Overview, Design concepts, and Details for the model Creating Intelligent Agents

Dale K. Brearcliffe[1[0000-0003-0142-000X]] and Andrew Crooks[1,2[0000-0002-5034-6654]]

[1] George Mason University, Fairfax VA 22030, USA
[2] University at Buffalo, Buffalo NY 14261
dbrearcl@gmu.edu & atcrooks@buffalo.edu

Within this document, we provide a description of the Agent-Base Model (ABM) "*Creating Intelligent Agents*" which is based on the updated Overview, Design concepts, and Details (ODD) protocol developed by Grimm *et al*. (2020). The objective of this ODD is to provide the reader a human comprehensible description of the workflow, code, and intent behind the design of the ABM. The ABM was designed using NetLogo (Wilensky, 1999, Version 6.1) and provides the user with a Graphical User Interface (GUI) as presented in Figure 1. The model itself can be found at: https://tinyurl.com/ML-Agents.

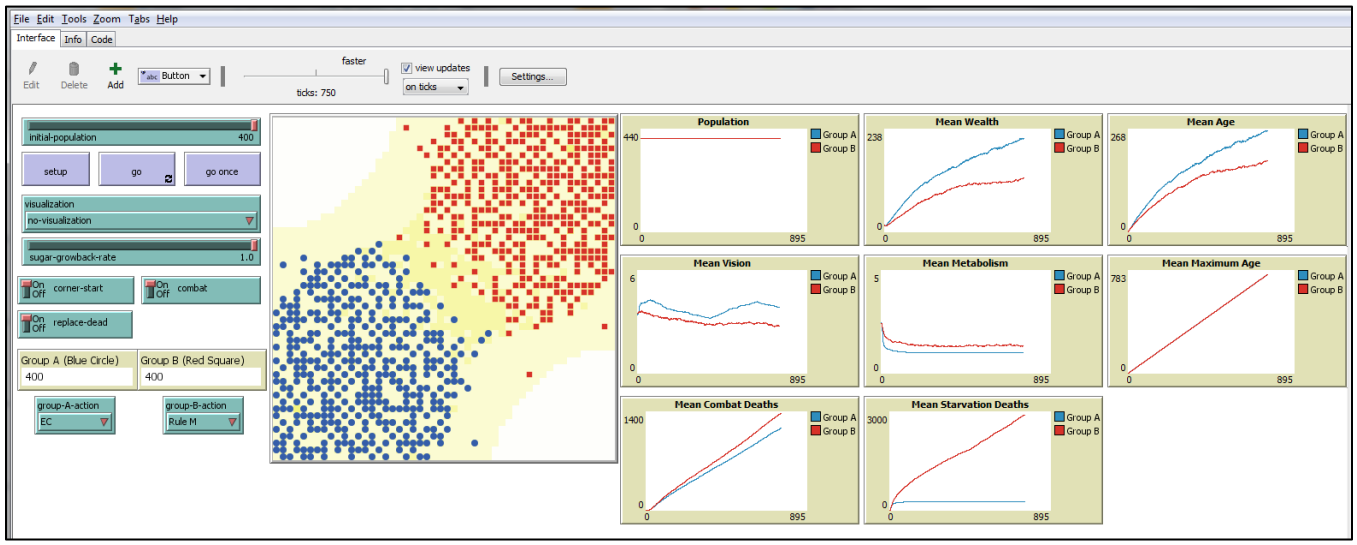

Figure 1 - Graphical User Interface of the "*Creating Intelligent Agents"* model. From left to right: input parameters, agents within their artificial world, and aggregate model outputs.

1. PURPOSE AND PATTERNS

The primary purpose of this model is to demonstrate the methods by which intelligent agents (i.e., agents which utilize machine learning techniques) can be incorporated into an ABM. It does so by contrasting agents using *a priori* rules with agents that evolve physical characteristics over time and agents that learn successful and unsuccessful decisions over time. The agents operate in an extension of the Sugarscape (Epstein and Axtell, 1996) ABM that has been used to explore migration, trade, wealth inequality, disease processes, sex, culture, and conflict. Conflict between two groups, where friends are in the same group and enemies are in the other, is the focus of the model as the individual agents seek to accumulate wealth in the form of sugar. The Sugarscape model, "*Sugarscape 2 Constant Growback*" (LI and Wilensky, 2009), included in the NetLogo model library was the foundation for this model's development. We chose Sugarscape (Epstein and Axtell, 1996) as it is well known and

understood in the agent-based modeling community and therefore would allow us to demonstrate how different learning techniques can be incorporated into an ABM without having to describe in detail the model itself. The secondary purpose of this model is to judge whether resources (software development time, wall clock time, and analytic time) expended on using intelligent agents is worth the effort. The final purpose of this model is to determine which of the four types of agents (i.e., *Rule M*, *Q-Learning*, *SARSA* and *EC* agents) are more successful, where success is measured by accumulated wealth (i.e., sugar).

Pattern 1: It is only the third purpose that provides an observable outcome from the model. The ability of the agents with *a priori* assigned rules (i.e., *Rule M*) is placed in conflict with the three types of learning agents (i.e., *Q-Learning*, *SARSA*, and *EC*). These types of learning agents are also placed in conflict with each other. From these conflicts, a variety of meta-patterns can be measured and compared. This involves a pairwise comparison of the four agent types for outcomes of mean wealth, mean vision, mean metabolism, cumulative combat deaths, cumulative starvation deaths, mean age, and maximum age (see Figure 1 and Section 7 for more details).

