Designing a realistic virtual bumblebee

Tim Marsden

Worcester Polytechnic Institute

Advisors:

Dr. Robert J Gegear

Dr. Elizabeth Ryder

Table of Contents

Figures

Tables

Abstract

Optimal Foraging Theory is a set of mathematical models used in the field of behavioral ecology to predict how animals should weigh foraging costs and benefits in order to maximize their food intake. One popular model, referred to as the Optimal Diet Model (ODM), focuses on how individuals should respond to variation in food quality in order to optimize food selection. The main prediction of the ODM is that low quality food items should only be accepted when higher quality items are encountered below a predicted threshold. Yet, many empirical studies have found that animals still include low quality items in their diet above such thresholds, indicating a sub-optimal foraging strategy. Here, we test the hypothesis that such 'partial preferences' are produced as a consequence of incomplete information on prey distributions resulting from memory limitations. To test this hypothesis, we used agent-based modeling in NetLogo to create a model of flower choice behavior in a virtual bumblebee forager (SimBee). We program virtual bee foragers with an adaptive decision-making algorithm based on the classic ODM, which we have modified to include memory. Our results show that the probability of correctly rejecting a low quality food item increases with memory size, suggesting that memory limitations play a significant role in driving partial preferences. We discuss the implications of this finding and further applications of our SimBee model in research and educational contexts.

Introduction

Animals routinely face the problem of deciding how and what to eat in complex foraging environments. To gain a better understanding of how animals adaptively address these foraging-related problems, behavioral researchers have developed numerous optimization models derived from the field of economics, collectively referred to as Optimal Foraging Theory (OFT). The central assumption of OFT is that natural selection has shaped the behavioral system of animals to weigh foraging costs and benefits and adopt foraging strategies that maximize their net rate of energy gain and therefore their probability of survival (Darwinian fitness) [Charnov, 1976]. Thus, each model predicts how animals should respond in a particular foraging environment within a given specific set of constraints. Such predictions are then tested empirically to determine whether foraging decisions in a particular floral environment are optimal [Zhang, 2014; Pyke, 1984].

One of the most widely used models within the OFT framework deals with the problem of diet selection. In this 'prey choice' model, foragers randomly encounter prey items that vary in quality and must decide whether to consume the item or reject it and continue searching for an alternative item [Sih, 2001]. This scenario assumes that animals are 'all-knowing' and therefore have all of the prey information needed to make the optimal decision. Given this assumption, the diet model predicts that animals will develop a foraging strategy that always rejects a lower quality prey item if the abundance of a higher quality item is above a particular threshold [Holt, 2013; Gonzales-Varo, 2013; Sih, 2013; Beecher, 2014]. Yet, most empirical tests of the prey choice model have found that animals still accept low quality items to some extent even when high quality items are abundant, a phenomenon known as partial preference. Although there have been several explanations for partial preferences proposed over the years, its cause remains unknown [Arundel, 2013; Couvillon, 2015]. Here, we describe a series of virtual experiments designed to investigate the role of memory limitations in driving partial preferences in bumblebee foragers.

Bumblebees are an ideal behavioral system to study mechanisms of adaptive prey choice. As social insects, they form colonies comprised of a single queen and up to several hundred workers. A small subset of workers called 'foragers' have the sole task of finding and collecting food for the colony in the form of floral nectar and pollen rewards [Dyer, 2004; Goulson, 2009]. Each forager faces the daunting task of maximizing nectar delivery to the colony in habitats containing multiple flower types that vary in their reward quality [Dyer, 2004; Das, 2013; Lourenco, 2015]. Foragers are not pre-programmed with information on floral reward quality; rather, they must learn and remember the reward level and sensory cues (color, odor, shape) associated with each type of flower that they visit [Muth, 2015; Faruq, 2013]. Memory therefore plays a fundamental role in bumblebee survival and reproductive success [Muth, 2015; Zhang, 2014].

To investigate the relationship between memory capacity, floral choice behavior, and reward intake, we used the agent-based modelling (ABM) software NetLogo to develop a virtual bumblebee-plant system called 'SimBee'. Many computational simulations rely largely on equation-based or analytical modeling, which does not allow for model flexibility at the individual level [Castello, 2013; Liu, 2015]. In contrast, agent-based modelling allows variability by defining the behaviors of individuals with their own set of decision rules, allowing the user to study interconnections between the agents themselves, as well as modelling collective behaviors [Helbing, 2012; Janssen, 2014; Mills, 2015]. Additional characteristics of ABM that are important with respect to biological applications: the modular structure is useful by allowing the modification of existing agent rules or the addition of new agents into pre-existing model structure, emergent properties of higher level relationships are the result of individual agents' local interactions, and abstraction of new behavior information can occur when there is incomplete knowledge about the target biological process based on known information [Politopoulos, 2007].

Other investigators have developed agent-based models simulating pollinator behavior. In particular, the 'A-bees-see' model simulates the differences in the scanning approaches of honeybees and bumblebees to targeting flowers, to develop a better understanding of the evolution of these behaviors [Bukovac, 2013]. The BEEHAVE model simulates multiple environmental stressors on honeybee colonies [Becher, 2014], while the EcoSimInGrid model explores the effects of multiple pollinators on plant communities [Qu, 2014]. Our SimBee model focuses on the behavior of an individual virtual bumblebee, which is programmed with the ability to learn the properties of a simulated floral environment and then use information stored in memory to make foraging decisions based on mathematical components contained in the classic prey selection model. Our long-term goal is to observe the collective impact of individual decisions using this foraging model on population dynamics over time.

In the work presented here, we describe the SimBee model, and use it to study the effects of limited memory capacity on behavioral choices of an individual bee interacting with varied floral environments. We show that partial preferences emerge from the optimal diet model as a natural consequence of bees with limited knowledge interacting with a random floral environment. We further show that how the virtual bee calculates the search time for a rewarding floral type has a large impact on its behavioral choices. Finally, we describe the ability of SimBee to generate environments that can be used to verify model predictions experimentally.

Methods

SimBee Model Overview, Design concepts, and Details (ODD)

The ODD is a standard method of describing ABM models [Grimm, 2010].
A complete description for each subroutine process and is located in the Code Structure section (Appendix 1), a detailed flowchart of the decision-making process is located in the results section (Figure 4), and the actual Netlogo program with comment descriptions can be found in Appendix 2.

Purpose

This model is designed to serve as an accurate representation of the basic foraging behavior of bees. It includes the behaviors that are involved in the bees collecting nectar, and bringing it back to the hive. The bee 'decides' which flowers to visit based on the optimal diet model (ODM) that we have modified to include a learning and memory component. The reason for creating this model is to provide a more realistic and consistent base upon which more involved models can be built, and test how individual bee behavior has been shaped by natural adaptively respond in heterogeneous floral environments.

Entities, State Variables, and Scales

The model has three different individuals: bees, flowers, and hives. The state variables for the agent types are summarized in Table 1. There are no special units or environment in this simulation.

Table 1: State Variables for SimBee Agents (ML = memory list)

Hive	Flower	Bee
Nectar Accumulated	Nectar Content	Location
	Handling Time	Heading
		Detected Flowers
		On Flower?
		Last Flower Visited
		Nectar Held
		Next Flower
		Time of Last Blue flower visited
		Time of Last Yellow flower visited
		Sample Mode?
		ML yellow handling time
		ML blue handling time
		ML yellow search time
		ML blue search time
		ML yellow energy content
		ML blue energy content

Process overview and Scheduling
Each tick, the bee moves forward one patch in a random direction (see stochasticity below) on where the bee is, it calls different functions. Each tick, each of the three breeds updates their variable values to account for nectar transfer if it occurs. Each tick, bees store and adjust information in memory lists if applicable. In this simulation, 10 ticks are equivalent to 1 second. A bee is expected to remain on a flower for the flower's handling time, default is 2 seconds (20 ticks). A bee can hold up to 800 units of nectar by default, and since the default nectar amount for blue flowers is 40ul and yellow flowers is 10ul, the bee is expected to visit between 60 and 120 flowers before returning to hive. (20 if all blue visits, 80 if all yellow visits)

Design Concepts

Basic principles:
SimBee models continuous foraging; there is no day/night cycle. When a hive is present and the bee has the maximum amount of nectar that they can hold, they bring it back to the hive. If no hive is present, the bee will collect nectar indefinitely until simulation end.

Objectives:
In this model, the bees' objective is to collect nectar, and if a hive is present, to bring it back to its hive.

Learning:
The bees remember the handling times, energy content, and search times of the past flowers encountered. The number of past flowers remembered (memory length) is user-defined, ranging from 1 to 1000.

Prediction:
The bees change whether their next flower encounter will be accepted or not based on the information from past flower visits stored in memory lists, using the ODM equation: $S1 > ((E1*h2) / E2) - h1$. The right side of this equation is referred to as the threshold.
S1 = average search time for past blue flowers
E1 = average energy content for past blue flowers
E2 = average energy content for past yellow flowers
h1 = average handling time for past blue flowers
h2 = average handling time for past yellow flowers
If S1 > threshold, the bee will accept both flower types if encountered. If S1 < threshold, the bee will accept only blue flowers if encountered.

Sensing:
The bees have a cone of sight, which they use to see nearby flowers in the direction in which they are heading. They are able to select a random flower within this cone

of detection. Bees also know the location of the last flower visited so they don't immediately return to it.

Interaction:
When the bee is on the same patch as the flower it has chosen to encounter next (from the mentioned above detected flowers) the bee "lands" on that flower and gathers all of its nectar, asking the flower to change its energy value to 0, and set its occupied? variable to true. The bee stays on that patch for the handling time corresponding to the selected flower, and the nectar quantity transferring from flower to bee. Once the bee leaves this location, the flower detects no bees are present and changes its occupied? Boolean back to false.

Stochasticity:
The direction in which the bee travels is based on pseudorandom numbers, the pseudorandom number generator used by Netlogo always uses Java's "strict math" library and details can be found in the Netlogo User Manual [Wilensky, 1999]. If it has not detected a flower to encounter, the bee's heading is adjusted randomly between 30 degrees right to 30 degrees left. This is to emulate the natural 'drifting' flight patterns bees exhibit in real life. The flower chosen by bees to be next encountered is also a stochastic process because according to ODM, prey is randomly encountered before the acceptance decision occurs. This means that if the bee has more than one flower in the detected range, the selection is based on pseudorandom numbers.

Collectives:
The bees are all part of a colony and all colonies are part of the population. The activity of the individual contributes to how the whole population is doing. In this model each trial only has one bee active at a time.

Observation:
The data outputs needed to observe internal dynamics, solve the problem the model was designed for, as well as system-level behavior include: total nectar gathered, the total number of flowers encountered, the number of times the bee's average blue search time fluctuated relative to the threshold, The number of yellow flowers encountered, the number of yellow flowers accepted, the number of blue flowers encountered, the number of blue flowers accepted, and the timestamp of when sampling phase ended. Output of the bee's information stored in memory lists is also required to confirm certain aspects of bee behavior are correct, such as keeping search time and handling time mutually exclusive.
The tools needed to obtain these outputs include Excel and BehaviorSpace.

Initialization
Upon clicking the setup button, a number of blue flowers determined by the blue-flower-count slider are generated in random locations, and a number of yellow flowers determined by the yellow-flower-count slider are generated in random locations. These flowers have variables defined by the following sliders: blue-

handling-time, yellow-handling-time, blue-energy, yellow-energy. A bee is also generated, with memory length defined by the bee-memory-length slider.
If testing identical environments with varying bee memory lengths, the 'generate-worlds' button should first be used. This will output world files with the same user-defined flower variables, each with a differing bee memory length: 1000, 10, 5, 3, 2, 1. These worlds can be manually imported afterward using the 'setup-imported' button, or automatically using BehaviorSpace.

Input Data
If testing a specific environment, or a set of identical environments with varying bee memory lengths, world input files are needed (from the 'generate-worlds' button described above). These files are named g(number).csv, ranging from g0.csv to whatever the number of worlds generated ends at.

