green) or (pcolor = grey)) [ (set i 11) ]
    ]
    set i (i + 1)
  ]

  set i 1

  while [i <= 10] [
    ask patch-left-and-ahead 35 i [ ; determines the patches left by 35 degrees in front of the Agent
      let h (remainder int ((towards myself) + 180) 360)
      let m []
      set m lput pcolor m
      set m lput h m
      set m lput i m
      ask myself [set GA_Perceive_Visual-Scan lput m GA_Perceive_Visual-Scan]
      if ((pcolor = green) or (pcolor = grey)) [ (set i 11) ]
    ]
    set i (i + 1)
  ]

  set i 1

  while [i <= 10] [
    ask patch-left-and-ahead 45 i [ ; determines the patches left by 45 degrees in front of the Agent
      let h (remainder int ((towards myself) + 180) 360)
```

```
    let m []
    set m lput pcolor m
    set m lput h m
    set m lput i m
    ask myself [set GA_Perceive_Visual-Scan lput m GA_Perceive_Visual-Scan]
    if ((pcolor = green) or (pcolor = grey)) [ (set i 11) ]
   ]
   set i (i + 1)
  ]

  set i 1

  while [i <= 10] [
   ask patch-left-and-ahead 55 i [ ; determines the patches left by 55 degrees in front of the Agent
    let h (remainder int ((towards myself) + 180) 360)
    let m []
    set m lput pcolor m
    set m lput h m
    set m lput i m
    ask myself [set GA_Perceive_Visual-Scan lput m GA_Perceive_Visual-Scan]
    if ((pcolor = green) or (pcolor = grey)) [ (set i 11) ]
   ]
   set i (i + 1)
  ]

  set i 1

  while [i <= 10] [
   ask patch-left-and-ahead 65 i [ ; determines the patches left by 65 degrees in front of the Agent
    let h (remainder int ((towards myself) + 180) 360)
    let m []
    set m lput pcolor m
    set m lput h m
    set m lput i m
    ask myself [set GA_Perceive_Visual-Scan lput m GA_Perceive_Visual-Scan]
    if ((pcolor = green) or (pcolor = grey)) [ (set i 11) ]
   ]
   set i (i + 1)
  ]

  set i 1

  while [i <= 10] [
   ask patch-left-and-ahead 75 i [ ; determines the patches left by 75 degrees in front of the Agent
    let h (remainder int ((towards myself) + 180) 360)
    let m []
    set m lput pcolor m
    set m lput h m
    set m lput i m
    ask myself [set GA_Perceive_Visual-Scan lput m GA_Perceive_Visual-Scan]
    if ((pcolor = green) or (pcolor = grey)) [ (set i 11) ]
   ]
   set i (i + 1)
  ]

  set i 1

  while [i <= 10] [
   ask patch-left-and-ahead 85 i [ ; determines the patches left by 85 degrees in front of the Agent
    let h (remainder int ((towards myself) + 180) 360)
    let m []
    set m lput pcolor m
    set m lput h m
    set m lput i m
    ask myself [set GA_Perceive_Visual-Scan lput m GA_Perceive_Visual-Scan]
    if ((pcolor = green) or (pcolor = grey)) [ (set i 11) ]
   ]
   set i (i + 1)
  ]

  set i 1

  while [i <= 10] [
   ask patch-left-and-ahead 95 i [ ; determines the patches left by 95 degrees in front of the Agent
    let h (remainder int ((towards myself) + 180) 360)
    let m []
    set m lput pcolor m
    set m lput h m
    set m lput i m
    ask myself [set GA_Perceive_Visual-Scan lput m GA_Perceive_Visual-Scan]
    if ((pcolor = green) or (pcolor = grey)) [ (set i 11) ]
   ]
   set i (i + 1)
  ]

 ;;;;; Right Scan;;;;;

  set i 1

  while [i <= 11] [
   ask patch-right-and-ahead 5 i [ ; determines the patches right by 5 degrees in front of the Agent
    let h (remainder int ((towards myself) + 180) 360)
    let m []
```

```
    set m lput pcolor m
    set m lput h m
    set m lput i m
    ask myself [set GA_Perceive_Visual-Scan lput m GA_Perceive_Visual-Scan]
    if ((pcolor = green) or (pcolor = grey)) [ (set i 12) ]
  ]
  set i (i + 1)
]

set i 1

while [i <= 10] [
  ask patch-right-and-ahead 15 i [ ; determines the patches right by 15 degrees in front of the Agent
    let h (remainder int ((towards myself) + 180) 360)
    let m []
    set m lput pcolor m
    set m lput h m
    set m lput i m
    ask myself [set GA_Perceive_Visual-Scan lput m GA_Perceive_Visual-Scan]
    if ((pcolor = green) or (pcolor = grey)) [ (set i 11) ]
  ]
  set i (i + 1)
]

set i 1

while [i <= 10] [
  ask patch-right-and-ahead 25 i [ ; determines the patches right by 25 degrees in front of the Agent
    let h (remainder int ((towards myself) + 180) 360)
    let m []
    set m lput pcolor m
    set m lput h m
    set m lput i m
    ask myself [set GA_Perceive_Visual-Scan lput m GA_Perceive_Visual-Scan]
    if ((pcolor = green) or (pcolor = grey)) [ (set i 11) ]
  ]
  set i (i + 1)
]

set i 1

while [i <= 10] [
  ask patch-right-and-ahead 35 i [ ; determines the patches right by 35 degrees in front of the Agent
    let h (remainder int ((towards myself) + 180) 360)
    let m []
    set m lput pcolor m
    set m lput h m
    set m lput i m
    ask myself [set GA_Perceive_Visual-Scan lput m GA_Perceive_Visual-Scan]
    if ((pcolor = green) or (pcolor = grey)) [ (set i 11) ]
  ]
  set i (i + 1)
]

set i 1

while [i <= 10] [
  ask patch-right-and-ahead 45 i [ ; determines the patches right by 45 degrees in front of the Agent
    let h (remainder int ((towards myself) + 180) 360)
    let m []
    set m lput pcolor m
    set m lput h m
    set m lput i m
    ask myself [set GA_Perceive_Visual-Scan lput m GA_Perceive_Visual-Scan]
    if ((pcolor = green) or (pcolor = grey)) [ (set i 11) ]
  ]
  set i (i + 1)
]

set i 1

while [i <= 10] [
  ask patch-right-and-ahead 55 i [ ; determines the patches right by 55 degrees in front of the Agent
    let h (remainder int ((towards myself) + 180) 360)
    let m []
    set m lput pcolor m
    set m lput h m
    set m lput i m
    ask myself [set GA_Perceive_Visual-Scan lput m GA_Perceive_Visual-Scan]
    if ((pcolor = green) or (pcolor = grey)) [ (set i 11) ]
  ]
  set i (i + 1)
]

set i 1

while [i <= 10] [
  ask patch-right-and-ahead 65 i [ ; determines the patches right by 65 degrees in front of the Agent
    let h (remainder int ((towards myself) + 180) 360)
    let m []
    set m lput pcolor m
    set m lput h m
    set m lput i m
```

```
    ask myself [set GA_Perceive_Visual-Scan lput m GA_Perceive_Visual-Scan]
    if ((pcolor = green) or (pcolor = grey)) [ (set i 11) ]
  ]
  set i (i + 1)
 ]

 set i 1

 while [i <= 10] [
  ask patch-right-and-ahead 75 i [ ; determines the patches right by 75 degrees in front of the Agent
    let h (remainder int ((towards myself) + 180) 360)
    let m []
    set m lput pcolor m
    set m lput h m
    set m lput i m
    ask myself [set GA_Perceive_Visual-Scan lput m GA_Perceive_Visual-Scan]
    if ((pcolor = green) or (pcolor = grey)) [ (set i 11) ]
  ]
  set i (i + 1)
 ]

 set i 1

 while [i <= 10] [
  ask patch-right-and-ahead 85 i [ ; determines the patches right by 85 degrees in front of the Agent
    let h (remainder int ((towards myself) + 180) 360)
    let m []
    set m lput pcolor m
    set m lput h m
    set m lput i m
    ask myself [set GA_Perceive_Visual-Scan lput m GA_Perceive_Visual-Scan]
    if ((pcolor = green) or (pcolor = grey)) [ (set i 11) ]
  ]
  set i (i + 1)
 ]

 set i 1

 while [i <= 10] [
  ask patch-right-and-ahead 95 i [ ; determines the patches right by 95 degrees in front of the Agent
    let h (remainder int ((towards myself) + 180) 360)
    let m []
    set m lput pcolor m
    set m lput h m
    set m lput i m
    ask myself [set GA_Perceive_Visual-Scan lput m GA_Perceive_Visual-Scan]
    if ((pcolor = green) or (pcolor = grey)) [ (set i 11) ]
  ]
  set i (i + 1)
 ]

end

;;----------------------;;;
;;;;;;  Hunter Agent  ;;;;;;
;;----------------------;;;

; The Hunter Agent represents three classes of hunters:
;     1. Level-1 Hunter:
;     2. Level-2 Hunter:
;     3. Level-3 Hunter
;
; The command "breed [ hunters hunter]" is defined in the World Agent / Beliefs section as required by NetLogo.
;

;;;;;;;;; Actions ;;;;;;;;;;;;;;;

to HA-action_talk-prey_found
  show "HA-action_talk-prey_found"
  WA_perception_sounds-prey_found

end

to HA_action_eat-prey
  ;show "Eating Prey"
  set GA_State_Energy-Current 300
  ;figure out where an Agent dies!!!
;  WA_Patch_perception_food-eaten
end

to HA_action_face-alpha_heading
  set heading HA_Belief_Alpha-Heading
end

to HA_action_face-prey
    facexy HA_Belief_Prey-X HA_Belief_Prey-Y
end

to HA_action_kill-prey
  WA_Patch_perception-kill_prey
end

to HA_action_move-alpha
```

```
  facexy HA_Belief_Alpha-X HA_Belief_Alpha-Y
  GA_action_move-fast
end

to HA_action_move-encircle
  facexy HA_Belief_Encircle-X HA_Belief_Encircle-Y
  let x-dif (HA_Belief_Encircle-X - xcor)
  let y-dif (HA_Belief_Encircle-Y - ycor)

  let d sqrt( ((x-dif) ^ 2) + ((y-dif) ^ 2) )

  ifelse d >= 5 [
    GA_action_move-run
  ][
    HA_action_move-pounce_encircle
  ]
end

to HA_action_move-pounce_encircle

  if label = "Beta1" [
    move-to patch HA_Belief_Beta1-Encircle-X HA_Belief_Beta1-Encircle-Y
  ]

  if label = "Beta2" [
    move-to patch HA_Belief_Beta2-Encircle-X HA_Belief_Beta2-Encircle-Y
  ]

  if label = "Omega1" [
    move-to patch HA_Belief_Encircle-X HA_Belief_Encircle-Y
  ]
end

to HA_action_move-prey_fast
 set heading HA_Belief_Hunter-Prey_Heading
 GA_action_move-fast
end

to HA_action_move-prey_normal
  facexy HA_Belief_Prey-X HA_Belief_Prey-Y
  GA_action_move-normal
end

to HA_action_move-prey_pounce
  set heading HA_Belief_Hunter-Prey_Heading
  GA_action_move-pounce_prey
end

to HA_action-search-alpha
  set heading (heading + 90)
end

to HA_action_search-prey
  set heading (heading + 10)
  GA_action_move-fast
end

;;;;;;;;;;;;;; Belief Revision Function (BRF) ;;;;;;;;;;;;;;;;

; The Beliefs of the Hunter Agent are instantiated under the Beliefs / Hunters-Own section of the World Agent.
; This occurs because NetLogo requires these variables to be declared at the beginning of the code (for reasons unknown).
; Though the class of hunter variables are not formally declared at this location in the code, they are described below. Note that
; each belief exists within a cycle, where the cycles existing for this specific implementation are the Deliberate Cycle (DC), the
; Hunting Cycle (HC), and the Pack Cycle (PC) (where the Pack Cycle is used only to distinguish between the Alpha, Beta, and Omega Hunters).