The Optimal Diet Model (Charnov and Orians, 1973)

The optimal diet model assumes that the predator encounters prey items of different types sequentially and randomly, and must decide whether to accept or reject the prey item once encountered in order to maximize rate of energetic gain. The profitability (P) of each prey item equals the energy content of the prey item divided by the handling time of the prey item (P = E/h). For our model, the handling time is the amount of time it takes the bee to consume all the available nectar from a flower. Search time and handling times are considered mutually exclusive events in the ODM, meaning the predator is always either searching or consuming prey, but never both.   In the SimBee model, only two prey types are considered. For the experiments conducted here, the more profitable prey type is blue flowers, which have an energy content of 40J and handling time of 2s, which equals a profitability of 20J/s. The less profitable prey type is yellow flowers, which have an energy content of 10J and handling time of 2s, which equals a profitability of 5J/s.

The bee must determine whether specialization (accepting the more profitable prey type) or generalization (accepting both prey types) will optimize the rate of energy gain, which is the total energy gained over the total time spent foraging, as shown in Figure 1. With both specialist (R1) and generalist (R1,2) rates calculated, the bee will specialize if R1 > R1,2, and generalize if R1 <= R1,2. Since the encounter rate is the inverse of the search time, this inequality reduces to Eqn 3 (Figure 1). Thus, the decision to specialize only depends on the search time for the more profitable prey type. If the blue flowers are plentiful (short search times), the bee shouldn't spend time accepting less profitable flowers; however, if blue flowers are scarce, the bee should accept both flower types. If the predator has complete knowledge of the parameters given in Figure 1, it should always choose either to specialize or generalize at all time (referred to as the 'all-or-none' rule for selecting the less profitable type); thus, the ODM predicts that animals should never show a partial preference.

Important variables:

E = energy per item

h = handling time per item

λ = encounter rate (# items / unit time)

T_s= total amount of time searching for prey items

(prey 1 = blue, prey 2 = yellow)

<u>Eqn1: Rate for specialist</u>

$= \lambda_1 T_s E_1 / (T_s + \lambda_1 T_s h_1)$

$R_1 = \lambda_1 E_1 / (1 + \lambda_1 h_1)$

<u>Eqn2: Rate for generalist</u>

$= (\lambda_1 T_s E_1 + \lambda_2 T_s E_2)/ (T_s + \lambda_1 T_s h_1 + \lambda_2 T_s h_2)$

$R_{1,2} = (\lambda_1 E_1 + \lambda_2 E_2) / (1 + \lambda_1 h_1 + \lambda_2 h_2)$

Specialize if $R_1 > R_{1,2}$ and generalize if R_1 < or = $R_{1,2}$

<u>Eqn3: Reduction to threshold equation</u>

$$\frac{\lambda 1 E1}{1 + \lambda 1 h1} > \frac{\lambda 1 E1 + \lambda 2 E2}{1 + \lambda 1 h1 + \lambda 2 h2} \implies \lambda 1 > \frac{E2}{E1h2 - E2h1}$$

$$S1 = \frac{1}{\lambda 1} \implies S1 < \frac{E1h2 - E2h1}{E2} \text{ (threshold)}$$

Figure 1: ODM Equations for specialization and generalization. For specializing, the total energy equals the product of the encounter rate (# of blue flowers encountered/time), Energy gained from each blue flower, and the total time spent searching. The total time spent foraging is search time added to the total time spent handling blue flowers, which equals the product of encounter rate, handling time required for each blue flower, and total time spent searching (Eqn1). For generalizing, the idea is similar except now the total energy is the sum of total yellow flower energy gained and total blue flower energy gained, and the total time includes total time spent handling yellow flowers and total time spent handling blue flowers (Eqn2). If Eqn1 > Eqn2, the bee should specialize; this inequality reduces to the threshold equation (Eqn3). If S1 < threshold, the bee should specialize.

Simulation experimental parameters

Unless stated otherwise, simulation experiments were performed in batch mode (Behavior Space in Netlogo) using the default settings described below. Bee memory length was one of the following settings: 1,000, 10, 5, 3, 2, or 1. The time required for each sampling period depended on this memory length. For example, if the memory length was 10, the bee would sample 10 blue flowers and 10 yellow flowers before entering forage mode. The forage time period was always 300,000 ticks after sampling ended, which is approximately 8 hours in real time (1 second = 10 ticks).

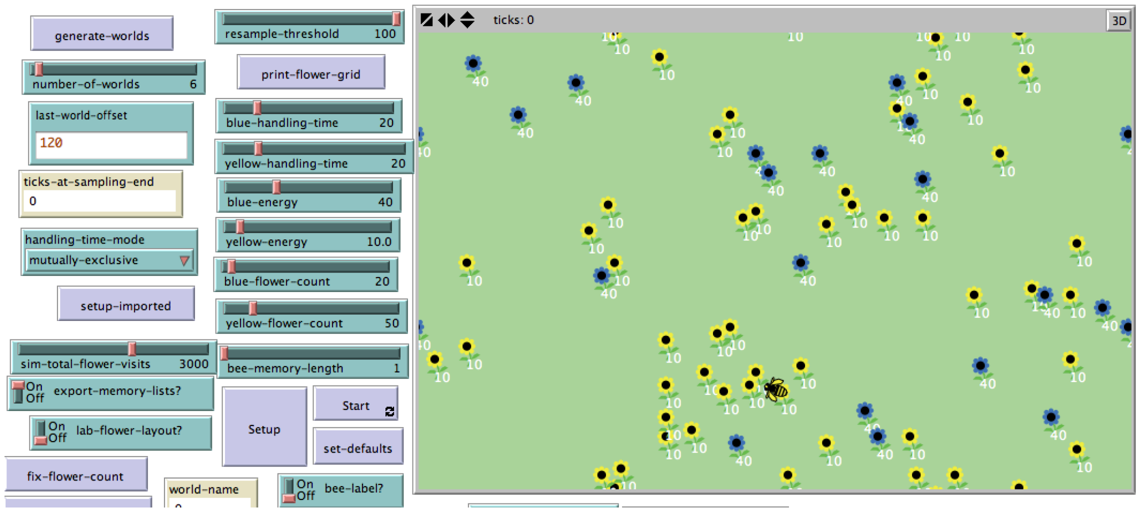
The more profitable flowers (prey1) were always blue, contained 40ul of nectar, and required 2 seconds of handling time. The less profitable flowers (prey2) were always yellow, contained 10ul of nectar, and required 2 seconds of handling time. Each flower instantly regenerates the stored nectar amount once a bee has harvested the flower and left the particular location. For generalized foraging environments, where both prey types are always accepted during foraging, the environment contained 10 blue and 50 yellow flowers, randomly placed within the landscape. For specialized foraging environments, when only the most profitable prey type is accepted during foraging, the environment contained 20 blue and 50 yellow flowers, randomly placed within the landscape.

Results

The SimBee model

The SimBee user interface is illustrated in Figure 2. Upon setup, the virtual environment, termed a 'world', is generated. A user-defined number of flowers is randomly distributed in the world for each flower type. Currently two types of flowers are allowed, depicted by blue and yellow flower icons. The user determines the energy content and handling times for each flower type. For experiments reported here, the blue flowers are always most profitable (energy content of 40J, handling time 2 seconds), and the yellow flowers are always least profitable (energy content of 10J, handling time 2 seconds). Flowers refill instantaneously after being accepted as prey by a bee.

The bee is placed at a random location in the environment. The bee moves around the simulation until it detects a flower. The bee detects adjacent flowers within its line of sight, using a user-defined view radius and arc. For experiments reported here, a view radius of 9 patches and an arc of 140° were used. (Figure 3). If there are multiple flowers within detection range, the bee will choose a random flower and travel to this location; this action is termed an encounter with the flower. The bee will accept the flower (take nectar from it) based on the optimal diet model strategy described below. If the bee accepts the flower, it will remain at the flower location for the handling time associated with that flower. The bee stores the search time and profitability from its most recent acceptances of each flower type, up to its memory capacity, which can range from 1 to 1000. The user can choose whether or not to include handling time in the search times for each flower type.



resample-threshold 100
number-of-worlds 6
last-world-offset 120
blue-handling-time 20
yellow-handling-time 20
ticks-at-sampling-end 0
blue-energy 40
handling-time-mode mutually-exclusive
yellow-energy 10.0
blue-flower-count 20
yellow-flower-count 50
sim-total-flower-visits 3000
bee-memory-length 1
On Off export-memory-lists?
On Off lab-flower-layout?
world-name
On Off bee-label?

Figure 2: SimBee user interface: The user can set model parameters using the sliders and inputs on the left, while observing the simulation in real time on the right.

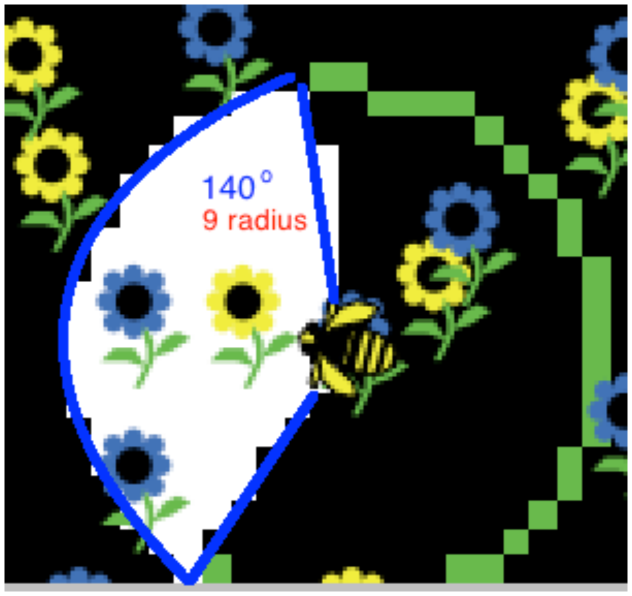


Figure 3: Bee detects flowers within a user-defined radius and angle. The green circle surrounding the bee demonstrates the distance the bee can detect from in relation to itself, while the white area visualizes the radius and angle the bee considers within its detection area.   Here, one yellow and two blue flowers are detected by the bee.

SimBee Procedure Overview

The SimBee model procedure overview is shown in Figure 4. After the interface is set up, an initial sampling mode is utilized by the bee in order to determine profitability of the available flower types. In sampling mode, the bee will always accept both prey types upon encountering them. Once the bee memory capacity is reached for both flower types, the bee switches to foraging mode. In our experiments, foraging mode is run for 300,000 steps (simulating approximately 8 hours, where each model step corresponds to 0.1 seconds).   During this time, each encounter decision to accept or reject a flower type depends on the threshold inequality derived from the Optimal Diet Model (See Methods, Fig. 1, Eqn 3). If the encounter is with a blue flower (most profitable), it is always accepted by the bee, and the flower's profitability and search time values are added to the bee memory. If the encounter is with a yellow flower (less profitable), the bee will accept the flower if the average blue flower search time is currently greater than the threshold calculation. Otherwise, the bee will reject this yellow flower and continue foraging.

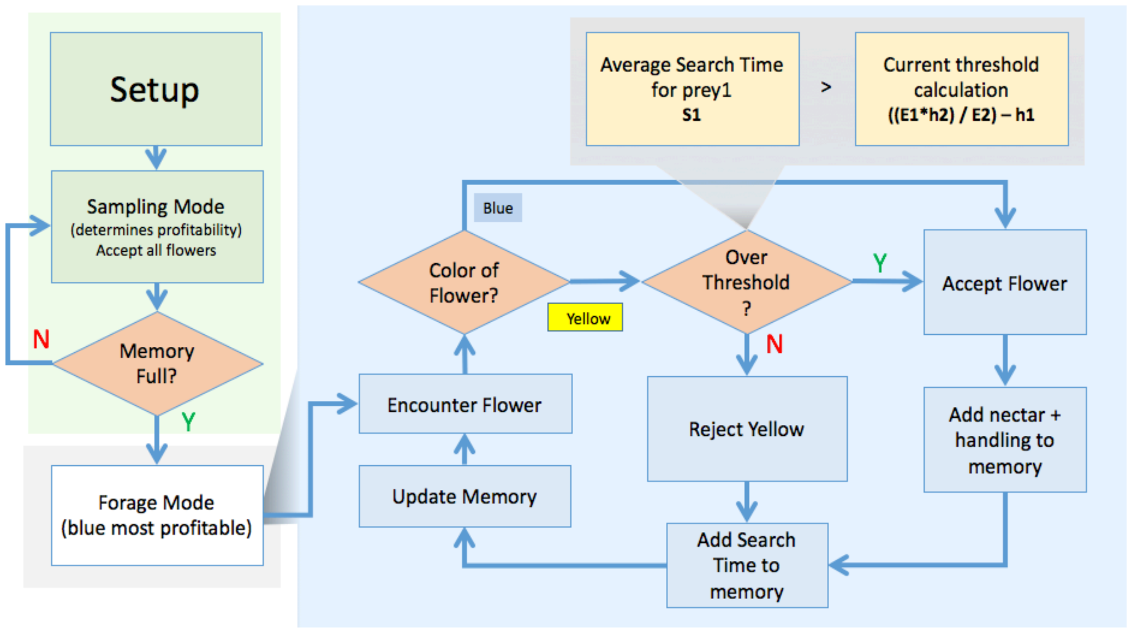


Figure 4: SimBee Procedure Overview. Upon setup, the bee enters sampling mode, where every flower encountered is accepted to determine profitability of flower types. Once a set number of flowers are accepted for each flower type, the bee will switch to forage mode, and use the information gathered during sampling mode to determine whether one of both flower types should be accepted. Upon flower encounter, if the flower type is blue (most profitable), it is automatically accepted, adds the information into its memory, and returns to searching for flowers. If the flower type is yellow (less profitable), it determines if the average search time for blue flowers is above or below the threshold. If below, it rejects the yellow flower and continues searching. If above, it accepts the yellow flower, adds the information to its memory, and returns to searching for flowers. This continues for a set interval of time (300k ticks, which is roughly 8 hours or one day of foraging).

Determining Specialized (S) and Generalized (G) environments

In order to test the behavioral effects of changes in memory, we first wanted to create environments in which a bee with complete knowledge of the environment (an 'all-knowing' bee) will either always specialize (S environment) or always generalize (G environment). For our simulated environments, we determined that a bee with memory length of 1000 was 'all-knowing', with essentially complete knowledge of the environment. We ran multiple trials of with bee memory length 1000 for a range of blue flower counts, and determined the environments in which bees always specialized or always generalized for every trial. By running 10 trials for environments with 50 yellow flowers and variable numbers of blue flowers, it was determined that 10 blue flowers and 50 yellow flowers always resulted in generalization while 20 blue flowers and 50 yellow flowers always resulted in specialization (Figure 5). For environments with intermediate numbers of blue flowers, the simulation sometimes generated G and sometimes S environments, due to the random placement of flowers within the environment. Thus we chose 10 and 20 blue flowers to define our G and S environments, respectively.

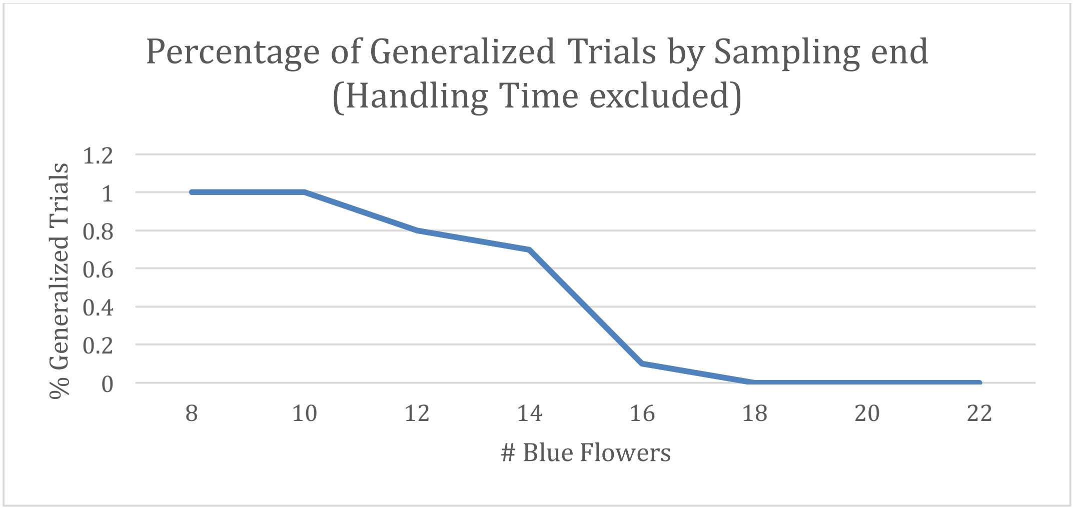
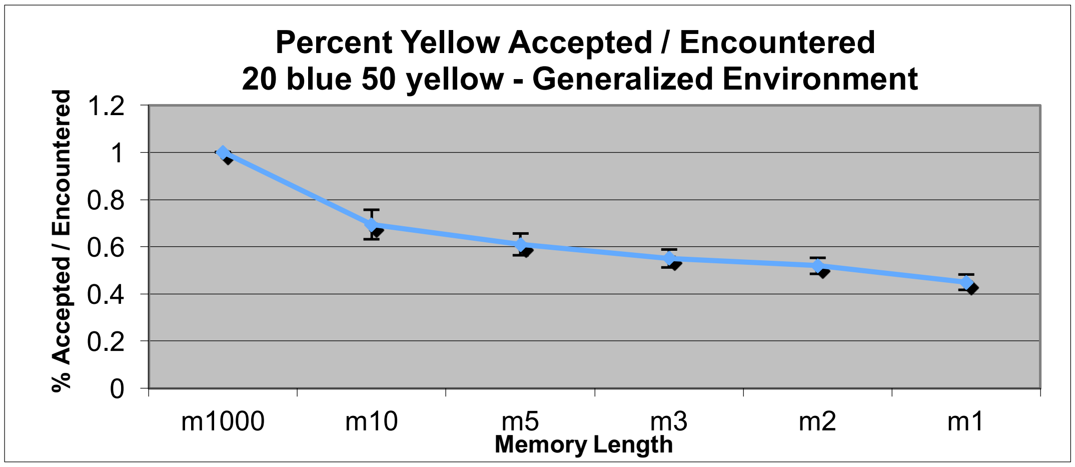


Figure 5: Determination of Specialized and Generalized Environments. 'All-knowing' bees were subjected to different worlds containing a specific number of blue flowers (8, 10, 12, etc.) and 50 yellow flowers. We ran 10 trials per number of blue flowers, with each trial containing a different distribution of each flower type. In these trials, search times for prey did not include handling times. A trial was considered to be 'generalized' if both prey items were selected over the simulation period.

Changing memory size affects bee behavior: Partial preference in ODM

In order to test whether memory strength affects behavioral choices, we ran the simulation with virtual bees of various memory lengths in the specialized and generalized environments defined above. We generated 10 random S environments and 10 random G environments. Within each environment, we tested a bee for each memory length (1000, 10, 5, 3, 2, 1). We measured the percent yellow flowers accepted; 0% acceptance indicates the bee had completely specialized on blue flowers, while 100% acceptance shows the bee had completely generalized and accepted both flower types. Any percentage between indicates that the bee fluctuated between specialization and generalization at certain points in the simulation. Averages for a generalized environment and specialized environment were graphed with 95% CI (Figure 6). These findings show that reduced memory can result in the appearance of partial preferences when the same bee with 'all-knowing' memory would always either specialize or generalize for each trial. Note that as memory size decreases, the virtual bee shows greater partial preferences; that is, it accepts yellow flowers more frequently in a specialized environment, or less frequently in a generalized environment.

(a)



(b)

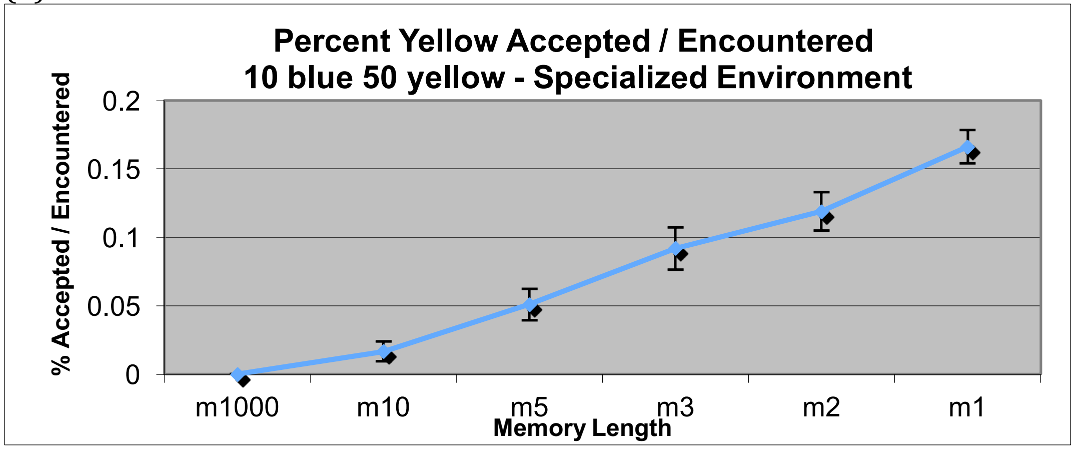


Figure 6: Percentage yellow flowers accepted over encountered by bees with varying memory lengths in generalized (a) and specialized (b) environments. 'Mx' denotes the number of blue-blue transitions (blue search times) that can be stored in memory. Partial preferences are observed when values are between 0 and 1. N=10 for each memory length.

The impact of limited memory and environmental variability on bee decision making

By examining paths, a bee might take during a simulation trial, we can understand why partial preferences are observed. Environmental variability's effect on decision making is illustrated in figure 7 with bee memory length 3. Here, the bee is in a generalizing environment, where a bee with a memory length of 1000 would always accept yellow flowers. For the bee taking the violet path, the average of the past three blue search times is greater than the threshold of 60, so the bee will accept the next yellow flower encountered (Figure 7). For the bee taking the red path, the average of the past three blue search times is less than the threshold of 60, so the bee will reject the next yellow flower encountered (Figure 7).

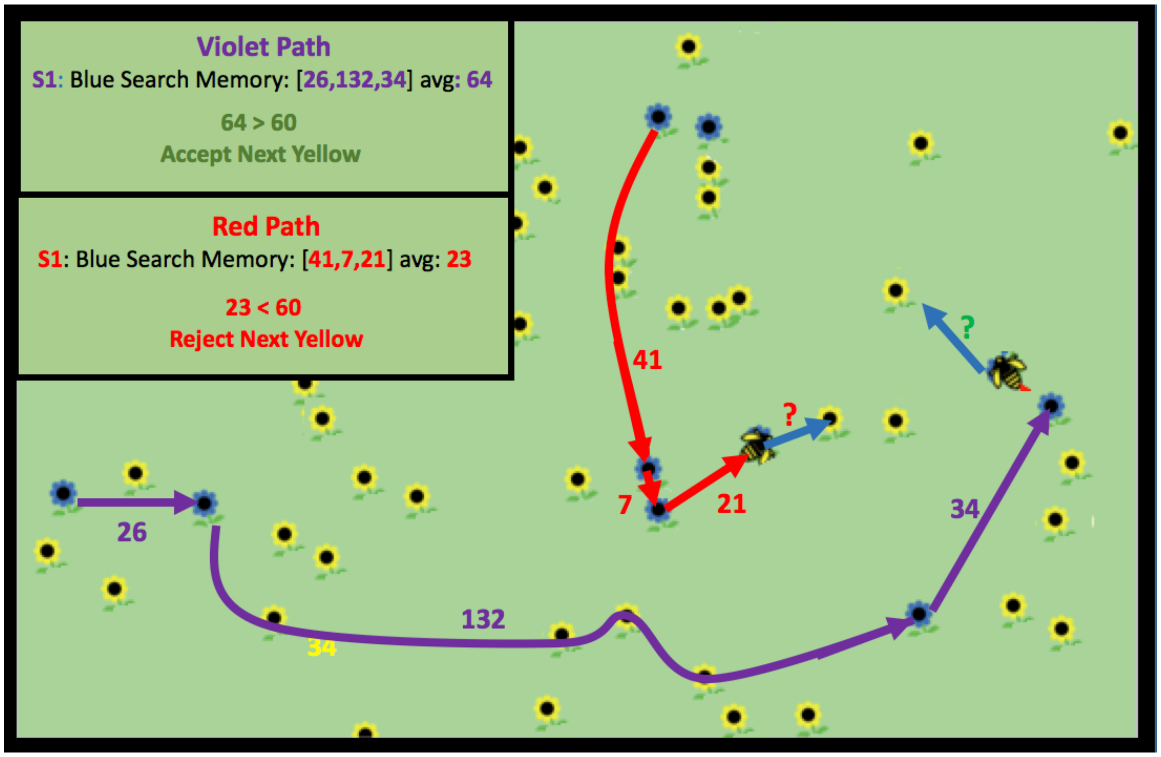


Figure 7: Schematic of how environmental variability can effect optimal decision making processes. For the bee following the red path, the past 3 blue flower encounters have an average search time of 23 ticks, which is below the threshold of 60. This indicates that the more profitable prey type is frequent enough that less profitable prey types should be rejected, so the incoming yellow flower encounter will be rejected. For the bee following the violet path, however, the average search time for the past 3 blue encounters is 64. This is above the threshold of 60, which indicates that the more profitable prey type is uncommon enough that accepting all prey types is optimal, and the incoming yellow flower encounter will be accepted.