;--Beliefs--;

;  HA_Belief_Prey-Seen                  Belief (DC): Prey has been visually observed - True/False.
;  HA_Belief_Prey-Known                 Belief (HC): HA has knowledge of the Prey's last known location. - True/False.
;  HA_Belief_Hunter-Prey_Heading        Belief (DC): Heading of HA to the prey.
;  HA_Belief_Hunter-Prey_Distance       Belief (DC): Distance from HA to prey.
;  HA_Belief_Prey-X                     Belief (HC): Prey's x-coord.
;  HA_Belief_Prey-Y                     Belief (HC): Prey's y-coord.
;  HA_Belief_Prey-Heading               Belief (HC): Prey's heading.
;  HA_Belief_Prey-Speed                 Belief (HC): Prey's speed.
;  HA_Belief_Prey-atSource              Belief (DC): The HA and the prey occupy the same patch.
;  HA_Belief_Prey-Sources               Belief (HC): List of all the sources of prey (and their X,y coordinates).
;  HA_Belief_Leader                     Belief (PC): The turtle number of the HA's superior.
;  HA_Belief_Alpha                      Belief (PC): The turtle number of the Alpha.
;  HA_Belief_Alpha-Seen                 Belief (DC): Alpha has been visually observed - True/False.
;  HA_Belief_Alpha-Known                Belief (HC): HA has knowledge of the Alpha's last location. - True/False.
;  HA_Belief_Hunter-Alpha_Heading       Belief (DC): The heading from the HA to the Alpha.
;  HA_Belief_Hunter-Alpha_Distance      Belief (DC): The distance from the HA to the Alpha.
;  HA_Belief_Alpha-X                    Belief (HC): The Alpha's x-coord.
;  HA_Belief_Alpha-Y                    Belief (HC): The Alpha's y-coord.
;  HA_Belief_Alpha-Heading              Belief (HC): Alpha's heading.
;  HA_Belief_Alpha-Speed                Belief (HC): Alpha's speed.
;  HA_Belief_Alpha-Prey_Distance        Belief (DC): The distance between the Alpha and the Prey.
;  HA_Belief_Alpha-Prey_Heading         Belief (DC): The heading from the Alpha and the Prey.
;  HA_Belief_Beta1                      Belief (PC): The turtle number of the Beta1.
;  HA_Belief_Beta1-Seen                 Belief (DC): Beta1 has been visually observed - True/False.
;  HA_Belief_Beta1-Known                Belief (HC): HA has knowledge of the Beta1's last location. - True/False.
;  HA_Belief_Hunter-Beta1_Heading       Belief (DC): The heading from the HA to the Beta1.
```

```
;  HA_Belief_Hunter-Beta1_Distance        Belief (DC): The distance from the HA to the Beta1.
;  HA_Belief_Beta1-X                      Belief (HC): The Beta1's x-coord.
;  HA_Belief_Beta1-Y                      Belief (HC): The Beta1's y-coord.
;  HA_Belief_Beta1-Prey_Distance          Belief (DC): The distance between the Beta1 and the Prey.
;  HA_Belief_Beta1-Prey_Heading           Belief (DC): The heading from the Beta1 and the Prey.
;  HA_Belief_Beta1-Encircle-X             Belief (DC): X-coord of the Beta1 encircle point.
;  HA_Belief_Beta1-Encircle-Y             Belief (DC): Y-coord of the Beta1 encircle point.
;  HA_Belief_Beta2                      Belief (PC): The turtle number of the Beta2.
;  HA_Belief_Beta2-Seen                   Belief (DC): Beta2 has been visually observed - True/False.
;  HA_Belief_Beta2-Known                  Belief (HC): HA has knowledge of the Beta2's last location. - True/False.
;  HA_Belief_Hunter-Beta2_Heading         Belief (DC): The heading from the HA to the Beta2.
;  HA_Belief_Hunter-Beta2_Distance        Belief (DC): The distance from the HA to the Beta2.
;  HA_Belief_Beta2-X                      Belief (HC): The Beta2's x-coord.
;  HA_Belief_Beta2-Y                      Belief (HC): The Beta2's y-coord.
;  HA_Belief_Beta2-Prey_Distance          Belief (DC): The distance between the Beta2 and the Prey.
;  HA_Belief_Beta2-Prey_Heading           Belief (DC): The heading from the Beta2 and the Prey.
;  HA_Belief_Beta2-Encircle-X             Belief (DC): X-coord of the Beta2 encircle point.
;  HA_Belief_Beta2-Encircle-Y             Belief (DC): Y-coord of the Beta2 encircle point.
;  HA_Belief_Omega1                     Belief (PC): The turtle number of Omega1.
;  HA_Belief_Omega1-Seen                  Belief (DC): Omega1 has been visually observed - True/False.
;  HA_Belief_Omega1-Known                 Belief: HA has knowledge of the Omega1's last location. - True/False.
;  HA_Belief_Omega2                     Belief (PC): The turtle number of Omega2.
;  HA_Belief_Omega2-Seen                  Belief: Omega2 has been visually observed - True/False.
;  HA_Belief_Omega2-Known                 Belief: HA has knowledge of the Omega2's last location. - True/False.
;  HA_Belief_Heard-Call                 Belief (DC): HA heard a call from one of the other HAs.
;  HA_Belief_Heard-Alpha                  Belief (HC): HA heard a call from the Alpha.
;  HA_Belief_Encircle-X                   Belief (DC): X-coord of the encircle point.
;  HA_Belief_Encircle-Y                   Belief (DC): Y-coord of the encircle point.
;  HA_Belief_Hunt-Mode                    Belief (HC): The hunt mode that the HA is in: Rest, Search, Encircle, Strike, Eat.
;  HA_State_Hunter-Type                   State (PC): The type of Hunting Agent: lioness, wolf, etc.
;  HA_State_Hunter-Class                  State (PC): The class of hunter: alpha, beta, omega, etc.
;  HA_State_Hunter-Level                  State (PC): The level of the HA is used to distinguish two HAs of the same Class (ie. Beta1 from Beta2).

;--Instantiation--;
to HA_belief_instantiate-lioness_alpha
  create-hunters 1
    [ set color red
      set size 5
      set shape "default"
      set pen-size 1

      setxy  0 0
      facexy 0 1

      set HA_State_Hunter-Type "Lioness"
      set HA_State_Hunter-Class "Alpha"
      set HA_Belief_Hunter-Leader [who] of self
      set HA_Belief_Beta1 1

      set label (word "Alpha" 1 )
      set label-color white

      set HA_Belief_Hunt-Mode "Rest"
      set GA_State_Energy-Current 75
      set GA_State_Water-Current 150
      set GA_Design_Speed-Max 5
      set GA_State_Speed-Current 0
      set GA_State_Voice-Level 7
    ]
end

to HA_belief_instantiate-lioness_beta
  create-hunters 1 [
    set size 5
    set shape "default"
    set pen-size 1

    setxy 10 -10 ;
    facexy 0 1

    set HA_State_Hunter-Type "Lioness"
    set HA_State_Hunter-Class "Beta"

    set HA_Belief_Hunter-Leader 0

    set label (word "Beta" 1)
    set label-color white
    set color 13

    set HA_Belief_Hunt-Mode "Rest"
    set GA_State_Energy-Current 75
    set GA_State_Water-Current 150
    set GA_Design_Speed-Max 5
    set GA_State_Speed-Current 0
    set GA_State_Voice-Level 7
    ]
end

to HA_belief_instantiate-wolf_alpha
  create-hunters 1
    [ set color red
      set size 5
      set shape "default"
```

```
    set pen-size 1

    setxy 0 0 ;random-xcor random-ycor
    facexy 0 1

    set HA_Belief_Beta1 -1
    set HA_Belief_Beta2 -1
    set HA_Belief_Omega1 -1
    set HA_Belief_Omega2 -1

    set HA_State_Hunter-Type "Wolf"
    set HA_State_Hunter-Class "Alpha"
    set HA_Belief_Hunter-Leader [who] of self

    let i 0
    ask hunters [
      if (HA_State_Hunter-Class = "Alpha") [ set i (i + 1) ]
      set HA_State_Hunter-Level i
    ]

    set label (word "Alpha" WA_Belief_HA-Alpha_Quantity )
    set label-color white

;     show [(word "("HA_State_Hunter-Type ", " HA_State_Hunter-Level ", " HA_Belief_Hunter-Leader ")"  )] of self

    set HA_Belief_Hunt-Mode "Rest"
    set GA_State_Energy-Current 75
    set GA_State_Water-Current 150
    set GA_Design_Speed-Max 5
    set GA_State_Speed-Current 0
    set GA_State_Voice-Level 7
  ]
end

to HA_belief_instantiate-wolf_beta
  create-hunters 1 [
;   set color 14
    set size 5
    set shape "default"
    set pen-size 1

  ;  setxy random-xcor random-ycor ; 0 0
    facexy 0 1

    set HA_Belief_Omega1 -1
    set HA_Belief_Omega2 -1

    set HA_State_Hunter-Type "Wolf"
    set HA_State_Hunter-Class "Beta"

    let i 0
    let j who
    ask hunters [
;   show [(word "Hunter - ("HA_State_Hunter-Type ", " HA_State_Hunter-Level ", " HA_Belief_Hunter-Leader ")"  )] of self
      if (HA_State_Hunter-Type = "Beta") [ set i (i + 1) ]
        let assigned false
        if (HA_State_Hunter-Type = "Alpha") [
          if HA_Belief_Beta1 = -1 [
            set HA_Belief_Beta1 j
            set assigned true
          ]
          if (assigned = false) and (HA_Belief_Beta2 = -1) [set HA_Belief_Beta2 j]
          set j who
;         show [(word "Alpha - ("HA_State_Hunter-Type ", " HA_State_Hunter-Level ", " HA_Belief_Hunter-Leader ", " HA_Belief_Hunter-Beta1 ", " HA_Belief_Hunter-Beta2 ")"  )]
                  of self
        ]
      ]
      set HA_State_Hunter-Level i
      set HA_Belief_Hunter-Leader j

;     show [(word "Beta - ("HA_State_Hunter-Type ", " HA_State_Hunter-Level ", " HA_Belief_Hunter-Leader ", " who ")"  )] of self

      if WA_Belief_HA-Beta_Quantity = 2 [
        set label (word "Beta" 1 )
        set color 13
        setxy 10 -10
        show "my colour is 13"
      ]

      if WA_Belief_HA-Beta_Quantity = 1 [
        set label (word "Beta" 2 )
        set color 14
        setxy -10 -10
        show "my colour is 14"
      ]

      set HA_Belief_Hunt-Mode "Rest"
      set label-color white

      set GA_State_Energy-Current 75
      set GA_State_Water-Current 150
      set GA_Design_Speed-Max 5
      set GA_State_Speed-Current 0
```

```
      set GA_State_Voice-Level 7
    ]
end

to HA_belief_instantiate-wolf_omega
  create-hunters 1
    [ set color 16
      set size 5
      set shape "default"
      set pen-size 1

      setxy random-xcor random-ycor ; 0 0
      facexy 0 1

      set HA_State_Hunter-Type "Wolf"
      set HA_State_Hunter-Class "Omega"

      let i 0
      let j who
      let k who
      let assigned false
      ask hunters [
;       set j k
;       show [(word "Hunter - ("HA_State_Hunter-Type ", " HA_State_Hunter-Level ", " HA_Belief_Hunter-Leader ")"  )] of self
        if (HA_State_Hunter-Type = "Omega") [ set i (i + 1) ]
;       set assigned false
        if (HA_State_Hunter-Type = "Alpha") and (assigned = false) [
          if HA_Belief_Omega1 = -1 [
            set HA_Belief_Omega1 j
            set k who
            set assigned true
          ]
          if (assigned = false) and (HA_Belief_Omega2 = -1)[
            set HA_Belief_Omega2 j
            set assigned true
            set k who
          ]
;         show [(word "Alpha/Beta - ("HA_State_Hunter-Type ", " HA_State_Hunter-Level ", " HA_Belief_Hunter-Leader ", " HA_Belief_Hunter-Omega1 ", " HA_Belief_Hunter-
                  Omega2 ")"  )] of self
        ]

        if (HA_State_Hunter-Type = "Beta") and (assigned = false) [
          if HA_Belief_Omega1 = -1 [
            set HA_Belief_Omega1 j
            set k who
            set assigned true
          ]
          if (assigned = false) and (HA_Belief_Omega2 = -1)[
            set HA_Belief_Omega2 j
            set assigned true
            set k who
          ]

        ]
;       show [(word "Alpha/Beta - ("HA_State_Hunter-Type ", " HA_State_Hunter-Level ", " HA_Belief_Hunter-Leader ", " HA_Belief_Hunter-Omega1 ", " HA_Belief_Hunter-
                  Omega2 ")"  )] of self
;       if assigned = true [set k who]
      ]
      set HA_State_Hunter-Level i
      set HA_Belief_Hunter-Leader k