Inclusion of handling time within search times

The ODM defines search time and handling times as mutually exclusive. In actuality, it's unclear whether bees are capable of calculating prey handling times separately from search times, especially when considering hundreds of past encounters. By default, SimBee removes handling times of yellow flower encounters that occurred between blue flower encounters, so that the average search time of blue flowers (S1) follows the ODM definition (Handling Time Excluded mode, Figure 8). We also incorporated the ability to include handling times in the calculation of search times, possibly more closely following an actual bee's perception of search times (Handling Time Included mode, Figure 8).

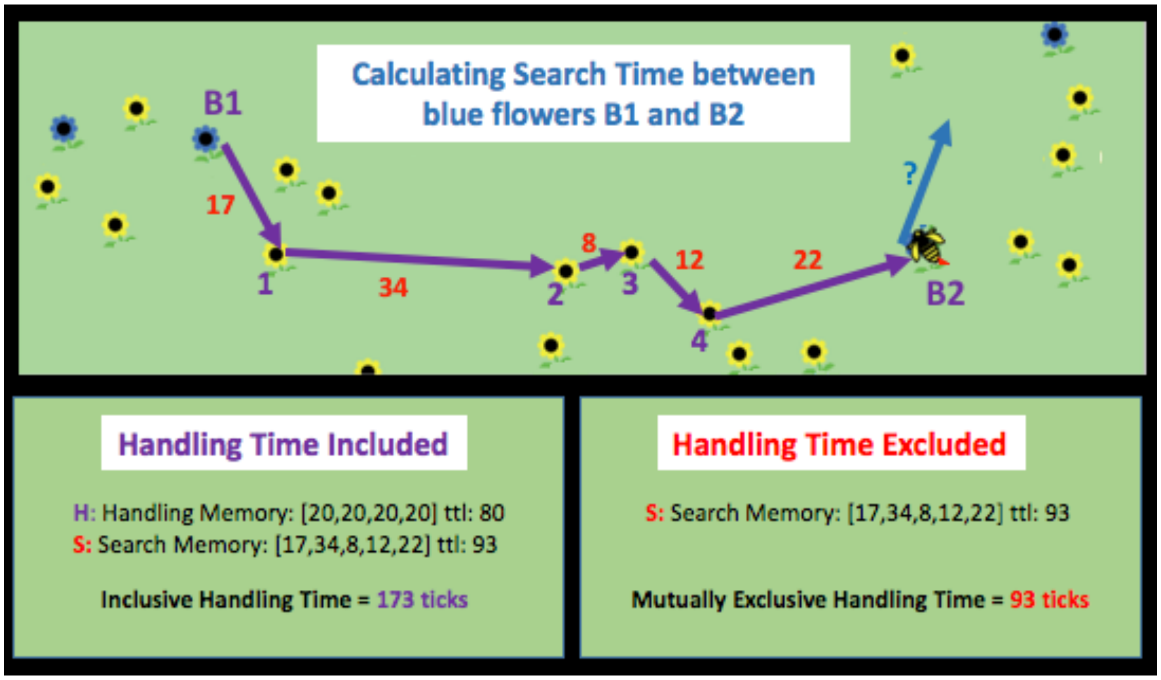

Calculating Search Time between blue flowers B1 and B2
B1
17
1
34
2
8
3
12
4
22
B2
?
Handling Time Included
H: Handling Memory: [20,20,20,20] ttl: 80
S: Search Memory: [17,34,8,12,22] ttl: 93
Inclusive Handling Time = 173 ticks
Handling Time Excluded
S: Search Memory: [17,34,8,12,22] ttl: 93
Mutually Exclusive Handling Time = 93 ticks

Figure 8: Search time and Handling time relationship in the SimBee model. When handling time is excluded from search time, the blue search time equals the sum of 5 search times (purple arrows) for 93 ticks. When handling time is included in search time, the blue search time includes the handling times for the 4 yellow flowers encountered; the total search time is 173 ticks.

Specialized (S) and Generalized (G) environments by Handling Time mode

When handling time is included in search time, the search time is much longer, and the numbers of flowers required to generate Specialized and Generalized environments change. By running 10 trials with a range of blue flower counts and plotting the percentage of generalized trials, we demonstrated that the number of blue flowers required to generate Specialized environments shifts from 10 blue flowers for handling time excluded, to 20 blue flowers for handling time included (Figure 9). For Generalized environments, the number of blue flowers shifts from 20 blue flowers for handling time excluded, to 40 blue flowers for handling time included (Figure 9). This shift was considered significant for the purpose of demonstrating how the relationship between handling and search time can alter how bees perceive generalized or specialized environments, while further trials are needed in order to consider this shift truly statistically significant.

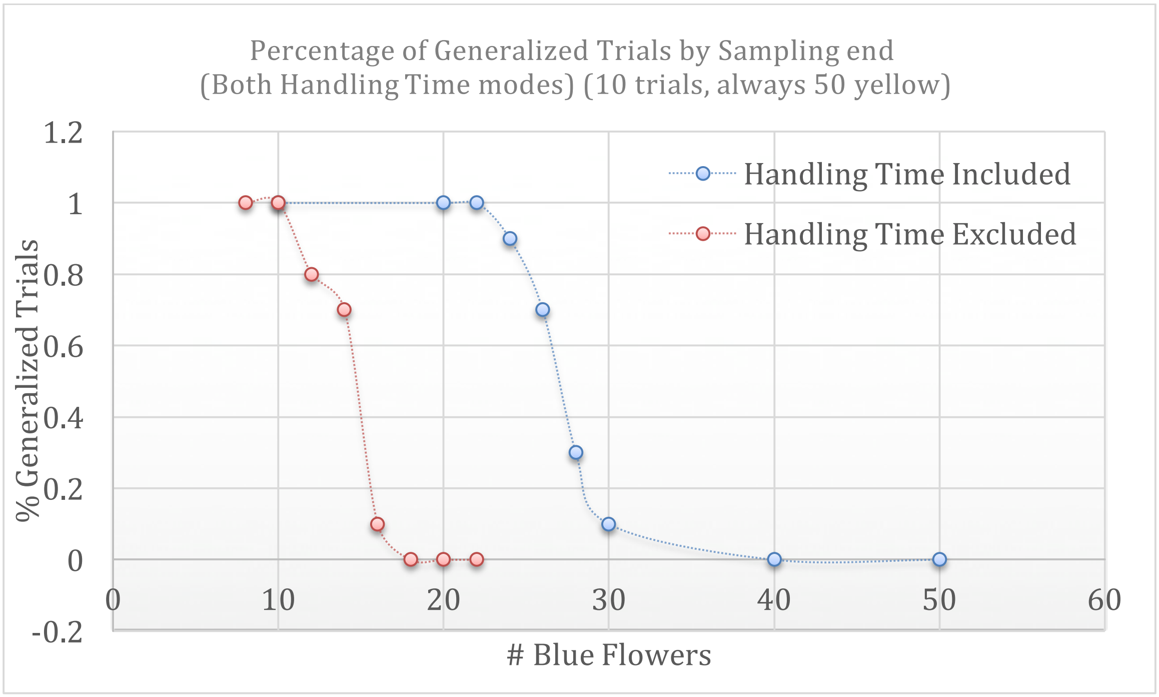


Figure 9: Specialized (S) and Generalized (G) environments by handling time mode. With handling time excluded, G environment is achieved with 10 blue flowers, and S environment is achieved with 20 blue flowers. When handling time is included in search times, however, G environment is shifted to 20 blue flowers, and S environment is shifted to 40 blue flowers.

Validating the SimBee Model in a lab setting

In order to test whether the model corresponds to real bee behavior, we wanted to generate simulation worlds that could be replicated and used in a laboratory environment. Comparison of real and simulated bee behavior would then enable us to validate and refine the simulation, as well as providing a way of estimating actual bee memory length and behavioral strategies. To accomplish this, we developed a switch to facilitate laboratory parameters for the SimBee model. For the floral environment layout, possible flower locations were constricted to a grid: 15 rows by 22 columns totaling 330 possible flower positions. Each row and column were 6.5cm apart, and knowing bees flight speed approximated 15cm/s between encounters, the simulation parameters were structured to emulate this behavior (Figure 10). Blue flowers were adjusted to contain 13ul of nectar and have a 4.5 second handling time. Yellow flowers were adjusted to contain 6.5ul of nectar and have a 4.5 second handling time.

Using the SimBee in Laboratory mode, we generated a laboratory environment corresponding with a simulation world (Figure 10). The laboratory replica replaces color discrimination with odor discrimination, because this forces bees to approach flowers very closely in order to discriminate rewarding and non-rewarding odors, similar to the programmed bee behavior in the model. Laboratory tests will allow us to determine whether we observe specialized and generalized environments, as well as partial preferences, similar to those observed in our simulations. The accuracy of this method of designing model parameters to emulate known lab specifications will remain unclear until assessment metrics are incorporated for comparison against actual bee behavior data.