;     show [(word "Me - ("HA_State_Hunter-Type ", " HA_State_Hunter-Level ", " HA_Belief_Hunter-Leader ", " who ")"  )] of self

      if WA_Belief_HA-Omega_Quantity = 1 [
        setxy 0 -25
      ]

      set label (word "Omega" WA_Belief_HA-Omega_Quantity )
      set HA_Belief_Hunt-Mode "Rest"
      set label-color white

      set GA_State_Energy-Current 75
      set GA_State_Water-Current 150
      set GA_Design_Speed-Max 5
      set GA_State_Speed-Current 0
      set GA_State_Voice-Level 7
    ]
end

;--BRF--;

to HA_brf_audio-update
  let sender 0
  let name "name"
  let sound-x 0
  let sound-y 0
  let sound-level 0
  let message "none"
  let sound-bearing 0
  let hunter-distance 500

  HA_perception_sound
```

```
  foreach HA_Perceive-Sounds [ [m] ->
    set sender (item 0 m)
    set name (item 1 m)
    set sound-x (item 2 m)
    set sound-y (item 3 m)
    set sound-level (item 4 m)
    set message (item 5 m)

    if (sender != [who] of self) and (ticks >= 2)[    ;(turtle sender != [who] of self) ... (xcor != sound-x) and (ycor != sound-y)

      set HA_Belief_Heard-Call true

;     show "heard call"
;     show [(word "HA_brf_hunters-audio - (" sender ", " sound-x ", " sound-y ", " sound-level ")"  )] of self
      if (name = "Alpha1") [
;       show "heard Alpha"
        set HA_Belief_Heard-Alpha true
        set hunter-distance (distance turtle sender)
        if hunter-distance > 1 [
          set HA_Belief_Prey-Heading towardsxy sound-x sound-y
        ]
      ]
      if (HA_Belief_Heard-Alpha = false) and (distance turtle sender < hunter-distance) [
        set hunter-distance (distance turtle sender)
        if hunter-distance > 1 [
          set HA_Belief_Prey-Heading towardsxy sound-x sound-y
        ]
      ]
    ]
  ]

end

to HA_brf_cycle
;  set HA_Belief_Prey false
;  set HA_Belief_Prey-atSource false
;  set HA_Belief_Hunter-Prey_Distance 100
;  set HA_Belief_Heard-Call false
;  set HA_Belief_Heard-Alpha false
;  set HA_Belief_Hunter-Leader false
;  set HA_Belief_Alpha false
;  set HA_Belief_Beta1 false
;  set HA_Belief_Beta2 false
;  set HA_Belief_Omega1 false

  GA_brf_cycle
  HA_perception_visual-scan
  HA_brf_visual-update
  HA_brf_audio-update

end

to HA_brf_lioness-spatial_update
  let dx_alpha-prey (HA_Belief_Prey-X - HA_Belief_Alpha-X)
  let dy_alpha-prey (HA_Belief_Prey-Y - HA_Belief_Alpha-Y)

  set HA_Belief_Alpha-Prey_Distance (sqrt( ((dx_alpha-prey) ^ 2) + ((dy_alpha-prey) ^ 2) ))

  set HA_Belief_Encircle-X (HA_Belief_Prey-X + dx_alpha-prey)
  set HA_Belief_Encircle-Y (HA_Belief_Prey-Y + dy_alpha-prey)

  set HA_Belief_Beta1-Encircle-X HA_Belief_Encircle-X
  set HA_Belief_Beta1-Encircle-Y HA_Belief_Encircle-Y

  set HA_Belief_Beta1-Prey_Distance (sqrt( ((HA_Belief_Beta1-X - HA_Belief_Prey-X) ^ 2) + ((HA_Belief_Beta1-Y - HA_Belief_Prey-Y) ^ 2) ))
end

to HA_brf_wolf-spatial_update
  ;let dx_beta1-prey (HA_Belief_Prey-X - HA_Belief_Beta1-X)
  ;let dy_beta1-prey (HA_Belief_Prey-Y - HA_Belief_Beta1-Y)

  ;let dx_beta2-prey (HA_Belief_Prey-X - HA_Belief_Beta2-X)
  ;let dy_beta2-prey (HA_Belief_Prey-Y - HA_Belief_Beta2-Y)

  let dx_alpha-prey (HA_Belief_Prey-X - HA_Belief_Alpha-X)
  let dy_alpha-prey (HA_Belief_Prey-Y - HA_Belief_Alpha-Y)

  show [(word "Prey-X - ("HA_Belief_Prey-X") Prey-Y - ("HA_Belief_Prey-Y")"  )] of self
  show [(word "Alpha-X - ("HA_Belief_Alpha-X") Alpha-Y - ("HA_Belief_Alpha-Y")"  )] of self

  set HA_Belief_Alpha-Prey_Distance (sqrt( ((dx_alpha-prey) ^ 2) + ((dy_alpha-prey) ^ 2) ))

  if (label = "Beta1") or (label = "Alpha1") [
    set HA_Belief_Beta1-Encircle-X (HA_Belief_Prey-X + (HA_Belief_Alpha-Prey_Distance * 0.5))
    set HA_Belief_Beta1-Encircle-Y (HA_Belief_Prey-Y + (HA_Belief_Alpha-Prey_Distance * 1.2))

    set HA_Belief_Beta1-Prey_Distance (sqrt( ((HA_Belief_Beta1-X - HA_Belief_Prey-X) ^ 2) + ((HA_Belief_Beta1-Y - HA_Belief_Prey-Y) ^ 2) ))
    set HA_Belief_Encircle-X HA_Belief_Beta1-Encircle-X
    set HA_Belief_Encircle-Y HA_Belief_Beta1-Encircle-Y
  ]

  if label = "Beta2" or (label = "Alpha1") [
    set HA_Belief_Beta2-Encircle-X (HA_Belief_Prey-X + (HA_Belief_Alpha-Prey_Distance * -0.5))
```

```
    set HA_Belief_Beta2-Encircle-Y (HA_Belief_Prey-Y + (HA_Belief_Alpha-Prey_Distance * 1.2))

    set HA_Belief_Beta2-Prey_Distance (sqrt( ((HA_Belief_Beta2-X - HA_Belief_Prey-X) ^ 2) + ((HA_Belief_Beta2-Y - HA_Belief_Prey-Y) ^ 2) ))
    set HA_Belief_Encircle-X HA_Belief_Beta2-Encircle-X
    set HA_Belief_Encircle-Y HA_Belief_Beta2-Encircle-Y
  ]

  if label = "Omega1" [
   ; set HA_Belief_Encircle-X (HA_Belief_Prey-X + ((HA_Belief_Alpha-X + HA_Belief_Beta1-X + HA_Belief_Beta2-X) / 3))
   ; set HA_Belief_Encircle-Y (HA_Belief_Prey-Y + ((HA_Belief_Alpha-Y + HA_Belief_Beta1-Y + HA_Belief_Beta2-Y) / 3))

    set HA_Belief_Encircle-X ((HA_Belief_Alpha-X + HA_Belief_Beta1-X + HA_Belief_Beta2-X) / 3)
    set HA_Belief_Encircle-Y ((HA_Belief_Alpha-Y + HA_Belief_Beta1-Y + HA_Belief_Beta2-Y) / 3)

    show [(word "Omega-Encircle-X - ("HA_Belief_Encircle-X") Omega-Encircle-Y - ("HA_Belief_Encircle-Y")"  )] of self
    show [(word "Alpha-X - ("HA_Belief_Alpha-X") Alpha-Y - ("HA_Belief_Alpha-Y")"  )] of self
    show [(word "Beta1-X - ("HA_Belief_Beta1-X") Beta1-Y - ("HA_Belief_Beta1-Y")"  )] of self
    show [(word "Beta2-X - ("HA_Belief_Beta2-X") Beta2-Y - ("HA_Belief_Beta2-Y")"  )] of self

;    set HA_Belief_Omega-Prey_Distance (sqrt( ((HA_Belief_Beta1-X - HA_Belief_Prey-X) ^ 2) + ((HA_Belief_Beta1-Y - HA_Belief_Prey-Y) ^ 2) ))
  ]

end

to HA_brf_reset-beliefs
  set HA_Belief_Prey-Known false
  set HA_Belief_Alpha-Known false
  set HA_Belief_Beta1-Known false
  set HA_Belief_Beta2-Known false
  set HA_Belief_Omega1-Known false
  set HA_Belief_Omega2-Known false
end

to HA_brf_visual-update
  let patch-colour 0
  let temp-heading 0
  let temp-distance 100
  let temp-x 0
  let temp-y 0
  let agent-number 1000
  let agent-heading 1000
  let agent-speed 1000
  let i 0
  set HA_Belief_Hunter-Prey_Distance 110

  set HA_Belief_Prey-Seen false
  set HA_Belief_Prey-atSource false
;-  set HA_Belief_Hunter-Prey_Distance 100
;-  set HA_Belief_Heard-Call false
;-  set HA_Belief_Heard-Alpha false
;  set HA_Belief_Hunter-Leader false
  set HA_Belief_Alpha-Seen false
  set HA_Belief_Beta1-Seen false
  set HA_Belief_Beta2-Seen false
  set HA_Belief_Omega1-Seen false
  set HA_Belief_Alpha-Prey_Attack false

;  let n first GA_Perceive_Visual-Scan
;  set GA_Belief_Patch-Colour (item 0 n)
;  show [(word "(HA - "GA_Belief_Patch-Colour")")] of self

 ; ifelse (GA_Belief_Patch-Colour = brown) or (GA_Belief_Patch-Colour = 19) [
;    show "Captured Prey!"
;    set HA_Belief_Prey-atSource true
;  ][
  foreach GA_Perceive_Visual-Scan [ [m] ->
    set patch-colour (item 0 m)
    set temp-heading (item 1 m)
    set temp-distance (item 2 m)
    set temp-x (item 3 m)
    set temp-y (item 4 m)
    set agent-number (item 5 m)
    set agent-heading (item 6 m)
    set agent-speed (item 7 m)

    ifelse (i <= 0) [
      set GA_Belief_Patch-Colour patch-colour
      if (GA_Belief_Patch-Colour = brown) or (GA_Belief_Patch-Colour = 19) [
        show "Captured Prey!"
        set HA_Belief_Prey-atSource true
        set HA_Belief_Prey-Seen true
      ]
    ][
      if (patch-colour = brown) [
        set HA_Belief_Prey-Seen true
        show "found prey"

        if (temp-distance < HA_Belief_Hunter-Prey_Distance) [
          set HA_Belief_Hunter-Prey_Distance temp-distance
          set HA_Belief_Hunter-Prey_Heading temp-heading
          set HA_Belief_Prey-X temp-x
          set HA_Belief_Prey-Y temp-y
          show [(word "Visual-HA_Belief_Prey-X - ("HA_Belief_Prey-X") Visual-HA_Belief_Prey-Y - ("HA_Belief_Prey-Y")"  )] of self
```