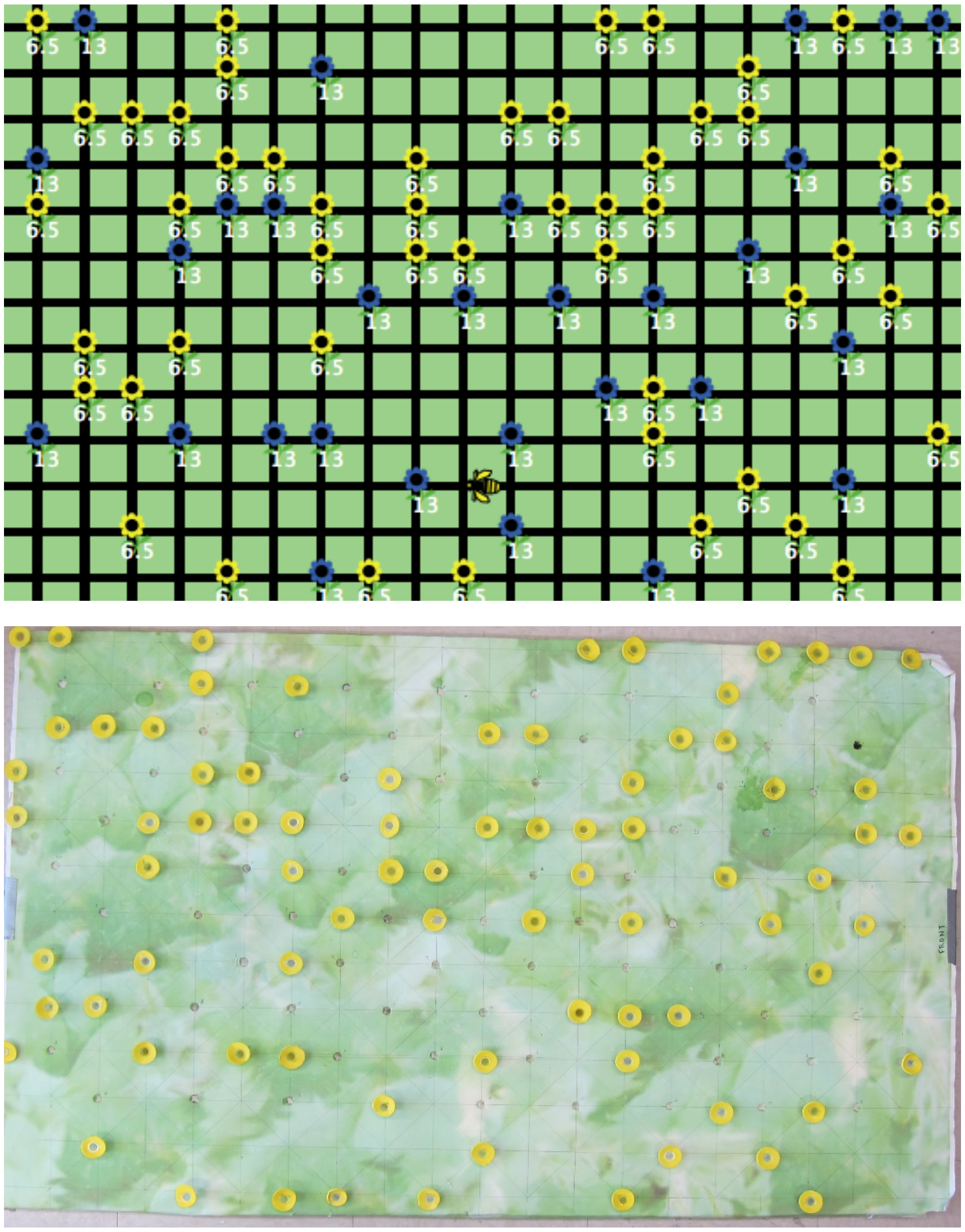


Figure 10: Testing simulation predictions with laboratory data. In Laboratory mode, the simulation generates floral environments on a grid (top panel) that is easily translated to a foam board with slots for positioning artificial flowers – microfuge tubes filled with sugar water 'nectar' surrounded by foam 'petals' (bottom panel). Odorants on the surface of flowers enables bees to differentiate low and high quality flowers. The board fits into a bee cage; bees can be released into the cage and their behavior tracked by video monitoring.

Discussion:

By programming virtual bee foragers with an adaptive decision-making algorithm based on Optimal Diet Model developed by Charnov (1976), modified to include memory, a realistic virtual bumblebee was developed that can be used to emulate actual bee behavior and better assist future studies involving foraging behavior on the population level. By comparing percentage of prey acceptance over encounter of varying memory sizes in the same environments, the results show that reductions in memory cause bees to make non-optimal choices when floral resources are randomly distributed in the environment. Previous diet model designs cannot explain partial preferences due to model constraints, however by incorporating memory ability into the model, partial preferences are a natural outcome from environmental variability. By investigating how handling times are incorporated into search times, the model allows comparison to actual bee behavior to determine how bees actually perceive search times during foraging, and also to estimate actual bee's memory capacity. These results suggest that environmental stressors affecting memory may contribute to population decline by increasing non-optimal foraging choices, which could ultimately reduce the reproductive potential of bumblebee colonies.

The BeeSim model is novel because it is a memory-based extension of the optimal diet model rules in the foraging model. In Becher et al's BEEHAVE model, an agent-based foraging model is derived for honeybee colonies that incorporates colony dynamics and an agent-based foraging model, however it does not incorporate the optimal diet model for agent decision making. Dyer et al's, Bee reverse-learning behavior simulations incorporate agent-based computer simulations that involve parameters of flower handling times and rewards in order to help predict efficiency in bee behavior when adjusting to rewarding flowers changing to unrewarding flowers, but does not address the diet model in their model designs. Qu et al's EcoSimInGrid model is a spatially explicit agent-based simulator for complex analyses of pollinators on plant populations, however their design does not incorporate behavioral choices based on the optimal diet model.

Zhang et al's recent experience-driven (RED) decision-making foraging model explains partial preferences by using a probability-based model that determines prey acceptance by comparing the profitability of the current encounter to the average of the most recent flower encounters. Instead of incorporating search times for various types into decision-making as in the optimal diet model, Zhang et al. include an aversion factor related to the hunger of the forager over time: foragers that haven't eaten recently are more likely to accept encountered prey. While an interesting model, this simulation is less likely to be relevant to bumblebee foragers, since their behavior is not driven by individual hunger [Goulson, 2009].

Future work using the SimBee model will involve more realistic aspects of the simulation, including changing floral environments, alternate memory weights, and additional prey types that offer a range of available rewards instead of a static value. Adjusting the decision-making algorithm to include multiple prey types that place more weight on the most recent data gathered will create a more realistic system, allowing bees to adapt to constantly changing resources available.

Predicting the effect of memory impairment on the overall bee population is the final goal of the SimBee model. The model will complement and enhance information gained from laboratory and field studies.

The SimBee model can also be used in an educational tool that allows students to learn about biological concepts through agent-based modelling. The simulation separates aspects of foraging into single variables, which gives students the ability to hypothesis test by adjusting single parameters of the model with separate sliders for each applicable model aspect. The model by design is user friendly and interactive with the real-time ability to adjust environmental values, which can facilitate public education and a greater understanding of the importance of bumblebee decline.

References:

Arundel, J., Oldroyd, B.P., Winter, S. (2013) Modelling estimates of honey bee (Apis spp.) colony density from drones. Ecological Modelling 267: 1-10

Becher, M.A., Grimm, V., Thorbek, P., Horn, J., Kennedy, P.J., Osborne, J.L. (2014) BEEHAVE: a systems model of honeybee colony dynamics and foraging to explore multifactorial causes of colony failure. Journal of Applied Ecology 51: 470–482.

Bukovac, Z., Dorin, A., Dyer, A.G. (2013) A-Bees See: A Simulation to Assess Social Bee Visual Attention During Complex Search Tasks. ECAL 2013: 276-283.

Castello, E., Yamamoto, T., Nakamura, T., Ishiguro, H. (2013) Task Allocation for a robotic swarm based on an Adaptive Response Threshold Model. Control, Automation and Systems (ICCAS) 12: 259-266

Charnov, E. (1976). Optimal Foraging: Attack Strategy of a Mantid. The American Naturalist 110: 141-151.

Couvillon, M.J., Toufailia, H.A., Butterfield, T.M., Shrell, F., Ratnieks, F.L.W., Shurch, R. (2015) Caffeinated Forage Tricks Honeybees into Increasing Foraging and Recruitment Behaviors. Current Biology 25: 2815-2818

Das, S., Biswas, S., Kundu, S. (2013) Synergizing fitness learning with proximity-based food source selection in artificial bee colony algorithm for numerical optimization. Applied Soft Computing 13: 4676-4694

Dyer, A.G., Chittka, L. (2004) Biological significance of distinguishing between similar colours in spectrally variable illumination: bumblebees (Bombus terrestris) as a case study. Journal of Comparative Physiology 190: 105-114

Dyer, A.G., Dorin, A., Reinhardt, V., Garcia, J.E., Rosa, M.G.P. (2014) Bee reverse-learning behavior and intra-colony differences: Simulations based on behavioral experiments reveal benefits of diversity. Ecological Modelling 277: 119-131.

Faruq, S., McOwan, P.W., Chittka, L. (2013) The biological significance of color constancy: An agent-based model with bees foraging from flowers under varied illumination. J. Vis 10: 1-4.

González-Varo, J.P., Biesmeijer, J.C., Bommarco, R., Potts, S.G., Schweiger, O., Smith, H.G., Steffan-Dewenter, I., Szentgyörgyi, H., Woyciechowski, M., Vilà, M. (2013) Combined Effects of global change pressures on animal-mediated pollination. Trends in Ecology & Evolution 28: 221-240

Goulson, D. Bumblebees: Behaviour, Ecology, and Conservation (2nd Edition). 2009. Oxford University Press.

Grimm, V., Berger, U., DeAngelis, D.L., Polhill, J.G., Giske, J., Railsback, S.F. (2010) The ODD protocol: A review and first update. Ecological Modelling 221: 2760–2768

Helbing, D. (2012) Social Self-Organization Agent-Based Simulations and Experiments to Study. Emerging Social Behavior 11: 341-349.

Holt, R.D. (2013). Unstable predator–prey dynamics permits the coexistence of generalist and specialist predators, and the maintenance of partial preferences. Israel Journal of Ecology and Evolution 59: 5-9

Janssen, M.A., Hill, K. (2014) Benefits of Grouping and Cooperative Hunting Among Ache Hunter–Gatherers: Insights from an Agent-Based Foraging Model. Human Ecology 42: 823-835

Liu, B., Cai, M., Yu, J. (2015) Swarm Intelligence and its Application in Abnormal Data Detection. Informatics 39: 1-13

Lourenco, R., Delgado, M., Korpimaki, E., Penteriani, V. (2015) Evaluating the influence of diet-related variables on breeding performance and home range behavior of a top predator. Population Ecology 57: 625-636

Mills, R., Correia, L. (2015) The Influence of Topology in Coordinating Collective Decision-Making in Bio-hybrid Societies. Progress in Artificial Intelligence 9273: 250-261

Muth, F. (2015) The effects of acute stress on learning and memory in bumblebees. Learning and motivation 50: 39-43

Politopoulos, I. (2007) Review and Analysis of Agent-based Models in Biology. University of Liverpool Tech Reports 07: 2-14

Pyke, G.H. (1984) Optimal Foraging Theory: A Critical Review. Ann. Rev. Ecol. Syst. 15: 523-75.

Qu, H., Seifan, T., Tielborger, K., Seifan, M. (2013) A spatially explicit agent-based simulation platform for investigating effects of shared pollination service on ecological communities. Simulation Modelling Practice and Theory 37: 107–124

Sih, A. (2013). Understanding variation in behavioral responses to human-induced rapid environmental change: a conceptual overview. Animal Behavior 85: 1077-1088.

Sih, A., Christensen, B. (2001) Optimal diet theory: when does it work, and when and why does it fail? Animal Behaviour 61: 379–390.

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

Zhang, F., Cang, H. (2014) Recent experience-driven behaviour optimizes foraging. Animal Behaviour 88: 13-19.

Appendix 1: Code Structure:

Bees

- Move-bees:
    - If not on-flower? [bees-search-for-flower]
    - If on-flower? [bees-collect-nectar]
- Bees-search-for-flower
    - if chosen-flower = NOBODY [bees-decide-next-flower]
    - If no flowers-in-view [move randomly]
    - If chosen-flower != NOBODY [face chosen-flower, move towards]
- Bees-decide-next-flower
    - IF any flowers-in-view [choose random flower]
    - Determine which prey type should be accepted
    - If color of chosen-flower = next-color [accept encounter]
- Bees-collect-nectar
    - Bees-determine-memory-values
    - Update-memory
    - bee-decide-prey
    - If time spent on flower reached, forage and leave
- Bees-determine-memory-values
    - If mutually-exclusive handling time:
        - Subtract handling-time of between encounters recorded
    - Store search-time in search memory
    - If accepting-encounter? true
        - Store energy and handling times in memory
- Update-memory
    - If memory list length > bee-memory-length
        - remove oldest memory value
    - If sampling and all memory lists filled
        - Set sampling false
        - Reset nectar held, encounter tracking variables
- Bee-decide-prey
    - Calculate averages for memory lists
        - Search times
        - Handling times
        - Energy contents
    - Determine prey1 = greatest energy / handling-time
    - Determine threshold
        - ((p1energy * p2handling) / p2energy) – p1handling
    - If average p1search-time > threshold, generalize
        - Accept any prey type encountered next
    - If average p1search-time < threshold, specialize
        - Accept only prey1 for next encounter

Flowers

- Regenerate-flower-nectar

    - If nectar content < flower-max-nectar [refill nectar]
- Flowers-track-occupation
    - If any bees-here [set occupied? true]
    - Else [set occupied? false]