```
      ]
    ]
    if (patch-colour = red) [
      set HA_Belief_Alpha-Seen true
      show "Visual Scan - I see Alpha"
      set HA_Belief_Alpha-Heading agent-heading
      set HA_Belief_Alpha-Speed agent-speed

      set HA_Belief_Hunter-Alpha_Heading temp-heading
      set HA_Belief_Hunter-Alpha_Distance temp-distance
      set HA_Belief_Alpha-X temp-x
      set HA_Belief_Alpha-Y temp-y
    ]
    if (patch-colour = 13) [
      set HA_Belief_Beta1-Seen true
      show "Visual Scan - I see Beta1"
      set HA_Belief_Hunter-Beta1_Heading temp-heading
      set HA_Belief_Hunter-Beta1_Distance temp-distance
      set HA_Belief_Beta1-X temp-x
      set HA_Belief_Beta1-Y temp-y
      show [(word "Visual-HA_Belief_Beta1-X - ("HA_Belief_Beta1-X") Visual-HA_Belief_Beta1-Y - ("HA_Belief_Beta1-Y")"  )] of self
    ]
    if (patch-colour = 14) [
      set HA_Belief_Beta2-Seen true
      show "Visual Scan - I see Beta2"
      set HA_Belief_Hunter-Beta2_Heading temp-heading
      set HA_Belief_Hunter-Beta2_Distance temp-distance
      set HA_Belief_Beta2-X temp-x
      set HA_Belief_Beta2-Y temp-y
      show [(word "Visual-HA_Belief_Beta2-X - ("HA_Belief_Beta1-X") Visual-HA_Belief_Beta2-Y - ("HA_Belief_Beta1-Y")"  )] of self
    ]
    if (patch-colour = 19) [                          ; this represents an HA and PA on the same patch agent.
      set HA_Belief_Alpha-Seen true
      set HA_Belief_Prey-Seen true
      set HA_Belief_Alpha-Prey_Attack true
      set HA_Belief_Alpha-Prey_Distance 0
      show "Visual Scan - Alpha on Prey"
      set HA_Belief_Alpha-Heading agent-heading
      set HA_Belief_Alpha-Speed agent-speed

      set HA_Belief_Hunter-Alpha_Heading temp-heading
      set HA_Belief_Hunter-Alpha_Distance temp-distance
      set HA_Belief_Alpha-X temp-x
      set HA_Belief_Alpha-Y temp-y

      set HA_Belief_Hunter-Prey_Distance temp-distance
      set HA_Belief_Hunter-Prey_Heading temp-heading
      set HA_Belief_Prey-X temp-x
      set HA_Belief_Prey-Y temp-y
    ]
   ]
   set i i + 1
  ]
end
;;;;;;;;;;;;;;;; Deliberation ;;;;;;;;;;;;;;;;

to HA_deliberate_lioness-alpha
  HA_brf_cycle
  GA_deliberate_cycle

  if HA_Belief_Beta1-Seen = true [
    show "Deliberate - I See Beta1"
    set HA_Belief_Beta1-Known true
  ]

  if GA_Desire_Food = true [     ; this is the entry condition into the "Hunting Cycle"
    if (HA_Belief_Hunt-Mode = "Rest") [
      set HA_Belief_Hunt-Mode "I Search for Prey"
      show "I'm Searching For Prey"
    ]
  ]

 if HA_Belief_Hunt-Mode = "I Search for Prey" [
    if HA_Belief_Heard-Call = true [
      set HA_Belief_Hunt-Mode "Pack Found Prey"
      show "Pack Found Prey"
    ]

    if (HA_Belief_Prey-Seen = true) [
      set HA_Belief_Hunt-Mode "I See Prey"
      set HA_Belief_Prey-Known true
      show "I See Prey"
      if (HA_Belief_Heard-Call = false) [
        HA_irf_communicate-prey_found
      ]
    ]

    if (HA_Belief_Prey-Seen = false) and (HA_Belief_Prey-Known = true)[
      set HA_Belief_Hunt-Mode "I Search for Prey at Last Known Location"
    ]

    if (HA_Belief_Heard-Call = false) and (HA_Belief_Prey-Seen = false) [
;     show "I move to search for prey"
```

```
    HA_irf_search-prey
  ]
 ]

 if HA_Belief_Hunt-Mode = "Pack Found Prey" [
  HA_irf_move-alpha
  set HA_Belief_Hunt-Mode "I Search for Prey"
  show "Back to I Search For Prey"
 ]

 if HA_Belief_Hunt-Mode = "I Search for Prey at Last Known Location" [
  ifelse (HA_Belief_Prey-Seen = false) and (xcor = HA_Belief_Prey-X) and (ycor = HA_Belief_Prey-Y) [
    set HA_Belief_Prey-Known False
    show "At Last Known Prey Location"
  ][
    HA_irf_move-prey_fast
    show "Moving to Last Known Prey Location"
  ]
  set HA_Belief_Hunt-Mode "I Search for Prey"
 ]

 if HA_Belief_Hunt-Mode = "I See Prey" [
  HA_irf_communicate-prey_found
  if (HA_Belief_Hunter-Prey_Distance >= 16) [ HA_irf_move-prey_fast ]
  if (HA_Belief_Hunter-Prey_Distance < 16) [
    set HA_Belief_Hunt-Mode "Encircle"
    show "Move to Encircle"
  ]
 ]

 if HA_Belief_Hunt-Mode = "Encircle" [
  HA_brf_lioness-spatial_update
  show [(word "Encircle-X - ("HA_Belief_Encircle-X") Encircle-Y - ("HA_Belief_Encircle-Y")"  )] of self
  if (HA_Belief_Encircle-X - HA_Belief_Beta1-X < 5) and (HA_Belief_Encircle-Y - HA_Belief_Beta1-Y < 5) [
    set HA_Belief_Hunt-Mode "Approach"
    show "I see Beta...Move to Approach"
  ]
 ]

 if HA_Belief_Hunt-Mode = "Approach" [
  if (HA_Belief_Beta1-Seen = true) [
    if ((HA_Belief_Hunter-Prey_Distance < 16) and (HA_Belief_Hunter-Prey_Distance > 7)) [
      HA_irf_move-prey_normal-linear
    ]
    if (HA_Belief_Hunter-Prey_Distance < 9) [
      set HA_Belief_Hunt-Mode "Strike"
      show "Move to Strike"
    ]
  ]
 ]

 if HA_Belief_Hunt-Mode = "Strike" [
  HA_irf_move-prey_pounce
  set HA_Belief_Hunt-Mode "Move to Kill"
 ]

 if HA_Belief_Hunt-Mode = "Kill" [
  if HA_Belief_Prey-atSource = true [
    HA_irf_kill-prey
    set HA_Belief_Hunt-Mode "Move to Eat"
  ]
 ]

 if HA_Belief_Hunt-Mode = "Eat" [
    HA_irf_eat-prey
  set HA_Belief_Hunt-Mode "Rest"
 ]

 if HA_Belief_Hunt-Mode = "Rest" [
  HA_brf_reset-beliefs
 ]

 show [(word "Alpha Hunt State - ("HA_Belief_Hunt-Mode")")] of self
 show [(word "Alpha Patch Colour - ("pcolor")")] of patch-here

end

to HA_deliberate_lioness-beta
 HA_brf_cycle
 GA_deliberate_cycle

 if HA_Belief_Alpha-Seen = true [
    show "Deliberate - I See Alpha"
    set HA_Belief_Alpha-Known true
 ]

 if GA_Desire_Food = true [     ; this is the entry condition into the "Hunting Cycle"
  if (HA_Belief_Hunt-Mode = "Rest") [
    set HA_Belief_Hunt-Mode "I Search for Prey"
    show "I'm Searching For Prey"
  ]
 ]
```

```
if HA_Belief_Hunt-Mode = "I Search for Prey" [
  if HA_Belief_Heard-Call = true [
    set HA_Belief_Hunt-Mode "Pack Found Prey"
    show "Pack Found Prey"
  ]

  if (HA_Belief_Prey-Seen = true) [
    set HA_Belief_Hunt-Mode "I See Prey"
    set HA_Belief_Prey-Known true
    show "I See Prey"
    if (HA_Belief_Heard-Call = false) and (HA_Belief_Alpha-Seen = false) [
      HA_irf_communicate-prey_found
    ]
  ]

  if (HA_Belief_Prey-Seen = false) and (HA_Belief_Prey-Known = true)[
    set HA_Belief_Hunt-Mode "I Search for Prey at Last Known Location"
  ]

  if (HA_Belief_Heard-Call = false) and (HA_Belief_Prey-Seen = false) [
    HA_irf_search-prey
  ]
]

if HA_Belief_Hunt-Mode = "Pack Found Prey" [
  HA_irf_move-alpha
  set HA_Belief_Hunt-Mode "I Search for Prey"
  show "Back to I Search For Prey"
]

if HA_Belief_Hunt-Mode = "I Search for Prey at Last Known Location" [
  ifelse (HA_Belief_Prey-Seen = false) and (xcor = HA_Belief_Prey-X) and (ycor = HA_Belief_Prey-Y) [
    set HA_Belief_Prey-Known False
    show "At Last Known Prey Location"
  ][
    HA_irf_move-prey_fast
    show "Moving to Last Known Prey Location"
  ]
  set HA_Belief_Hunt-Mode "I Search for Prey"
]

if HA_Belief_Hunt-Mode = "I See Prey" [
  if (HA_Belief_Hunter-Prey_Distance >= 16) [ HA_irf_move-prey_fast ]
  if (HA_Belief_Hunter-Prey_Distance < 16) and (HA_Belief_Alpha-Known = true) [
    set HA_Belief_Hunt-Mode "Encircle"
    show "Move to Encircle"
  ]
 if (HA_Belief_Hunter-Prey_Distance < 16) and (HA_Belief_Alpha-Known = false) [
   HA_irf_search-alpha
  ]
]

if HA_Belief_Hunt-Mode = "Encircle" [
  HA_brf_lioness-spatial_update
  show [(word "Encircle-X - ("HA_Belief_Encircle-X") Encircle-Y - ("HA_Belief_Encircle-Y")"  )] of self
  HA_irf_move-encircle
  if (xcor = precision HA_Belief_Encircle-X 0) and (ycor =  precision HA_Belief_Encircle-Y 0) [
    HA_irf_face-prey
    set HA_Belief_Hunt-Mode "Approach"
    show "Move to Approach"
  ]
]

if HA_Belief_Hunt-Mode = "Approach" [
  if (HA_Belief_Prey-Seen = true) and (HA_Belief_Alpha-Known = true) [
    show "Approach - I see Prey and Alpha"
    HA_brf_lioness-spatial_update

;     if (abs(HA_Belief_Hunter-Prey_Heading - HA_Belief_Hunter-Alpha_Heading) <= 20 ) [ HA_irf_move-prey_normal-linear ]
    if HA_Belief_Beta1-Prey_Distance > (HA_Belief_Alpha-Prey_Distance - 1) [
       HA_irf_move-prey_normal-linear
    ]

    if HA_Belief_Beta1-Prey_Distance <= (HA_Belief_Alpha-Prey_Distance - 1) [
       ; wait until Alpha moves closer to Prey.
    ]

    if (HA_Belief_Hunter-Prey_Distance < 9) [
      set HA_Belief_Hunt-Mode "Strike"
      show "Move to Strike"
    ]
  ]
]

if HA_Belief_Hunt-Mode = "Strike" [
  HA_irf_move-prey_pounce
  set HA_Belief_Hunt-Mode "Move to Kill"
]

if HA_Belief_Hunt-Mode = "Kill" [
  if HA_Belief_Prey-atSource = true [
    HA_irf_kill-prey
    set HA_Belief_Hunt-Mode "Move to Eat"
```

```
  ]
 ]

 if HA_Belief_Hunt-Mode = "Eat" [
   HA_irf_eat-prey
  set HA_Belief_Hunt-Mode "Rest"
 ]

 if HA_Belief_Hunt-Mode = "Rest" [
  HA_brf_reset-beliefs
 ]

 show [(word "Beta Hunt State - ("HA_Belief_Hunt-Mode")")] of self

end

to HA_deliberate_wolf-alpha
 HA_brf_cycle
 GA_deliberate_cycle

 if HA_Belief_Beta1-Seen = true [
  show "Deliberate - I See Beta1"
  set HA_Belief_Beta1-Known true
 ]

 if HA_Belief_Beta2-Seen = true [
    show "Deliberate - I See Beta2"
    set HA_Belief_Beta2-Known true
 ]

 if GA_Desire_Food = true [     ; this is the entry condition into the "Hunting Cycle"
  if (HA_Belief_Hunt-Mode = "Rest") [
    set HA_Belief_Hunt-Mode "I Search for Prey"
    show "I'm Searching For Prey"
  ]
 ]

 if HA_Belief_Hunt-Mode = "I Search for Prey" [
  if HA_Belief_Heard-Call = true [
    set HA_Belief_Hunt-Mode "Pack Found Prey"
    show "Pack Found Prey"
  ]

  if (HA_Belief_Prey-Seen = true) [
    set HA_Belief_Hunt-Mode "I See Prey"
    set HA_Belief_Prey-Known true
    show "I See Prey"
    if (HA_Belief_Heard-Call = false) [
      HA_irf_communicate-prey_found
    ]
  ]

  if (HA_Belief_Prey-Seen = false) and (HA_Belief_Prey-Known = true)[
    set HA_Belief_Hunt-Mode "I Search for Prey at Last Known Location"
  ]

  if (HA_Belief_Heard-Call = 0) and (HA_Belief_Prey-Seen = false) [
    show "I move to search for prey"
    HA_irf_search-prey
  ]
 ]

 if HA_Belief_Hunt-Mode = "Pack Found Prey" [
  HA_irf_move-alpha
  set HA_Belief_Hunt-Mode "I Search for Prey"
  show "Back to I Search For Prey"
 ]

 if HA_Belief_Hunt-Mode = "I Search for Prey at Last Known Location" [
  ifelse (HA_Belief_Prey-Seen = false) and (xcor = HA_Belief_Prey-X) and (ycor = HA_Belief_Prey-Y) [
    set HA_Belief_Prey-Known False
    show "At Last Known Prey Location"
  ][
    HA_irf_move-prey_fast
    show "Moving to Last Known Prey Location"
  ]
  set HA_Belief_Hunt-Mode "I Search for Prey"
 ]

 if HA_Belief_Hunt-Mode = "I See Prey" [
  HA_irf_communicate-prey_found
  if (HA_Belief_Hunter-Prey_Distance >= 16) [ HA_irf_move-prey_fast ]
  if (HA_Belief_Hunter-Prey_Distance < 16) and (HA_Belief_Beta1-Seen = true) and (HA_Belief_Beta2-Seen = true)[
    set HA_Belief_Hunt-Mode "Encircle"
    show "Move to Encircle"
  ]
 ]

 if HA_Belief_Hunt-Mode = "Encircle" [
  HA_brf_wolf-spatial_update

  let beta1-dx abs(HA_Belief_Beta1-Encircle-X - HA_Belief_Beta1-X)
  let beta1-dy abs(HA_Belief_Beta1-Encircle-Y - HA_Belief_Beta1-Y)
```

```
    let beta2-dx abs(HA_Belief_Beta2-Encircle-X - HA_Belief_Beta2-X)
    let beta2-dy abs(HA_Belief_Beta2-Encircle-Y - HA_Belief_Beta2-Y)

show [(word "( Encircle1 - "HA_Belief_Beta1-X ", " HA_Belief_Beta1-Y ", "HA_Belief_Beta2-X ", "HA_Belief_Beta2-Y ")"  )] of self
show [(word "( Encircle2 - "HA_Belief_Beta1-Encircle-X ", "HA_Belief_Beta1-Encircle-Y ", "HA_Belief_Beta2-Encircle-X ", "HA_Belief_Beta2-Encircle-Y ")"  )] of self
show [(word "( Encircle3 - "beta1-dx ", " beta1-dy ", " beta2-dx ", " beta2-dy ")"  )] of self

    if (beta1-dx <= 8 ) and (beta1-dy <= 8 ) and (beta2-dx <= 8 ) and (beta2-dy <= 8 ) [
;   if (abs(HA_Belief_Hunter-Prey_Heading - HA_Belief_Hunter-Beta1_Heading) <= 20 ) and (abs(HA_Belief_Hunter-Prey_Heading - HA_Belief_Hunter-Beta2_Heading) <= 20
                )[

       set HA_Belief_Hunt-Mode "Approach"
       show "Move to Approach"
     ]

    if HA_Belief_Prey-Seen = false [
       set HA_Belief_Hunt-Mode "I Search for Prey"
     ]
  ]

  if HA_Belief_Hunt-Mode = "Approach" [
    if (HA_Belief_Beta1-Seen = true) and (HA_Belief_Beta2-Seen = true) [
      show "I see Beta1 and Beta2" ]
      if ((HA_Belief_Hunter-Prey_Distance < 16) and (HA_Belief_Hunter-Prey_Distance > 7)) [
        HA_irf_move-prey_normal-linear ]
      if (HA_Belief_Hunter-Prey_Distance < 9) [
        set HA_Belief_Hunt-Mode "Strike"
        show "Move to Strike"
      ]
      if HA_Belief_Prey-Seen = false [
        set HA_Belief_Hunt-Mode "I Search for Prey"
      ]
    ]

  if HA_Belief_Hunt-Mode = "Strike" [
    HA_irf_move-prey_pounce
    set HA_Belief_Hunt-Mode "Move to Kill"
  ]

  if HA_Belief_Hunt-Mode = "Kill" [
    if HA_Belief_Prey-atSource = true [
      HA_irf_kill-prey
      set HA_Belief_Hunt-Mode "Move to Eat"
    ]
  ]

  if HA_Belief_Hunt-Mode = "Eat" [
     HA_irf_eat-prey
     set HA_Belief_Hunt-Mode "Rest"
  ]

  if HA_Belief_Hunt-Mode = "Rest" [
    HA_brf_reset-beliefs
  ]

 show [(word "Alpha Hunt State - ("HA_Belief_Hunt-Mode")")] of self
 show [(word "Alpha Patch Colour - ("pcolor")")] of patch-here

end

to HA_deliberate_wolf-beta
  HA_brf_cycle
  GA_deliberate_cycle

  if HA_Belief_Alpha-Seen = true [
      show "Deliberate - I See Alpha"
      set HA_Belief_Alpha-Known true
;     if (abs(xcor - HA_Belief_Alpha-X) < 4) and (abs(ycor - HA_Belief_Alpha-Y) < 4) [
;       show "I'm next to Alpha"
;     ]
  ]

  if GA_Desire_Food = true [     ; this is the entry condition into the "Hunting Cycle"
    if (HA_Belief_Hunt-Mode = "Rest") [
      set HA_Belief_Hunt-Mode "I Search for Prey"
      show "I'm Searching For Prey"
    ]
  ]

  if HA_Belief_Hunt-Mode = "I Search for Prey" [
    if HA_Belief_Heard-Call = true [
      set HA_Belief_Hunt-Mode "Pack Found Prey"
      show "Pack Found Prey"
    ]

    if (HA_Belief_Prey-Seen = true) [
      set HA_Belief_Hunt-Mode "I See Prey"
      set HA_Belief_Prey-Known true
      show "I See Prey"
      if (HA_Belief_Heard-Call = false) and (HA_Belief_Alpha-Seen = false) [
        HA_irf_communicate-prey_found
      ]
    ]
```

```
  if (HA_Belief_Prey-Seen = false) and (HA_Belief_Prey-Known = true)[
    set HA_Belief_Hunt-Mode "I Search for Prey at Last Known Location"
  ]

  if (HA_Belief_Heard-Call = 0) and (HA_Belief_Prey-Seen = false) [
    show "I move to search for prey"
    HA_irf_search-prey
  ]
 ]

 if HA_Belief_Hunt-Mode = "Pack Found Prey" [
   HA_irf_move-alpha
   set HA_Belief_Hunt-Mode "I Search for Prey"
   show "Back to I Search For Prey"
 ]

 if HA_Belief_Hunt-Mode = "I Search for Prey at Last Known Location" [
;    if (HA_Belief_Prey-Seen = true) [
;      set HA_Belief_Hunt-Mode "I See Prey"
;      set HA_Belief_Prey-Known true
;      show "Last Known Location - I See Prey"
;    ]
   ifelse (HA_Belief_Prey-Seen = false) and (xcor = HA_Belief_Prey-X) and (ycor = HA_Belief_Prey-Y) [
     set HA_Belief_Prey-Known False
     show "At Last Known Prey Location"
   ][
     HA_irf_move-prey_fast
     show "Moving to Last Known Prey Location"
   ]
   set HA_Belief_Hunt-Mode "I Search for Prey"
 ]

 if HA_Belief_Hunt-Mode = "I See Prey" [
   if (HA_Belief_Hunter-Prey_Distance >= 16) [ HA_irf_move-prey_fast ]
   if (HA_Belief_Hunter-Prey_Distance < 16) and (HA_Belief_Alpha-Known = true) [
     set HA_Belief_Hunt-Mode "Encircle"
     show "Move to Encircle"
   ]
  if (HA_Belief_Hunter-Prey_Distance < 16) and (HA_Belief_Alpha-Known = false) [
    HA_irf_search-alpha
   ]
 ]

 if HA_Belief_Hunt-Mode = "Encircle" [
   HA_brf_wolf-spatial_update
   show [(word "Encircle-X - ("HA_Belief_Encircle-X") Encircle-Y - ("HA_Belief_Encircle-Y")"  )] of self
   HA_irf_move-encircle
   if (xcor = precision HA_Belief_Encircle-X 0) and (ycor =  precision HA_Belief_Encircle-Y 0) [
     HA_irf_face-prey
     set HA_Belief_Hunt-Mode "Approach"
     show "Move to Approach"
   ]
 ]

;  if HA_Belief_Hunt-Mode = "Approach" [
 ;   if (HA_Belief_Prey-Seen = true) and (HA_Belief_Alpha-Known = true) [
 ;     show "Approach - I see Prey and Alpha"
 ;     HA_brf_wolf-spatial_update
 ;     HA_irf_move-prey_normal-linear

;      if label = "Beta1" [
;        if HA_Belief_Beta1-Prey_Distance > (HA_Belief_Alpha-Prey_Distance - 5) [
;          HA_irf_move-prey_normal-linear
;        ]
;
;        if HA_Belief_Beta1-Prey_Distance <= (HA_Belief_Alpha-Prey_Distance - 5) [
;          ; wait until Alpha moves closer to Prey.
;        ]

;      ]

 ;     if label = "Beta2" [
 ;       if HA_Belief_Beta2-Prey_Distance > (HA_Belief_Alpha-Prey_Distance - 5) [
  ;        HA_irf_move-prey_normal-linear
   ;     ]
;
 ;       if HA_Belief_Beta2-Prey_Distance < (HA_Belief_Alpha-Prey_Distance - 5) [
  ;        ; wait until Alpha moves closer to Prey.
   ;     ]
    ;  ]

;      if (abs(HA_Belief_Hunter-Prey_Heading - HA_Belief_Hunter-Alpha_Heading) <= 20 ) [ HA_irf_move-prey_normal-linear ]
;      if (HA_Belief_Hunter-Prey_Distance < 9) [
;        set HA_Belief_Hunt-Mode "Strike"
;        show "Move to strike"
;      ]
;    ]
;  ]

  if HA_Belief_Hunt-Mode = "Approach" [
    if (HA_Belief_Prey-Seen = true) and (HA_Belief_Alpha-Known = true) [
      show "Approach - I see Prey and Alpha"
```

```
      HA_brf_wolf-spatial_update
      if (HA_Belief_Hunter-Prey_Distance > (HA_Belief_Alpha-Prey_Distance)) [
        HA_irf_move-prey_normal-linear
      ]
    ]

;     if (abs(HA_Belief_Hunter-Prey_Heading - HA_Belief_Hunter-Alpha_Heading) <= 20 ) [ HA_irf_move-prey_normal-linear ]


    if (HA_Belief_Hunter-Prey_Distance < 9) and (HA_Belief_Alpha-Prey_Attack = true) [
      set HA_Belief_Hunt-Mode "Strike"
      show "Move to strike"
    ]
  ]

  if HA_Belief_Hunt-Mode = "Strike" [
    HA_irf_move-prey_pounce
    set HA_Belief_Hunt-Mode "Move to Kill"
  ]

  if HA_Belief_Hunt-Mode = "Kill" [
    if HA_Belief_Prey-atSource = true [
      HA_irf_kill-prey
      set HA_Belief_Hunt-Mode "Move to Eat"
    ]
  ]

  if HA_Belief_Hunt-Mode = "Eat" [
    HA_irf_eat-prey
    set HA_Belief_Hunt-Mode "Rest"
  ]

  if HA_Belief_Hunt-Mode = "Rest" [
    HA_brf_reset-beliefs
  ]

  show [(word "Beta Hunt State - ("HA_Belief_Hunt-Mode")")] of self

end

to HA_deliberate_wolf-omega
  HA_brf_cycle
  GA_deliberate_cycle

  if HA_Belief_Alpha-Seen = true [
      show "Deliberate - I See Alpha"
      set HA_Belief_Alpha-Known true
  ]

  if HA_Belief_Beta1-Seen = true [
    show "Deliberate - I See Beta1"
    set HA_Belief_Beta1-Known true
  ]

  if HA_Belief_Beta2-Seen = true [
      show "Deliberate - I See Beta2"
      set HA_Belief_Beta2-Known true
  ]

  if GA_Desire_Food = true [     ; this is the entry condition into the "Hunting Cycle"
    if (HA_Belief_Hunt-Mode = "Rest") [
      set HA_Belief_Hunt-Mode "I Search for Prey"
      show "I'm Searching For Prey"
    ]
  ]

  if HA_Belief_Hunt-Mode = "I Search for Prey" [

    if HA_Belief_Heard-Call = true [
      set HA_Belief_Hunt-Mode "Pack Found Prey"
      show "Pack Found Prey"
    ]

    if (HA_Belief_Prey-Seen = true) [
      set HA_Belief_Hunt-Mode "I See Prey"
      set HA_Belief_Prey-Known true
      show "I See Prey"
      if (HA_Belief_Heard-Call = false) and (HA_Belief_Alpha-Seen = false) [
        HA_irf_communicate-prey_found
      ]
    ]

    if (HA_Belief_Prey-Seen = false) and (HA_Belief_Prey-Known = true)[
      set HA_Belief_Hunt-Mode "I Search for Prey at Last Known Location"
    ]