Appendix 2: Full SimBee Code

```
;-------------------------------------------------breeds---------------------------------------------------------------------------
breed [flowers flower]
breed [bees bee]
breed [hives hive]

;-------------------------------------------------variables--------------------------------------------------------------------------
globals[
  ; -

  ticks-at-sampling-end              ; ----- output variable: time when sampling ended
  specialized-at-sampling-end?    ; ----- output variable: if bee specialized at sampling end

  world-name                  ; ----- output variable: what the world output file is named

  hive-1                    ; ---- ID of 1st hive
  hive-2                 ; ---- ID of 2nd hive

  switch-count                          ; ----- output variable: # times flower color != last visit
  total-count                  ; ----- output variable: # flower visits total
  error-visit-count                  ; ----- debug variable: if bee visits unintended flower
  travel-distances                  ; ----- output variable: distances between visited flowers
  threshold-switch-count             ; ----- output variable: # times S1 went above/below threshold

  NectarGathered                  ; ----- output variable: total nectar bee gathered

  current-iteration                  ; ----- behaviorspace variable: tracks current run

  next-run?                  ; ----- behaviorspace variable: if true, behaviorspace deletes current world file, starts next run
  done-time?                  ; ----- behaviorspace variable: also used to end start next run in some cases

     blue-search-mean                  ; ----- OFT variable: average blue search time of bee memory
   yellow-search-mean                  ; ----- OFT variable: average yellow search time of bee memory
    yellow-handling-mean                  ; ----- OFT variable: avg yellow handling time
    blue-handling-mean                  ; ----- OFT variable: avg blue handling time
    yellow-energy-mean                  ; ----- OFT variable: avg yellow energy content
    blue-energy-mean                  ; ----- OFT variable: avg blue energy content
    prey-1                  ; ----- OFT variable: defines which prey type is most rewarding
    next-prey                  ; ----- OFT variable: defines if next encounter will be specialized or generalized
    search-threshold              ; ----- OFT variable: calculates what threshold is:  (E1*h2) / E2) - h1

    last-visit-yellow                  ; ----- graph variable: updates sequence graph with yellow bar
    last-visit-blue                  ; ----- graph variable: updates sequence graph with blue bar
    same-color-inarow                  ; ----- graph variable: tracks sets of same-color visit patterns
    PropNecBlue                  ; ----- graph variable: displays proportion of nectar has come from blue flowers
    PropNecYellow                  ; ----- graph variable: displays proportion of nectar has come from yellow flowers
  blue-population                  ; ----- graph variable: tracks # blue flowers in environment
  yellow-population                  ; ----- graph variable: tracks # yellow flowers in environment
  last-flower1                  ; ----- debug variable: displays last flower visited
  this-flower1                  ; ----- debug variable: displays current flower visiting

   blue-accepted                  ; ----- graph variable: displays # blue flowers accepted
  blue-encountered                  ; ----- graph variable: displays # blue flowers encountered
  yellow-accepted                  ; ----- graph variable: displays # yellow flowers accepted
  yellow-encountered                  ; ----- graph variable: displays # yellow flowers encountered
  blue-ratio1                  ; ----- graph variable: displays % blue flowers accepted / encountered
  yellow-ratio1                  ; ----- graph variable: displays % yellow flowers accepted / encountered
  seq-update?                  ; ----- graph variable: if true, updates visit sequence graph

  sampling-file-name                  ; ----- output variable: name of separate export file for sampling data if needed
  specialized?                  ; ----- output variable: indicates of bee specialized or not at sampling end
  landscapeID                  ; ----- output variable: unique # for each generated environment, for multiple runs with same layout


  ; -


]

bees-own [
  on-flower?                  ; ----- indicates if flower present
  hive-belongs-to                  ; ----- which hive it belongs to
  bee-nectar-collected                  ; ----- amount of nectar collected
  time-on-flower                  ; ----- how long bee has been sitting on flower
  flowers-in-view                  ; ----- agentset of all flowers detected in view
  last-flower                  ; ----- ID of last flower visited
  chosen-flower                  ; ----- ID of next flower to visit
  next-color                  ; ----- color of next flower
  yellow-handling-memory                  ; ----- list of past yellow flower handling times
  blue-handling-memory                  ; ----- list of past blue handling times
  yellow-energy-memory                  ; ----- list of yellow energy contents
  blue-energy-memory                  ; ----- list of blue energy contents
  yellow-search-memory                  ; ----- list of yellow search times
  blue-search-memory                  ; ----- list of blue search times
  last-blue-time                  ; ----- timestamp of last blue encounter
  last-yellow-time                  ; ----- timestamp of last yellow encounter
  flower-landing-memory                  ; ----- list of timestamps when past flowers visited
  just-landed?                  ; ----- indicates if bee just arrived to flower or not
  sampling?                  ; ----- indicates if bee is still sampling
  prey1                  ; ----- indicates the more rewarding prey type
  prey2                  ; ----- indicates the less rewarding prey type
  nextprey                  ; ----- equals "both" if bee is generalized, "blue" if specialized
  last-color                  ; ----- color of last flower visited
  accept-encounter?                  ; ----- indicates if bee will accept the current encounter

]

flowers-own [
  flower-nectar-content                  ; ----- amount of nectar in flower
  flower-handling-time                  ; ----- amount of handling time for flower
  occupied?                  ; ----- indicates if a bee is on the flower
  flower-type                  ; ----- indicates prey type
  block-number                  ; ----- indicates which region flower is located in

]

hives-own [
  hive-nectar-content                  ; ----- amount of nectar that's been returned to the hive
]

to generate-worlds  ; ----------------------------- create worlds for behaviorspace
  let original-number number-of-worlds                  ; ----- to offset world file names
  let original-handling yellow-handling-time                  ; ----- saves original yellow handling time, to reset once worlds generated

  set number-of-worlds number-of-worlds + last-world-offset                  ; ----- offsets world file names
  while [number-of-worlds > last-world-offset = true] [

    setup
    set bee-decision-type "prey-model"
    set bee-memory-length 1000                                             ; ----- export world for 'all-knowing' bee
    set number-of-worlds number-of-worlds - 1
    set world-name (word "g" number-of-worlds  ".csv")
    export-world world-name
    set bee-memory-length 10                                             ; ----- adjust world name and bee memory length, export world
    set number-of-worlds number-of-worlds - 1
    set world-name (word "g" number-of-worlds  ".csv")
    export-world world-name
    set bee-memory-length 5                                             ; ----- *
    set number-of-worlds number-of-worlds - 1
    set world-name (word "g" number-of-worlds  ".csv")
    export-world world-name
    set bee-memory-length 3                                             ; ----- *
    set number-of-worlds number-of-worlds - 1
    set world-name (word "g" number-of-worlds  ".csv")
    export-world world-name
    set bee-memory-length 2                                             ; ----- *
    set number-of-worlds number-of-worlds - 1
    set world-name (word "g" number-of-worlds  ".csv")
    export-world world-name
    set bee-memory-length 1                                             ; ----- *
    set number-of-worlds number-of-worlds - 1
    set world-name (word "g" number-of-worlds  ".csv")
    export-world world-name

  ]
  set last-world-offset last-world-offset + original-number                  ; ----- ensures world files aren't overwritten
  set number-of-worlds original-number                                ; ----- returns world count
  set yellow-handling-time original-handling                                ; ----- resets yellow handling time

end


to export-lab-world                                ; ---------------- export specific environment for lab recreation
  export-world export-world-name
  export-view (word  "view-" export-world-name ".png")                                ; ----- exports image of landscape generated
end


to determine-world                                ; -----; ---------------- for behaviorspace: determine next world to import and run
  let num-test 1000                                ; ----- starts very high to enure last world generated is first
  let world-test (word "g" num-test  ".csv")
  while [file-exists? world-test = false][                                ; ----- continues searching until filename exists
    set num-test num-test - 1
    set world-test (word "g" num-test  ".csv")
    if num-test < 0 [error "World doesn't exist"]                                ; ----- indicates no world left to import
  ]
  set world-name world-test                     ; ----- sets world name to import once valid world name found

end

to setup-imported                       ; -----; ------------- for behaviorspace: import next world and setup
  clear-all
  reset-ticks
  setup-globals
  determine-world
  import-world world-name                       ; ----- imports world name determined from determine-world

end







;-----------------------------------------------setting values ------------------------------------------------------------------------


to fix-flower-count                       ; -----; ------------- for lab environment: make sure total flower count is correct
  set yellow-flower-count total-flower-count - blue-flower-count
end




;---------------------------------------------------setup--------------------------------------------------------------------------------
                       ; -----;initializes breeds and environment
to setup
  clear-all
  reset-ticks

  setup-globals

  set travel-distances [1]                  ; ----- for monitoring distances travelled
                        ; -----;; call setup functions

  setup-patches
  setup-flowers
  space-flowers
  setup-bees
 ; setup-hives                        ; ----- hives not currently used
  space-flowers
  set-labels

  setup-export


end

to setup-globals                        ; ----- adds initial values to variables that can't be empty at initialization
    set blue-search-mean 1
    set yellow-search-mean 1
    set yellow-handling-mean 1
    set blue-handling-mean 1
    set yellow-energy-mean 1
    set blue-energy-mean 1
    set search-threshold 1
    set threshold-switch-count 0
    set done-time? false
    set yellow-encountered 1
    set yellow-accepted 1
    set blue-encountered 1
    set blue-accepted 1
    set blue-ratio1 1
    set yellow-ratio1 1
    set seq-update? false
    set landscapeID random-float 1
end




;----setup-patches----
;makes the background green
to setup-patches
   ask patches [ set pcolor 57 ]
end
```

```
;----setup-flowers----
;initializes color, shape, size, and location of flowers

to space-flowers                                ; -----;; ------------- adjust flowers into lab-environment spacing grid
  ask flowers [

    ifelse lab-flower-layout? = true [
      while [pxcor mod 5 != 3 or pycor mod 5 != 0 or any? other flowers-here][setxy random-pxcor random-pycor]       ; ----- if lab layout wanted, ensures
flower placed in available grid spots
      ]
    [
      while [any? other flowers in-radius 2 = true or any? hives in-radius 2 = true]                               ; ----- if random layout wanted, ensures flowers don't
overlap
     [setxy random-pxcor random-pycor]]
  ]
end

to setup-flowers                                ; ----- creates flower agents
  set-default-shape flowers "flower"
  create-flowers yellow-flower-count [                                ; ----- yellow flowers
    set flower-type "f1"
    setxy random-pxcor random-pycor
    set color yellow
    set size 5
    set occupied? true
    set flower-nectar-content yellow-energy
    set flower-handling-time yellow-handling-time
  ]
  set yellow-population yellow-flower-count

    create-flowers blue-flower-count [                                ; ----- blue flowers
    set flower-type "f2"
    setxy random-pxcor random-pycor
   ; if any? flowers-here [setxy random-pxcor random-pycor]
    set color blue
    set size 5
    set occupied? true
    set flower-nectar-content blue-energy
    set flower-handling-time blue-handling-time
  ]
   set blue-population blue-flower-count

end





;----setup-bees----
;initializes color, shape, size, location, and variables of bees
to setup-bees

    create-bees number-of-bees
    [
      set shape "bee 2"
      set color 45
      set size 4
      setxy 0 -20
      set on-flower? false
      set time-on-flower 0                                          ; ----- indicates no time has been spent on flower to start
      set hive-belongs-to 1
      set bee-nectar-collected 0
      set just-landed? false
      set accept-encounter? true
      set flowers-in-view flowers with [self != [last-flower] of myself] in-cone view-radius view-degrees       ; ----- adds flower for initialiation