;    show [(word "Heard-Call - ("HA_Belief_Heard-Call") Seen-Prey - ("HA_Belief_Prey-Seen")"  )] of self

    if (HA_Belief_Heard-Call = 0) and (HA_Belief_Prey-Seen = false) [
      show "I move to search for prey"
      HA_irf_search-prey
    ]
  ]
```

```
  if HA_Belief_Hunt-Mode = "Pack Found Prey" [
    HA_irf_move-alpha
    set HA_Belief_Hunt-Mode "I Search for Prey"
    show "Back to I Search For Prey"
  ]

  if HA_Belief_Hunt-Mode = "I Search for Prey at Last Known Location" [
    ifelse (HA_Belief_Prey-Seen = false) and (xcor = HA_Belief_Prey-X) and (ycor = HA_Belief_Prey-Y) [
      set HA_Belief_Prey-Known False
      show "At Last Known Prey Location"
    ][
      HA_irf_move-prey_fast
      show "Moving to Last Known Prey Location"
    ]
    set HA_Belief_Hunt-Mode "I Search for Prey"
  ]

  if HA_Belief_Hunt-Mode = "I See Prey" [
    if (HA_Belief_Hunter-Prey_Distance >= 16) [ HA_irf_move-prey_fast ]
    if (HA_Belief_Hunter-Prey_Distance < 16) and (HA_Belief_Alpha-Known = true) [
      set HA_Belief_Hunt-Mode "Encircle"
      show "Move to Encircle"
    ]
   if (HA_Belief_Hunter-Prey_Distance < 16) and (HA_Belief_Alpha-Known = false) [
     HA_irf_search-alpha
    ]
  ]

  if HA_Belief_Hunt-Mode = "Encircle" [
    HA_brf_wolf-spatial_update
;    show [(word "Encircle-X - ("HA_Belief_Encircle-X") Encircle-Y - ("HA_Belief_Encircle-Y")"  )] of self
    HA_irf_move-encircle
    if (xcor = precision HA_Belief_Encircle-X 0) and (ycor =  precision HA_Belief_Encircle-Y 0) [
;      HA_irf_face-prey
      set HA_Belief_Hunt-Mode "Approach"
      show "Move to Approach"
    ]
  ]

  if HA_Belief_Hunt-Mode = "Approach" [
;    if (HA_Belief_Hunter-Beta1_Distance < 7) or (HA_Belief_Hunter-Beta2_Distance < 7) [
;      HA_irf_face-prey
;    ]
;    if (HA_Belief_Prey-Seen = true) and (HA_Belief_Alpha-Known = true) [
;      show "Approach - I see Prey and Alpha"
;      HA_brf_wolf-spatial_update
;      HA_irf_move-prey_normal-linear
;    ]

;      if (abs(HA_Belief_Hunter-Prey_Heading - HA_Belief_Hunter-Alpha_Heading) <= 20 ) [ HA_irf_move-prey_normal-linear ]

    HA_irf_face-prey

    if (HA_Belief_Prey-Seen = true) and (HA_Belief_Alpha-Known = true) [
      if (HA_Belief_Hunter-Prey_Distance >= 9) [
        HA_irf_move-prey_normal-linear
      ]

      if (HA_Belief_Hunter-Prey_Distance < 9) and (HA_Belief_Alpha-Prey_Attack = true) [
        set HA_Belief_Hunt-Mode "Strike"
        show "Move to strike"
      ]
    ]
  ]

  if HA_Belief_Hunt-Mode = "Strike" [
    HA_irf_move-prey_pounce
    set HA_Belief_Hunt-Mode "Move to Kill"
  ]

 if HA_Belief_Hunt-Mode = "Kill" [
    if HA_Belief_Prey-atSource = true [
      HA_irf_kill-prey
      set HA_Belief_Hunt-Mode "Move to Eat"
    ]
  ]

  if HA_Belief_Hunt-Mode = "Eat" [
    HA_irf_eat-prey
    set HA_Belief_Hunt-Mode "Rest"
  ]

  if HA_Belief_Hunt-Mode = "Rest" [
    HA_brf_reset-beliefs
  ]

  show [(word "Omega Hunt State - ("HA_Belief_Hunt-Mode")")] of self
end

;;;;;;;;;;;;;; Desire Review Function (DRF) ;;;;;;;;;;;;;;;;

to HA_drf_desire
```

```
end

;;;;;;;;;;;;;;; Intention Review Function (IRF) ;;;;;;;;;;;;;

to HA_irf_communicate-prey_found
  HA-action_talk-prey_found

end

to HA_irf_eat-prey
  HA_action_eat-prey
end

to HA_irf_face-prey
  HA_action_face-prey
end

to HA_irf_kill-prey
  HA_action_kill-prey
end

to HA_irf_move-alpha
  HA_action_move-alpha
  HA_action_face-alpha_heading
end

to HA_irf_move-encircle
  HA_action_move-encircle
end

to HA_irf_move-prey_fast
 HA_action_move-prey_fast
end

to HA_irf_move-prey_normal-linear
  HA_action_move-prey_normal
end

to HA_irf_move-prey_pounce
  HA_action_move-prey_pounce
end

to HA_irf_search-alpha
  HA_action-search-alpha
end

to HA_irf_search-prey
  HA_action_search-prey
end

;;;;;;;;;;;;;;; Perceptions ;;;;;;;;;;

;  HA_Perceive_Visual-Scan          Perception: this is a list of perceived objects during a visual scan.
;  HA_Perceive-Scent-Scan           Perception: this is a list of scents on the patches immediately surrounding the Hunter Agent.

to HA_perception_scent-scan_atSource
  set HA_Perceive-Scents []

  ask patch-here [
    let m []
    set m lput WA_Patch_State_Scent-Type m
    set m lput WA_Patch_State_Scent-Level m
    ask myself [set HA_Perceive-Scents lput m HA_Perceive-Scents]
  ]

end

to HA_perception_scent-scan_box
  set HA_Perceive-Scents []

  ask patch-ahead 1 [
    let m []
    set m lput WA_Patch_State_Scent-Type m
    set m lput WA_Patch_State_Scent-Level m
    ask myself [set HA_Perceive-Scents lput m HA_Perceive-Scents]
  ]

  ask patch-right-and-ahead 45 1 [
    let m []
    set m lput WA_Patch_State_Scent-Type m
    set m lput WA_Patch_State_Scent-Level m
    ask myself [set HA_Perceive-Scents lput m HA_Perceive-Scents]
  ]

  ask patch-right-and-ahead 90 1 [
    let m []
    set m lput WA_Patch_State_Scent-Type m
    set m lput WA_Patch_State_Scent-Level m
    ask myself [set HA_Perceive-Scents lput m HA_Perceive-Scents]
  ]
```

```
  ask patch-right-and-ahead 135 1 [
    let m []
    set m lput WA_Patch_State_Scent-Type m
    set m lput WA_Patch_State_Scent-Level m
    ask myself [set HA_Perceive-Scents lput m HA_Perceive-Scents]
  ]

  ask patch-right-and-ahead 180 1 [
    let m []
    set m lput WA_Patch_State_Scent-Type m
    set m lput WA_Patch_State_Scent-Level m
    ask myself [set HA_Perceive-Scents lput m HA_Perceive-Scents]
  ]

  ask patch-right-and-ahead 225 1 [
    let m []
    set m lput WA_Patch_State_Scent-Type m
    set m lput WA_Patch_State_Scent-Level m
    ask myself [set HA_Perceive-Scents lput m HA_Perceive-Scents]
  ]

  ask patch-right-and-ahead 270 1 [
    let m []
    set m lput WA_Patch_State_Scent-Type m
    set m lput WA_Patch_State_Scent-Level m
    ask myself [set HA_Perceive-Scents lput m HA_Perceive-Scents]
  ]

  ask patch-right-and-ahead 315 1 [
    let m []
    set m lput WA_Patch_State_Scent-Type m
    set m lput WA_Patch_State_Scent-Level m
    ask myself [set HA_Perceive-Scents lput m HA_Perceive-Scents]
  ]

end

to HA_perception_sound
  set HA_Perceive-Sounds []
  let sender 0
  let name "name"
  let sound-x 0
  let sound-y 0
  let sound-level 0
  let message "HA_perception_sound-message"
  let n []

  ask patch-here [
    foreach WA_Patch_State_Sounds [ [m] ->
      set sender (item 0 m)
      set name (item 1 m)
      set sound-x (item 2 m)
      set sound-y (item 3 m)
      set sound-level (item 4 m)
      set message (item 5 m)

      set n []
      if sound-level >= 1 [
        set n lput sender n
        set n lput name n
        set n lput sound-x n
        set n lput sound-y n
        set n lput sound-level n
        set n lput message n

        ask myself [set HA_Perceive-Sounds lput n HA_Perceive-Sounds]
      ]
    ]
  ]
;  show "HA_perception_sound" ; [(word "brf_sounds - ("message")")] of self

end

to HA_perception_visual-scan

  set GA_Perceive_Visual-Scan []

;;;;; Scan Agent's Patch;;;;;

  ask patch-here [ ; determines the patch that the Agent is standing on
    let m []
    set m lput pcolor m
    set m lput 0 m
    set m lput 0 m
    set m lput pxcor m
    set m lput pycor m
    set m lput WA_Patch_State_Agent-Number m
    set m lput WA_Patch_State_Agent-Heading m
    set m lput WA_Patch_State_Agent-Speed m
    ask myself [set GA_Perceive_Visual-Scan lput m GA_Perceive_Visual-Scan]
  ]

;;;;; Center Scan;;;;;
```

```
  set GA_State_Scan-Distance 1

  let i GA_State_Scan-Distance

  while [i <= 40] [
    ask patch-ahead i [ ; determines the patches in front of the Agent
      let h (remainder int ((towards myself) + 180) 360)
      let m []
      set m lput pcolor m
      set m lput h m
      set m lput i m
      set m lput pxcor m
      set m lput pycor m
      set m lput WA_Patch_State_Agent-Number m
      set m lput WA_Patch_State_Agent-Heading m
      set m lput WA_Patch_State_Agent-Speed m
      ask myself [set GA_Perceive_Visual-Scan lput m GA_Perceive_Visual-Scan]
      if ((pcolor = green) or (pcolor = grey)) [ (set i 41) ]
    ]
    set i (i + 1)
  ]

;;;;; Left Scan;;;;;

  set i 1

  while [i <= 40] [
    ask patch-left-and-ahead 2.5 i [ ; determines the patches left by 5 degrees in front of the Agent
      let h (remainder int ((towards myself) + 180) 360)
      let m []
      set m lput pcolor m
      set m lput h m
      set m lput i m
      set m lput pxcor m
      set m lput pycor m
      set m lput WA_Patch_State_Agent-Number m
      set m lput WA_Patch_State_Agent-Heading m
      set m lput WA_Patch_State_Agent-Speed m
      ask myself [set GA_Perceive_Visual-Scan lput m GA_Perceive_Visual-Scan]
      if ((pcolor = green) or (pcolor = grey)) [ (set i 41) ]
    ]
    set i (i + 1)
  ]

  set i 1

  while [i <= 40] [
    ask patch-left-and-ahead 5 i [ ; determines the patches left by 5 degrees in front of the Agent
      let h (remainder int ((towards myself) + 180) 360)
      let m []
      set m lput pcolor m
      set m lput h m
      set m lput i m
      set m lput pxcor m
      set m lput pycor m
      set m lput WA_Patch_State_Agent-Number m
      set m lput WA_Patch_State_Agent-Heading m
      set m lput WA_Patch_State_Agent-Speed m
      ask myself [set GA_Perceive_Visual-Scan lput m GA_Perceive_Visual-Scan]
      if ((pcolor = green) or (pcolor = grey)) [ (set i 41) ]
    ]
    set i (i + 1)
  ]

  set i 1

  while [i <= 40] [
    ask patch-left-and-ahead 7.5 i [ ; determines the patches left by 5 degrees in front of the Agent
      let h (remainder int ((towards myself) + 180) 360)
      let m []
      set m lput pcolor m
      set m lput h m
      set m lput i m
      set m lput pxcor m
      set m lput pycor m
      set m lput WA_Patch_State_Agent-Number m
      set m lput WA_Patch_State_Agent-Heading m
      set m lput WA_Patch_State_Agent-Speed m
      ask myself [set GA_Perceive_Visual-Scan lput m GA_Perceive_Visual-Scan]
      if ((pcolor = green) or (pcolor = grey)) [ (set i 41) ]
    ]
    set i (i + 1)
  ]

  set i 1

  while [i <= 40] [
    ask patch-left-and-ahead 10 i [ ; determines the patches left by 45 degrees in front of the Agent
      let h (remainder int ((towards myself) + 180) 360)
      let m []
      set m lput pcolor m
      set m lput h m
```

```
    set m lput i m
    set m lput pxcor m
    set m lput pycor m
    set m lput WA_Patch_State_Agent-Number m
    set m lput WA_Patch_State_Agent-Heading m
    set m lput WA_Patch_State_Agent-Speed m
    ask myself [set GA_Perceive_Visual-Scan lput m GA_Perceive_Visual-Scan]
    if ((pcolor = green) or (pcolor = grey)) [ (set i 41) ]
  ]
  set i (i + 1)
]

set i 1

while [i <= 40] [
  ask patch-left-and-ahead 12.5 i [ ; determines the patches left by 5 degrees in front of the Agent
    let h (remainder int ((towards myself) + 180) 360)
    let m []
    set m lput pcolor m
    set m lput h m
    set m lput i m
    set m lput pxcor m
    set m lput pycor m
    set m lput WA_Patch_State_Agent-Number m
    set m lput WA_Patch_State_Agent-Heading m
    set m lput WA_Patch_State_Agent-Speed m
   ask myself [set GA_Perceive_Visual-Scan lput m GA_Perceive_Visual-Scan]
    if ((pcolor = green) or (pcolor = grey)) [ (set i 41) ]
  ]
  set i (i + 1)
]

set i 1

while [i <= 40] [
  ask patch-left-and-ahead 15 i [ ; determines the patches left by 15 degrees in front of the Agent
    let h (remainder int ((towards myself) + 180) 360)
    let m []
    set m lput pcolor m
    set m lput h m
    set m lput i m
    set m lput pxcor m
    set m lput pycor m
    set m lput WA_Patch_State_Agent-Number m
    set m lput WA_Patch_State_Agent-Heading m
    set m lput WA_Patch_State_Agent-Speed m
    ask myself [set GA_Perceive_Visual-Scan lput m GA_Perceive_Visual-Scan]
    if ((pcolor = green) or (pcolor = grey)) [ (set i 41) ]
  ]
  set i (i + 1)
]

set i 1

while [i <= 40] [
  ask patch-left-and-ahead 17.5 i [ ; determines the patches left by 5 degrees in front of the Agent
    let h (remainder int ((towards myself) + 180) 360)
    let m []
    set m lput pcolor m
    set m lput h m
    set m lput i m
    set m lput pxcor m
    set m lput pycor m
    set m lput WA_Patch_State_Agent-Number m
    set m lput WA_Patch_State_Agent-Heading m
    set m lput WA_Patch_State_Agent-Speed m
    ask myself [set GA_Perceive_Visual-Scan lput m GA_Perceive_Visual-Scan]
    if ((pcolor = green) or (pcolor = grey)) [ (set i 41) ]
  ]
  set i (i + 1)
]

set i 1

while [i <= 40] [
  ask patch-left-and-ahead 20 i [ ; determines the patches left by 45 degrees in front of the Agent
    let h (remainder int ((towards myself) + 180) 360)
    let m []
    set m lput pcolor m
    set m lput h m
    set m lput i m
    set m lput pxcor m
    set m lput pycor m
    set m lput WA_Patch_State_Agent-Number m
    set m lput WA_Patch_State_Agent-Heading m
    set m lput WA_Patch_State_Agent-Speed m
    ask myself [set GA_Perceive_Visual-Scan lput m GA_Perceive_Visual-Scan]
    if ((pcolor = green) or (pcolor = grey)) [ (set i 41) ]
  ]
  set i (i + 1)
]

set i 1
```

```
while [i <= 40] [
  ask patch-left-and-ahead 22.5 i [ ; determines the patches left by 5 degrees in front of the Agent
    let h (remainder int ((towards myself) + 180) 360)
    let m []
    set m lput pcolor m
    set m lput h m
    set m lput i m
    set m lput pxcor m
    set m lput pycor m
    set m lput WA_Patch_State_Agent-Number m
    set m lput WA_Patch_State_Agent-Heading m
    set m lput WA_Patch_State_Agent-Speed m
    ask myself [set GA_Perceive_Visual-Scan lput m GA_Perceive_Visual-Scan]
    if ((pcolor = green) or (pcolor = grey)) [ (set i 41) ]
  ]
  set i (i + 1)
]

set i 1

while [i <= 40] [
  ask patch-left-and-ahead 25 i [ ; determines the patches left by 25 degrees in front of the Agent
    let h (remainder int ((towards myself) + 180) 360)
    let m []
    set m lput pcolor m
    set m lput h m
    set m lput i m
    set m lput pxcor m
    set m lput pycor m
    set m lput WA_Patch_State_Agent-Number m
    set m lput WA_Patch_State_Agent-Heading m
    set m lput WA_Patch_State_Agent-Speed m
    ask myself [set GA_Perceive_Visual-Scan lput m GA_Perceive_Visual-Scan]
    if ((pcolor = green) or (pcolor = grey)) [ (set i 41) ]
  ]
  set i (i + 1)
]

set i 1

while [i <= 35] [
  ask patch-left-and-ahead 30 i [ ; determines the patches left by 45 degrees in front of the Agent
    let h (remainder int ((towards myself) + 180) 360)
    let m []
    set m lput pcolor m
    set m lput h m
    set m lput i m
    set m lput pxcor m
    set m lput pycor m
    set m lput WA_Patch_State_Agent-Number m
    set m lput WA_Patch_State_Agent-Heading m
    set m lput WA_Patch_State_Agent-Speed m
    ask myself [set GA_Perceive_Visual-Scan lput m GA_Perceive_Visual-Scan]
    if ((pcolor = green) or (pcolor = grey)) [ (set i 36) ]
  ]
  set i (i + 1)
]

set i 1

while [i <= 30] [
  ask patch-left-and-ahead 35 i [ ; determines the patches left by 35 degrees in front of the Agent
    let h (remainder int ((towards myself) + 180) 360)
    let m []
    set m lput pcolor m
    set m lput h m
    set m lput i m
    set m lput pxcor m
    set m lput pycor m
    set m lput WA_Patch_State_Agent-Number m
    set m lput WA_Patch_State_Agent-Heading m
    set m lput WA_Patch_State_Agent-Speed m
    ask myself [set GA_Perceive_Visual-Scan lput m GA_Perceive_Visual-Scan]
    if ((pcolor = green) or (pcolor = grey)) [ (set i 31) ]
  ]
  set i (i + 1)
]

set i 1

while [i <= 25] [
  ask patch-left-and-ahead 40 i [ ; determines the patches left by 45 degrees in front of the Agent
    let h (remainder int ((towards myself) + 180) 360)
    let m []
    set m lput pcolor m
    set m lput h m
    set m lput i m
    set m lput pxcor m
    set m lput pycor m
    set m lput WA_Patch_State_Agent-Number m
    set m lput WA_Patch_State_Agent-Heading m
    set m lput WA_Patch_State_Agent-Speed m
```

```
    ask myself [set GA_Perceive_Visual-Scan lput m GA_Perceive_Visual-Scan]
    if ((pcolor = green) or (pcolor = grey)) [ (set i 26) ]
  ]
  set i (i + 1)
]

set i 1

while [i <= 25] [
  ask patch-left-and-ahead 45 i [ ; determines the patches left by 45 degrees in front of the Agent
    let h (remainder int ((towards myself) + 180) 360)
    let m []
    set m lput pcolor m
    set m lput h m
    set m lput i m
    set m lput pxcor m
    set m lput pycor m
    set m lput WA_Patch_State_Agent-Number m
    set m lput WA_Patch_State_Agent-Heading m
    set m lput WA_Patch_State_Agent-Speed m
    ask myself [set GA_Perceive_Visual-Scan lput m GA_Perceive_Visual-Scan]
    if ((pcolor = green) or (pcolor = grey)) [ (set i 26) ]
  ]
  set i (i + 1)
]

set i 1

while [i <= 20] [
  ask patch-left-and-ahead 55 i [ ; determines the patches left by 45 degrees in front of the Agent
    let h (remainder int ((towards myself) + 180) 360)
    let m []
    set m lput pcolor m
    set m lput h m
    set m lput i m
    set m lput pxcor m
    set m lput pycor m
    set m lput WA_Patch_State_Agent-Number m
    set m lput WA_Patch_State_Agent-Heading m
    set m lput WA_Patch_State_Agent-Speed m
    ask myself [set GA_Perceive_Visual-Scan lput m GA_Perceive_Visual-Scan]
    if ((pcolor = green) or (pcolor = grey)) [ (set i 21) ]
  ]
  set i (i + 1)
]

set i 1

while [i <= 20] [
  ask patch-left-and-ahead 65 i [ ; determines the patches left by 45 degrees in front of the Agent
    let h (remainder int ((towards myself) + 180) 360)
    let m []
    set m lput pcolor m
    set m lput h m
    set m lput i m
    set m lput pxcor m
    set m lput pycor m
    set m lput WA_Patch_State_Agent-Number m
    set m lput WA_Patch_State_Agent-Heading m
    set m lput WA_Patch_State_Agent-Speed m
    ask myself [set GA_Perceive_Visual-Scan lput m GA_Perceive_Visual-Scan]
    if ((pcolor = green) or (pcolor = grey)) [ (set i 21) ]
  ]
  set i (i + 1)
]

set i 1

while [i <= 20] [
  ask patch-left-and-ahead 75 i [ ; determines the patches left by 45 degrees in front of the Agent
    let h (remainder int ((towards myself) + 180) 360)
    let m []
    set m lput pcolor m
    set m lput h m
    set m lput i m
    set m lput pxcor m
    set m lput pycor m
    set m lput WA_Patch_State_Agent-Number m
    set m lput WA_Patch_State_Agent-Heading m
    set m lput WA_Patch_State_Agent-Speed m
    ask myself [set GA_Perceive_Visual-Scan lput m GA_Perceive_Visual-Scan]
    if ((pcolor = green) or (pcolor = grey)) [ (set i 21) ]
  ]
  set i (i + 1)
]

set i 1

while [i <= 20] [
  ask patch-left-and-ahead 85 i [ ; determines the patches left by 45 degrees in front of the Agent
    let h (remainder int ((towards myself) + 180) 360)
    let m []
    set m lput pcolor m
```

```
    set m lput h m
    set m lput i m
    set m lput pxcor m
    set m lput pycor m
    set m lput WA_Patch_State_Agent-Number m
    set m lput WA_Patch_State_Agent-Heading m
    set m lput WA_Patch_State_Agent-Speed m
    ask myself [set GA_Perceive_Visual-Scan lput m GA_Perceive_Visual-Scan]
    if ((pcolor = green) or (pcolor = grey)) [ (set i 21) ]
   ]
   set i (i + 1)
  ]

;;;;; Right Scan;;;;;

  set i 1

  while [i <= 40] [
   ask patch-right-and-ahead 2.5 i [ ; determines the patches right by 5 degrees in front of the Agent
    let h (remainder int ((towards myself) + 180) 360)
    let m []
    set m lput pcolor m
    set m lput h m
    set m lput i m
    set m lput pxcor m
    set m lput pycor m
    set m lput WA_Patch_State_Agent-Number m
    set m lput WA_Patch_State_Agent-Heading m
    set m lput WA_Patch_State_Agent-Speed m
    ask myself [set GA_Perceive_Visual-Scan lput m GA_Perceive_Visual-Scan]
    if ((pcolor = green) or (pcolor = grey)) [ (set i 41) ]
   ]
   set i (i + 1)
  ]

  set i 1

  while [i <= 40] [
   ask patch-right-and-ahead 5 i [ ; determines the patches right by 5 degrees in front of the Agent
    let h (remainder int ((towards myself) + 180) 360)
    let m []
    set m lput pcolor m
    set m lput h m
    set m lput i m
    set m lput pxcor m
    set m lput pycor m
    set m lput WA_Patch_State_Agent-Number m
    set m lput WA_Patch_State_Agent-Heading m
    set m lput WA_Patch_State_Agent-Speed m
    ask myself [set GA_Perceive_Visual-Scan lput m GA_Perceive_Visual-Scan]
    if ((pcolor = green) or (pcolor = grey)) [ (set i 41) ]
   ]
   set i (i + 1)
  ]

  set i 1

  while [i <= 40] [
   ask patch-right-and-ahead 7.5 i [ ; determines the patches right by 5 degrees in front of the Agent
    let h (remainder int ((towards myself) + 180) 360)
    let m []
    set m lput pcolor m
    set m lput h m
    set m lput i m
    set m lput pxcor m
    set m lput pycor m
    set m lput WA_Patch_State_Agent-Number m
    set m lput WA_Patch_State_Agent-Heading m
    set m lput WA_Patch_State_Agent-Speed m
    ask myself [set GA_Perceive_Visual-Scan lput m GA_Perceive_Visual-Scan]
    if ((pcolor = green) or (pcolor = grey)) [ (set i 41) ]
   ]
   set i (i + 1)
  ]

  set i 1

  while [i <= 40] [
   ask patch-right-and-ahead 10 i [ ; determines the patches right by 45 degrees in front of the Agent
    let h (remainder int ((towards myself) + 180) 360)
    let m []
    set m lput pcolor m
    set m lput h m
    set m lput i m
    set m lput pxcor m
    set m lput pycor m
    set m lput WA_Patch_State_Agent-Number m
    set m lput WA_Patch_State_Agent-Heading m
    set m lput WA_Patch_State_Agent-Speed m
    ask myself [set GA_Perceive_Visual-Scan lput m GA_Perceive_Visual-Scan]
    if ((pcolor = green) or (pcolor = grey)) [ (set i 41) ]
   ]
   set i (i + 1)
```