      set last-flower one-of flowers                                          ; ----- chooses random flower as last-flower so variable isn't empty at start
      set chosen-flower one-of flowers with [self != [last-flower] of myself]                            ; ----- * chosen-flower
      set next-color [color] of chosen-flower
      set yellow-search-memory [10]                                          ; ----- added default number to lists so variable isn't empty at start
      set blue-search-memory [10]                                          ; ----- *
      set yellow-handling-memory [20]                                          ; ----- *
      set blue-handling-memory [20]                                          ; ----- *
      set yellow-energy-memory [10]                                          ; ----- *
      set blue-energy-memory [40]                                          ; ----- *
      set flower-landing-memory [1]                                          ; ----- *
      set last-blue-time 0                                          ; ----- *
      set last-yellow-time 0                                          ; ----- *
      set sampling? true                                          ; ----- indicates bee starts in sampling mode
      set prey1 blue                                          ; ----- indicates bee considers blue as prey1 to start
      set prey2 yellow                                          ; ----- indicates bee considers yellow as prey2 to start
      set nextprey "both"                                          ; ----- indicates bee can visit both prey types to start
      set PropNecBlue .5                                          ; ----- 50% nectar from blue to start
      set PropNecYellow .5                                         ; ----- 50% nectar from yellow to start
      set blue-accepted 0
      set blue-encountered 0
      set yellow-accepted 0
      set yellow-encountered 0
      set yellow-ratio1 1                                          ; ----- indicates 100% yellow encounters accepted to start
      set blue-ratio1 1                                            ; ----- indicates 100% blue encounters accepted to start
    ]
end

;----setup-hives----
;initializes color, shape, size, location ... of hives
to setup-hives                                   ; ----- not in current use
  if number-of-hives = 1
  [
    ;initialize all values for a single hive
    create-hives 1
    [
      set shape "beehive"
      set color 45
      set size 10
      setxy -50 0
      set heading 0
      set hive-1 self
      set hive-nectar-content 0
    ]
  ]

  if number-of-hives = 2
  [
    ;initialize all values for hive 1
    create-hives 1
    [
      set shape "beehive"
      set color 45
      set size 10
      setxy 50 50
      set heading 0
      set hive-1 self
      set hive-nectar-content 0
    ]
    ;initialize all values for hive 2
    create-hives 1
    [
      set shape "beehive"
      set color 45
      set size 10
      setxy -50 -50
      set heading 0
      set hive-2 self
      set hive-nectar-content 0
    ]
  ]
end



;-------------------------------------------------------go-------------------------------------------------------------------------------------
;sets the simulation in motion
to go
  if show-radius?                                  ; ----- debug: redraw range of detect-flowers each time bee moves
  [clear-patches draw-view]

  regenerate-flower-nectar
  move-bees
  flowers-track-occupation
  set-labels
  tick
  if export? [export-data]                         ; ----- for data output: export data if applicable requirements met
end


to draw-view                                       ; ----- ; debug ; draws the bee's view range
      ask bees [
        ask patches in-cone view-radius 360 with [distance myself > (view-radius - 1)][set pcolor green]
        ask patches in-cone view-radius view-degrees with [distance myself > (1)][set pcolor white]]
end



to move-bees                           ; -----    ; determines when bees execute certain behaviors
  ask bees
  [
    if on-flower? = false [ bees-search-for-flower ]                    ; ----- ; while the bee is not on a flower collecting nectar
    if on-flower? [ bees-collect-nectar ]                                          ; ----- ;while the bee is on a flower
   ]
end


to bees-search-for-flower                                                ; -----                 ; directs bee searching behavior while not on a flower
  set flowers-in-view flowers with [self != [last-flower] of myself] in-cone view-radius view-degrees                 ; ----- updates detected flower agentset
     if chosen-flower = NOBODY [ bees-decide-next-flower ]                                                                    ; ----- ; ----- end of "if chosen-flower = NOBODY"
section


     if any? flowers-in-view = false or chosen-flower = NOBODY                        ; -----if the bee sees no flowers it moves forward randomly
     [
      forward 1
      right random 30
      left random 30
     ]

     if any? flowers-in-view and chosen-flower != NOBODY  ;when the bee sees the chosen flower it heads towards it
     [
      set heading towards chosen-flower
      if on-flower? = false [ forward 1 ]
     ]

     if any? flowers-here                                   ; ----- when the bee reaches the chosen-flower it sets the variable on-flower? to true
     [
      ifelse [occupied?] of one-of flowers-here = true            ; ----- if flower already occupied, move on
      [forward 1]
      [
       ifelse one-of flowers-here = chosen-flower [              ; ----- if bee reached chosen-flower, start collect procedure, else continue towards chosen-flower
       set on-flower? true
       forward 0]
       [forward 1]
      ]
     ]
end

to bees-decide-next-flower                                             ; ----- bee decides whether to accept the next encounter
  set last-color [color] of last-flower

      ifelse random-flower? [                                                       ; ----- if bee should choose a random flower in detected flower agentset

        if any? flowers-in-view [
       set chosen-flower one-of flowers-in-view

       if bee-decision-type = "prey-model" [                                      ; ----- prey-model: if chosen flower is correct prey type, bee will accept encounter

       ifelse [color] of chosen-flower = next-color
         [set accept-encounter? true]
         [set accept-encounter? false]
       if nextprey = "both" [set accept-encounter? true]                                ; ----- if bee is generalized / sampling, ensures chosen flower will be accepted

       ]

       if bee-decision-type = "PropNec" [              ; ----- PropNec: if chosen-flower color has higher %nectar, ensure acceptance. If lower %nectar, P(visit) =
f1/f2 % chance
        ifelse PropNecBlue >= PropNecYellow [
        ifelse [color] of chosen-flower = blue
        [set accept-encounter? true]
        [
         ifelse random-float 1 < (PropNecYellow / PropNecBlue)
         [set accept-encounter? true]
         [set accept-encounter? false]
        ]
        ]
        [
        ifelse [color] of chosen-flower = yellow
        [set accept-encounter? true]
        [
         ifelse random-float 1 < (PropNecBlue / PropNecYellow)
         [set accept-encounter? true]
         [set accept-encounter? false]
        ]
        ]
        ]
       if sampling? [set accept-encounter? true]                              ; ----- if bee in sampling mode, ensures acceptance
       ]
      ]
      [
       set chosen-flower min-one-of flowers-in-view with                                ; ----- random-flower? is false: bee selects closest flower in detected
agentset
       [color = [next-color] of myself and self != [last-flower] of myself][distance myself]
      ]
```

```
end



to bees-collect-nectar                                  ; -----  directs bee collection behavior while on a flower
 let this-flower one-of flowers-here


  bees-determine-memory-values                          ; ----- calculate search times for memory storage
  update-memory                                         ; ----- adjust memory lists if applicable
  set NectarGathered bee-nectar-collected               ; ----- update monitor with new nectar amount held
  bee-decide-prey                                       ; ----- bee updates prey decisions
  set seq-update? true                                  ; ----- graph: update sequence graph


  set just-landed? false                                ; ----- ends one-time-only portion of calculations when bee is landed

  if not accept-encounter? [set time-on-flower [flower-handling-time] of this-flower + 1]     ; ----- if bee accepted encounter, it must wait at this location for
(handling-time) ticks
  set time-on-flower (time-on-flower + 1)
  if this-flower != chosen-flower [set error-visit-count error-visit-count + 1]           ; ----- debug: if this flower wasn't the chosen-flower, track error

  if last-flower != nobody [ifelse [color] of last-flower != [color] of this-flower
  [
    if time-on-flower >= [flower-handling-time] of this-flower + SwitchCostPenalty         ; ----- switch cost: if last visited was different color, increased bee
pause time required
    [
      ask this-flower [set flower-nectar-content 0]                                      ; ----- ask flower to deplete nectar held
        let this-distance distance last-flower
  set travel-distances lput this-distance travel-distances                                     ; ----- update list of distances travelled
      set last-flower this-flower
      set last-flower1 last-flower
      set on-flower? false
      set chosen-flower NOBODY
      bees-choose-color
      set switch-count switch-count + 1
      set total-count total-count + 1
      set time-on-flower 0
      forward 2
    ]
  ]
  [
    if time-on-flower >= [flower-handling-time] of this-flower                        ; ----- switch cost: if last visited was same color, less pause time was required
    [
      ask this-flower [set flower-nectar-content 0]
        let this-distance distance last-flower
  set travel-distances lput this-distance travel-distances
      set last-flower this-flower
      set on-flower? false
      set chosen-flower NOBODY
      bees-choose-color
      set total-count total-count + 1
      set time-on-flower 0
      forward 2
    ]
  ]
  ]
end

to bees-determine-memory-values
   let this-flower one-of flowers-here
  set this-flower1 this-flower

  set just-landed? true                                 ; ----- ensures certain tasks only occur once, at landing (adding to memory lists, etc)

  if time-on-flower = 0 and last-flower != nobody[
  ifelse [color] of this-flower = [color] of last-flower                         ; ----- adjusts color streak monitor variables
  [set same-color-inarow same-color-inarow + 1][set same-color-inarow 0]

  if [color] of this-flower = yellow [
    let last-search (ticks - last-yellow-time)                                    ; ----- calculates time since last yellow flower visit

    if handling-time-mode = "mutually-exclusive" [        ; ----- if mutually-exclusive prey-model: subtract handling times of visited flowers of other prey type
from search time
      let yellow-temp flower-landing-memory
    while [length yellow-temp > 1 = true] [
     ifelse first yellow-temp < last-yellow-time [set yellow-temp but-first yellow-temp]                ; ----- if timestamp earlier than last-yellow, remove from list
 [
       set last-search last-search - blue-handling-time                              ; ----- if timestamp between last-yellow and current, subtract from search time, remove
from list
       set yellow-temp but-first yellow-temp]
    ]
    ]
    set yellow-search-memory lput last-search yellow-search-memory                    ; ----- add search time to yellow search memory
    set last-yellow-time (ticks + yellow-handling-time)                            ; ----- set new last-yellow-time, adjust for handling time
    set yellow-encountered yellow-encountered + 1
    set yellow-ratio1 yellow-accepted / yellow-encountered
    if accept-encounter? = true [                                   ; ----- if accepting encounter: also add current flower's handling time / energy to memory lists
    set flower-landing-memory lput ticks flower-landing-memory
    set yellow-energy-memory lput [flower-nectar-content] of this-flower yellow-energy-memory
    set yellow-handling-memory lput [flower-handling-time] of this-flower yellow-handling-memory
    set yellow-accepted yellow-accepted + 1
    set yellow-ratio1 yellow-accepted / yellow-encountered                        ; ----- update accepted/encountered ratio after memory update
    set bee-nectar-collected (bee-nectar-collected + [flower-nectar-content] of this-flower)    ; ----- add flower's nectar content to nectar-held
    ]
  ]
  if [color] of this-flower = blue [                                  ; ----- repeats time since last flower visit calculations, but for blue flowers
    let last-search (ticks - last-blue-time)
   if handling-time-mode = "mutually-exclusive" [         ; ----- if mutually-exclusive prey-model: subtract handling times of visited flowers of other prey type
from search time
    let blue-temp flower-landing-memory
    while [length blue-temp > 1 = true] [
      ifelse first blue-temp < last-blue-time [set blue-temp but-first blue-temp]               ; ----- if timestamp earlier than last-blue, remove from list
 [
       set last-search last-search - yellow-handling-time                          ; ----- if timestamp between last-blue and current, subtract from search time, remove
from list
       set blue-temp but-first blue-temp]
    ]
   ]
    set blue-search-memory lput last-search blue-search-memory
    set last-blue-time (ticks + blue-handling-time)
    set blue-encountered blue-encountered + 1
    set blue-ratio1 blue-accepted / blue-encountered