```
]

set i 1

while [i <= 40] [
  ask patch-right-and-ahead 12.5 i [ ; determines the patches right by 5 degrees in front of the Agent
    let h (remainder int ((towards myself) + 180) 360)
    let m []
    set m lput pcolor m
    set m lput h m
    set m lput i m
    set m lput pxcor m
    set m lput pycor m
    set m lput WA_Patch_State_Agent-Number m
    set m lput WA_Patch_State_Agent-Heading m
    set m lput WA_Patch_State_Agent-Speed m
    ask myself [set GA_Perceive_Visual-Scan lput m GA_Perceive_Visual-Scan]
    if ((pcolor = green) or (pcolor = grey)) [ (set i 41) ]
 ]
  set i (i + 1)
]

set i 1

while [i <= 40] [
  ask patch-right-and-ahead 15 i [ ; determines the patches right by 15 degrees in front of the Agent
    let h (remainder int ((towards myself) + 180) 360)
    let m []
    set m lput pcolor m
    set m lput h m
    set m lput i m
    set m lput pxcor m
    set m lput pycor m
    set m lput WA_Patch_State_Agent-Number m
    set m lput WA_Patch_State_Agent-Heading m
    set m lput WA_Patch_State_Agent-Speed m
    ask myself [set GA_Perceive_Visual-Scan lput m GA_Perceive_Visual-Scan]
    if ((pcolor = green) or (pcolor = grey)) [ (set i 41) ]
  ]
  set i (i + 1)
]

set i 1

while [i <= 40] [
  ask patch-right-and-ahead 17.5 i [ ; determines the patches right by 5 degrees in front of the Agent
    let h (remainder int ((towards myself) + 180) 360)
    let m []
    set m lput pcolor m
    set m lput h m
    set m lput i m
    set m lput pxcor m
    set m lput pycor m
    set m lput WA_Patch_State_Agent-Number m
    set m lput WA_Patch_State_Agent-Heading m
    set m lput WA_Patch_State_Agent-Speed m
    ask myself [set GA_Perceive_Visual-Scan lput m GA_Perceive_Visual-Scan]
    if ((pcolor = green) or (pcolor = grey)) [ (set i 41) ]
  ]
  set i (i + 1)
]

set i 1

while [i <= 40] [
  ask patch-right-and-ahead 20 i [ ; determines the patches right by 45 degrees in front of the Agent
    let h (remainder int ((towards myself) + 180) 360)
    let m []
    set m lput pcolor m
    set m lput h m
    set m lput i m
    set m lput pxcor m
    set m lput pycor m
    set m lput WA_Patch_State_Agent-Number m
    set m lput WA_Patch_State_Agent-Heading m
    set m lput WA_Patch_State_Agent-Speed m
    ask myself [set GA_Perceive_Visual-Scan lput m GA_Perceive_Visual-Scan]
    if ((pcolor = green) or (pcolor = grey)) [ (set i 41) ]
  ]
  set i (i + 1)
]

set i 1

while [i <= 40] [
  ask patch-right-and-ahead 22.5 i [ ; determines the patches right by 5 degrees in front of the Agent
    let h (remainder int ((towards myself) + 180) 360)
    let m []
    set m lput pcolor m
    set m lput h m
    set m lput i m
    set m lput pxcor m
    set m lput pycor m
```

```
    set m lput WA_Patch_State_Agent-Number m
    set m lput WA_Patch_State_Agent-Heading m
    set m lput WA_Patch_State_Agent-Speed m
    ask myself [set GA_Perceive_Visual-Scan lput m GA_Perceive_Visual-Scan]
    if ((pcolor = green) or (pcolor = grey)) [ (set i 41) ]
  ]
  set i (i + 1)
]

set i 1

while [i <= 40] [
  ask patch-right-and-ahead 25 i [ ; determines the patches right by 25 degrees in front of the Agent
    let h (remainder int ((towards myself) + 180) 360)
    let m []
    set m lput pcolor m
    set m lput h m
    set m lput i m
    set m lput pxcor m
    set m lput pycor m
    set m lput WA_Patch_State_Agent-Number m
    set m lput WA_Patch_State_Agent-Heading m
    set m lput WA_Patch_State_Agent-Speed m
    ask myself [set GA_Perceive_Visual-Scan lput m GA_Perceive_Visual-Scan]
    if ((pcolor = green) or (pcolor = grey)) [ (set i 41) ]
  ]
  set i (i + 1)
]

set i 1

while [i <= 35] [
  ask patch-right-and-ahead 30 i [ ; determines the patches right by 45 degrees in front of the Agent
    let h (remainder int ((towards myself) + 180) 360)
    let m []
    set m lput pcolor m
    set m lput h m
    set m lput i m
    set m lput pxcor m
    set m lput pycor m
    set m lput WA_Patch_State_Agent-Number m
    set m lput WA_Patch_State_Agent-Heading m
    set m lput WA_Patch_State_Agent-Speed m
    ask myself [set GA_Perceive_Visual-Scan lput m GA_Perceive_Visual-Scan]
    if ((pcolor = green) or (pcolor = grey)) [ (set i 36) ]
  ]
  set i (i + 1)
]

set i 1

while [i <= 30] [
  ask patch-right-and-ahead 35 i [ ; determines the patches right by 35 degrees in front of the Agent
    let h (remainder int ((towards myself) + 180) 360)
    let m []
    set m lput pcolor m
    set m lput h m
    set m lput i m
    set m lput pxcor m
    set m lput pycor m
    set m lput WA_Patch_State_Agent-Number m
    set m lput WA_Patch_State_Agent-Heading m
    set m lput WA_Patch_State_Agent-Speed m
    ask myself [set GA_Perceive_Visual-Scan lput m GA_Perceive_Visual-Scan]
    if ((pcolor = green) or (pcolor = grey)) [ (set i 31) ]
  ]
  set i (i + 1)
]

set i 1

while [i <= 25] [
  ask patch-right-and-ahead 40 i [ ; determines the patches right by 45 degrees in front of the Agent
    let h (remainder int ((towards myself) + 180) 360)
    let m []
    set m lput pcolor m
    set m lput h m
    set m lput i m
    set m lput pxcor m
    set m lput pycor m
    set m lput WA_Patch_State_Agent-Number m
    set m lput WA_Patch_State_Agent-Heading m
    set m lput WA_Patch_State_Agent-Speed m
    ask myself [set GA_Perceive_Visual-Scan lput m GA_Perceive_Visual-Scan]
    if ((pcolor = green) or (pcolor = grey)) [ (set i 26) ]
  ]
  set i (i + 1)
]

set i 1

while [i <= 25] [
  ask patch-right-and-ahead 45 i [ ; determines the patches right by 45 degrees in front of the Agent
```

```
    let h (remainder int ((towards myself) + 180) 360)
    let m []
    set m lput pcolor m
    set m lput h m
    set m lput i m
    set m lput pxcor m
    set m lput pycor m
    set m lput WA_Patch_State_Agent-Number m
    set m lput WA_Patch_State_Agent-Heading m
    set m lput WA_Patch_State_Agent-Speed m
    ask myself [set GA_Perceive_Visual-Scan lput m GA_Perceive_Visual-Scan]
    if ((pcolor = green) or (pcolor = grey)) [ (set i 26) ]
  ]
  set i (i + 1)
 ]

set i 1

 while [i <= 20] [
   ask patch-right-and-ahead 55 i [ ; determines the patches right by 45 degrees in front of the Agent
     let h (remainder int ((towards myself) + 180) 360)
     let m []
     set m lput pcolor m
     set m lput h m
     set m lput i m
     set m lput pxcor m
     set m lput pycor m
     set m lput WA_Patch_State_Agent-Number m
     set m lput WA_Patch_State_Agent-Heading m
     set m lput WA_Patch_State_Agent-Speed m
     ask myself [set GA_Perceive_Visual-Scan lput m GA_Perceive_Visual-Scan]
     if ((pcolor = green) or (pcolor = grey)) [ (set i 21) ]
   ]
   set i (i + 1)
 ]

set i 1

 while [i <= 20] [
   ask patch-right-and-ahead 65 i [ ; determines the patches right by 45 degrees in front of the Agent
     let h (remainder int ((towards myself) + 180) 360)
     let m []
     set m lput pcolor m
     set m lput h m
     set m lput i m
     set m lput pxcor m
     set m lput pycor m
     set m lput WA_Patch_State_Agent-Number m
     set m lput WA_Patch_State_Agent-Heading m
     set m lput WA_Patch_State_Agent-Speed m
     ask myself [set GA_Perceive_Visual-Scan lput m GA_Perceive_Visual-Scan]
    if ((pcolor = green) or (pcolor = grey)) [ (set i 21) ]
   ]
   set i (i + 1)
 ]

set i 1

while [i <= 20] [
   ask patch-right-and-ahead 75 i [ ; determines the patches right by 45 degrees in front of the Agent
     let h (remainder int ((towards myself) + 180) 360)
     let m []
     set m lput pcolor m
     set m lput h m
     set m lput i m
     set m lput pxcor m
     set m lput pycor m
     set m lput WA_Patch_State_Agent-Number m
     set m lput WA_Patch_State_Agent-Heading m
     set m lput WA_Patch_State_Agent-Speed m
     ask myself [set GA_Perceive_Visual-Scan lput m GA_Perceive_Visual-Scan]
     if ((pcolor = green) or (pcolor = grey)) [ (set i 25) ]
   ]
   set i (i + 1)
 ]

set i 1

 while [i <= 20] [
   ask patch-right-and-ahead 85 i [ ; determines the patches right by 45 degrees in front of the Agent
     let h (remainder int ((towards myself) + 180) 360)
     let m []
     set m lput pcolor m
     set m lput h m
     set m lput i m
     set m lput pxcor m
     set m lput pycor m
     set m lput WA_Patch_State_Agent-Number m
     set m lput WA_Patch_State_Agent-Heading m
     set m lput WA_Patch_State_Agent-Speed m
     ask myself [set GA_Perceive_Visual-Scan lput m GA_Perceive_Visual-Scan]
     if ((pcolor = green) or (pcolor = grey)) [ (set i 25) ]
   ]
```

```
    set i (i + 1)
  ]
end

;;;;;The following code was an attempt to simplify the visual perception using a sub-procedure;;;;;;;;;;;;;;;;
 ;set GA_State_Scan-Distance 1

 ; while [GA_State_Scan-Distance <= 20] [
 ;   ask patch-right-and-ahead 85 GA_State_Scan-Distance [ ; determines the patches right by 45 degrees in front of the Agent
 ;     HA_perception_visual-scan_subcode
 ;  ]
 ;   set GA_State_Scan-Distance (GA_State_Scan-Distance + 1)
;  ]
;end

;to HA_perception_visual-scan_subcode
;  let h (remainder int ((towards myself) + 180) 360)
;  let m []
;  set m lput pcolor m
;  set m lput h m
; set m lput GA_State_Scan-Distance m
;  set m lput pxcor m
 ; set m lput pycor m
;  set m lput WA_Patch_State_Agent-Number m
;  set m lput WA_Patch_State_Agent-Heading m
;  set m lput WA_Patch_State_Agent-Speed m
;  ask myself [set GA_Perceive_Visual-Scan lput m GA_Perceive_Visual-Scan]
;  if ((pcolor = green) or (pcolor = grey)) [ (set GA_State_Scan-Distance 25) ]
;end
```