    if accept-encounter? = true [                                   ; ----- if accepting encounter: also add current flower's handling time / energy to memory lists
    set flower-landing-memory lput ticks flower-landing-memory
    set blue-energy-memory lput [flower-nectar-content] of this-flower blue-energy-memory
    set blue-handling-memory lput [flower-handling-time] of this-flower blue-handling-memory
    set blue-accepted blue-accepted + 1
    set blue-ratio1 blue-accepted / blue-encountered
    set bee-nectar-collected (bee-nectar-collected + [flower-nectar-content] of this-flower)    ; ----- add flower's nectar content to nectar-held
    ]
    if accept-encounter? = false [                                   ; ----- debug: blue flowers should always be accepted, if not then error is logged
      ask this-flower [set color red set size 10]
      export-interface (word  "blue-not-accepted " bee-memory-length " - " bee-decision-type " - "  random-float 1.0 ".png")
      ask this-flower [set color blue set size 5]
  ]
  ]                                                 ; ----- end of search time adjustment / memory list additions
  ]

end

to bees-choose-color

  if nextprey = "yellow" [
    ifelse random 100 < resample-threshold [set next-color yellow][set next-color blue]
  ]
  if nextprey = "blue" [
    ifelse random 100 < resample-threshold [set next-color blue][set next-color yellow]
  ]
  if nextprey = "both" [
    ifelse random 100 < 50 [set next-color blue][set next-color yellow]
  ]


end







to bees-deliver-nectar-to-hive                                  ; ----- if nectar-held reaches bee-nectar-max, return to hive and empty nectar-held into hive
(inactive)
  if any? hives-here
  [ask one-of hives-here [
      set hive-nectar-content (hive-nectar-content + [bee-nectar-collected] of myself)
      set NectarGathered NectarGathered + [bee-nectar-collected] of myself]
  set bee-nectar-collected 0
  ]
  if hive-belongs-to = 1
  [set heading towards hive-1 if any? hives-here = false [forward 1]]
  if hive-belongs-to = 2
  [set heading towards hive-2 if any? hives-here = false [forward 1]]
end

to regenerate-flower-nectar                                    ; ----- if flower unoccupied and nectar less than max, refill nectar
  ask flowers
  [
  if color = blue [
  if flower-nectar-content < blue-energy [set flower-nectar-content blue-energy]
  if flower-nectar-content > blue-energy [set flower-nectar-content blue-energy]
  set flower-handling-time  blue-handling-time
  ]
  if color = yellow [
  if flower-nectar-content < yellow-energy [set flower-nectar-content  yellow-energy]
  if flower-nectar-content < yellow-energy [set flower-nectar-content  yellow-energy]
  set flower-handling-time yellow-handling-time
  ]
  ]
end

to flowers-track-occupation                                   ; ----- if any bees on flower, set occupied? true
  ask flowers
  [
    ifelse any? bees-here
    [set occupied? true]
    [set occupied? false]
  ]
end


to set-labels                                   ; ----- debug: change agent labels
  ifelse bee-nectar-label?
  [ask bees [set label precision bee-nectar-collected 3]]
  [ask bees [set label ""]]
  ifelse flower-nectar-label?
  [ask flowers [set label precision flower-nectar-content 3]]
  [ask flowers [set label ""]]
  ifelse hive-nectar-label?
  [ask hives [set label precision hive-nectar-content 3]]
  [ask hives [set label ""]]
  if bee-label? [
    if bee-label-as = "blue-search-times" [ask bees [set label blue-search-memory]]
    if bee-label-as = "yellow-search-times" [ask bees [set label yellow-search-memory]]
  ]
end


to update-memory  ;; -------- memory-length based update-memory - Active
  ask bees [  ; if bee's memory lists have a new nectar added to 1st list index and it's longer than allowed, last value removed (independent from each other)
      if length yellow-energy-memory > bee-memory-length [set yellow-energy-memory but-first yellow-energy-memory]
      if length blue-energy-memory > bee-memory-length [set blue-energy-memory but-first blue-energy-memory]
      if length yellow-handling-memory > bee-memory-length [set yellow-handling-memory but-first yellow-handling-memory]
      if length blue-handling-memory > bee-memory-length [set blue-handling-memory but-first blue-handling-memory]
      if length yellow-search-memory > bee-memory-length [set yellow-search-memory but-first yellow-search-memory]
      if length blue-search-memory > bee-memory-length [set blue-search-memory but-first blue-search-memory]
      if length flower-landing-memory > 50 [set flower-landing-memory but-first flower-landing-memory]


      if not empty? blue-search-memory [set blue-search-mean mean blue-search-memory]                ; ----- update list averages for OFT threshold
      if empty? yellow-search-memory = false [set yellow-search-mean mean yellow-search-memory]
      if empty? yellow-handling-memory = false [set yellow-handling-mean mean yellow-handling-memory]
      if empty? blue-handling-memory = false [set blue-handling-mean mean blue-handling-memory]
      if empty? yellow-energy-memory = false [set yellow-energy-mean mean yellow-energy-memory]
      if empty? blue-energy-memory = false [set blue-energy-mean mean blue-energy-memory]
      set search-threshold ((blue-energy-mean * yellow-handling-mean )/ yellow-energy-mean )- blue-handling-mean

      if sampling? and length blue-energy-memory >= bee-memory-length and length yellow-energy-memory >= bee-memory-length [  ; ----- if both memory
lists full, end sampling mode
        export-sampling-files
        set ticks-at-sampling-end ticks
        ifelse blue-search-mean >= search-threshold
        [set specialized-at-sampling-end? false]
        [set specialized-at-sampling-end? true]
        set sampling? false
        set threshold-switch-count 0                                   ; ----- reset accepted / encountered monitors now that sampling ended
        set total-count 0
        set blue-encountered 0
        set blue-accepted 0
        set yellow-encountered 0
        set yellow-accepted 0
        set NectarGathered 0
        ask bees [set bee-nectar-collected 0]
        ]   ; if bee visited enough flowers to fill it's list, no longer sampling

      set search-threshold ((blue-energy-mean * yellow-handling-mean )/ yellow-energy-mean )- blue-handling-mean          ; ----- update search-threshold
with new memory values

     if not empty? yellow-energy-memory and not empty? blue-energy-memory [                              ; ----- PropNec: update % values with new memory
values
       let yes mean yellow-energy-memory
       let bes mean blue-energy-memory
   set PropNecYellow precision (yes / (yes + bes)) 3
   set PropNecBlue precision (bes / (yes + bes)) 3
   if seq-update? = true [                                          ; ----- graph: update sequence graphs
    ifelse [color] of last-flower = yellow [
    set last-visit-yellow 1
    set last-visit-blue 0
    set seq-update? false
   ]
   [
    set last-visit-blue 1
    set last-visit-yellow 0
    set seq-update? false
   ]
  ]
 ]
 ]
end


to bee-decide-prey
 if bee-decision-type = "prey-model" [
  let yem yellow-energy-mean
  let yhm yellow-handling-mean
  let bem blue-energy-mean
  let bhm blue-handling-mean
  let lastprey nextprey
  ifelse (yem / yhm) > (bem / bhm)                                    ; ----- determine which flower color is prey1 (best choice)
  [
   set prey1 "yellow"                                   ; ----- if yellow is prey1:
   set prey2 "blue"
   ifelse yellow-search-mean > ((yem * bhm / bem ) - yhm)                  ; ----- determine if bee should generalize or specialize
   [
    set nextprey "both"
   ]
   [
    set nextprey "yellow"
   ]
  ]
  [
   set prey1 "blue"                                     ; ----- if blue is prey1:
   set prey2 "yellow"
   ifelse blue-search-mean > ((bem * yhm / yem) - bhm)                   ; ----- determine if bee should generalize or specialize
   [
    set nextprey "both"
   ]
   [
    set nextprey "blue"
   ]
  ]
  if lastprey != nextprey [set threshold-switch-count threshold-switch-count + 1]                 ; ----- if generalize/specialize changed, increase threshold-
switch-count
  set prey-1 prey1
  set next-prey nextprey
 ]
 if bee-decision-type = "PropNec" [                    ; ----- for PropNec: determine which prey type has higher % nectar in memory, set as prey1
  let yes mean yellow-energy-memory
  let bes mean blue-energy-memory
  ifelse bes >= yes [
   set prey1 "blue"
   set prey2 "yellow"
  ]
  [
   set prey1 "yellow"
   set prey2 "blue"
  ]
  set prey-1 prey1
  set next-prey "NA"
 ]
end




to setup-export                                         ; ----- create export file if one doesn't exist
 set sampling-file-name (word export-file-name "-sampling.csv")
 if file-exists? sampling-file-name = false [create-sampling-files]
 if file-exists? export-file-name = false [create-files]
 if file-exists? export-file-name2 = false [create-files2]
 if file-exists? sampling-file-name = false [create-sampling-files]
end


to export-data ; export-specific 'forever loop' procedure
```

```
 ; if ticks = 1 and current-iteration = 1 [create-files] ; create output spreadsheet, add variable names and definition once per experiment trial
  ;if ticks mod (sim-iteration-length - 1) = 0 [export-files] ; output total variables at the end of each iteration (total nectar gathered by hive1 , etc)
  if (ticks - ticks-at-sampling-end) mod (sim-iteration-length - 1) = 0 [export-files]
end

to create-files ; initial output headers and relevant variable settings at the start of each trial
  let spacer ","
  file-open export-file-name
  file-print(list spacer "Output name: " spacer export-file-name spacer "Date+Time Started: " spacer date-and-time spacer)
    file-print(list spacer "NectarGathered" spacer "total-count" spacer "bee-decision-type" spacer "threshold-switch-count" spacer
      "yellow-energy" spacer "blue-energy" spacer "yellow-handling-time" spacer "blue-handling-time" spacer "yellow-flower-count" spacer
      "blue-flower-count" spacer "bee-memory-length" spacer "yellow-encountered" spacer "yellow-accepted" spacer "blue-encountered" spacer
      "blue-accepted" spacer "handling-time-mode" spacer "search-threshold" spacer "specialized-at-sampling-end?" spacer "world-name" spacer
      "landscapeID" spacer "ticks" spacer "ticks-at-sampling-end" spacer)
  file-close
end

to create-files2 ; initial output headers and relevant variable settings at the start of each trial
  let spacer ","
  file-open export-file-name2
  file-print(list spacer "Output name: " spacer export-file-name spacer "Date+Time Started: " spacer date-and-time spacer)
  file-print(list spacer "blue-handling-time: " spacer blue-handling-time spacer "yellow-handling-time: " spacer
    yellow-handling-time spacer "blue-energy: " spacer blue-energy spacer "yellow-energy: " spacer
    yellow-energy "yellow-flower-count: " spacer yellow-flower-count spacer "bee-memory-length: " spacer bee-memory-length spacer)
  file-print(list spacer "Blue-flower-count" spacer "Blue-search-mean" spacer "inequality" spacer )
  file-close
end




to export-files2
  let spacer ","
  let inequality (((blue-energy-mean * yellow-handling-mean )/ yellow-energy-mean )- blue-handling-mean)
  file-open export-file-name2
  file-print(list spacer blue-flower-count spacer blue-search-mean spacer inequality spacer )
file-close
end

to export-files                                         ; ----- at trial end, export listed variables into a new spreadsheet row, set next-run? true
  if next-run? != true [
  let spacer ","
  let ActualPVL (total-count - switch-count) /  (total-count + 1)
  let MeanTD mean travel-distances
  ifelse [sampling?] of bees = false [file-open (word "sampling-" export-file-name)][file-open export-file-name]
  file-print(list spacer NectarGathered spacer total-count spacer bee-decision-type spacer threshold-switch-count spacer
    yellow-energy spacer blue-energy spacer yellow-handling-time spacer blue-handling-time spacer yellow-flower-count spacer
    blue-flower-count spacer bee-memory-length spacer yellow-encountered spacer yellow-accepted spacer blue-encountered spacer
     blue-accepted spacer handling-time-mode spacer search-threshold spacer specialized-at-sampling-end? spacer
     world-name spacer landscapeID spacer ticks spacer ticks-at-sampling-end spacer)
  file-close
  export-interface (word  "results " bee-memory-length " - " bee-decision-type " - "  random-float 1.0 ".png")
  if export-memory-lists? = true [export-lists]
  set next-run? true
  ]
end

to export-lists                                         ; ----- export entire memory lists in seperate spreadsheet, if needed
  if next-run? != true [
  let spacer ","
  file-open (word export-file-name "-lists.csv")
  file-print (list spacer bee-decision-type spacer bee-memory-length spacer yellow-handling-time spacer
    landscapeID spacer world-name spacer ([blue-search-memory] of bees) spacer)
   file-close
  ]
end

to create-sampling-files
  let spacer ","
  file-open sampling-file-name
end


to export-sampling-files                                         ; ----- export variables at sampling end to seperate spreadsheet, if needed
  if next-run? != true [
  let spacer ","
  file-open sampling-file-name
  file-print(list spacer sampling? spacer bee-memory-length spacer bee-decision-type spacer NectarGathered spacer total-count spacer
    yellow-encountered spacer yellow-accepted spacer blue-encountered spacer blue-accepted spacer threshold-switch-count spacer
    yellow-energy-mean spacer blue-energy-mean spacer yellow-search-mean spacer blue-search-mean spacer search-threshold spacer specialized? spacer
landscapeID spacer world-name spacer )
  ]
end
```

```
to update-label ; change the bee's label during the simulation with the drop-down menu 'bee-label-as' , allows real-time diagnosing
  ifelse bee-label? [
    if bee-label-as = "blue-search-times" [ask bees [set label blue-search-memory]]
    if bee-label-as = "yellow-search-times" [ask bees [set label yellow-search-memory]]
  ]
  [
    ask bees [ set label ""]
  ]

end
```