2. ENTITIES, STATE VARIABLES, AND SCALES

2.1. Entities

There are seven types of entities included in this model. Four are agents, one is a collective, one is a grid cell, and one is environmental (see Section 5 for entity initialization details).

1) *Standard (Rule M)*[1] agents have hard coded rules imposed on them *a priori* to the model execution. The rule set does not change during model execution. Replacement agents are assigned initial attribute values using the same process used by agents at the beginning of the model (Section 7.1).
2) *Evolutionary Computing (EC)* agents also have hard coded rules imposed on them *a priori* to the model execution. However, some of the attributes of replacement agents are based upon the most successful agents of a collective rather than random values (which will be discussed in more detail in Section 7.2).
3) *Q-Learning* agents have no *a priori* assigned rules. Instead, they have a decision matrix showing the actions they can take, given their state (see Section 7.5). Initially the matrix is blank, and agents make random decisions for which they receive a reward from an *a priori* provided reward table. Over time the decision table changes based on the best decisions and random explorations of the agent. These agents may ignore the decision table for future actions (within reinforcement learning this is known off-policy decision-making, see Section 7.3).
4) *State-Action-Reward-State-Action (SARSA) Learning* agents are an extension of *Q-Learning* agents. They are much the same, except their decisions use the same policy that generates the current action to generate the next action (within reinforcement learning this is known as on-policy decision-making, see Section 7.4).
5) *Breed (Collective)* are groups of agents of the same agent type (Section 4.10).
6) *Grid Cells (Patches - A NetLogo term)* are spatial locations representing equal portions of a two-dimensional world. See Section 5 for initialization details.
7) *Tick (Environmental)* is a discrete interval (Stevens, 1946) unit of equal time.

[1] Rule M refers to the designation of this type of agent used by Epstein and Axtell (1996).

2.2. State Variables and Scale

There are two *Collectives (Breeds - A NetLogo term)* active in the model during initialization and the execution of the model; these are labeled "Group A" and "Group B." Each collective holds only one type of agent (i.e., *Rule M*, *Q-Learning* which is chosen by the modeler at model initialization) and both *Collectives* can have the same entity type (e.g., *Q-Learning* and *Q-Learning*). All agents (i.e., *Rule M*, *EC*, *Q-Learning*, and *SARSA*), are each initialized (Section 5) with different attribute values. The attributes common amongst the agents are sugar, metabolism, vision, and age. *Q-Learning* and *SARSA* agents have additional attributes that hold their current state and action (which are discussed more in Sections 7.3 and 7.4 respectively). The *Q-Learning* and *SARSA* learning agents also have a Q-Value matrix that represents their accumulated tacit knowledge of past decisions. *Grid Cells* are initialized (see Section 5) with sugar values (psugar) that change during model execution. The sugar values represent two piles, one in the northeast corner and the other in the southwest corner of a non-toroidal grid sized fifty-by-fifty cells. Finally, *Ticks* are temporal intervals of equal value that increase monotonically during model execution. Each *Tick* has no equivalency to wall clock time. The values these state variables can assume are listed in Table 1.

Table 1 - Model State Variables

Attribute Name	Represents	Type	Value Range
action	The current action of the agent.	Integer	[0,3]
age	The number of ticks an agent has existed.	Integer	[0,20000]
breed	Which of the two groups an agent belongs to.	Ordinal	{Group A, Group B}
metabolism	The amount of sugar that each agent loses (consumes) each tick.	Integer	[1,4]
psugar	The amount of sugar in a Grid Cell.	Decimal	[0,4] step 0.1
q-values	The matrix of Q-Values. (state versus action)	Matrix [3x4] of Float	[0,+∞)
state	The current state of the agent.	Integer	[0,2]
sugar	The amount of sugar the agent has.	Decimal	[0.1,+∞) step 0.1
tick	A unit of equal time.	Integer	[0,20000]
vision	The distance (in cells) that an agent can see in the horizontal and vertical directions directly in line with the agent's cell location (von Neumann neighborhood).	Integer	[1,6]

3. PROCESS OVERVIEW AND SCHEDULING

After initialization (see Section 5), the model proceeds in equal interval (*Ticks*) of time. At each time step, an agent is selected using a random uniform distribution from all agents (both *Collectives*) that have yet to be selected during the current time interval. This continues until all agents have been selected. Each of the four types of agents takes a different path in the model execution as show in Figure 2 (and discussed in more detail in Section 7). There are only two agent types in use during any model run so only two of these paths will be followed. The *Rule M* agent, as noted above (Section 2.1) is based on the original Sugarscape model (Epstein and Axtell, 1996), attacks if it is in a Strong position (see Section 4.3) or Jumps to a random *Grid Cell* if not. The *EC* agent is similar but will Retreat if in a Weak position (see Section 4.3). The *Q-Learning* agent's decision path is more complicated. It has a fourth available action of staying stationary, and makes its decision (i.e., Attack, Retreat, Jump, or Stay) based on its experience of the three state types (i.e., Strong, Weak, No Contact) using an off-policy method to select an action (see Section 2.1). The *SARSA* agent's path is the same as the *Q-Learning* agent's with one difference, it uses an on-policy method to select an action (see Section 2.1). After all

agents have been selected, the execution paths of the agents converge with the replacement of agents that have been attacked (i.e., killed). *EC* agents have an additional step whereby new agents are based on mutated versions of the wealthiest agents of their *Collective* (see Section 4.3). The model ends when either all agents in a *Collective* are dead or maximum time has been reached.

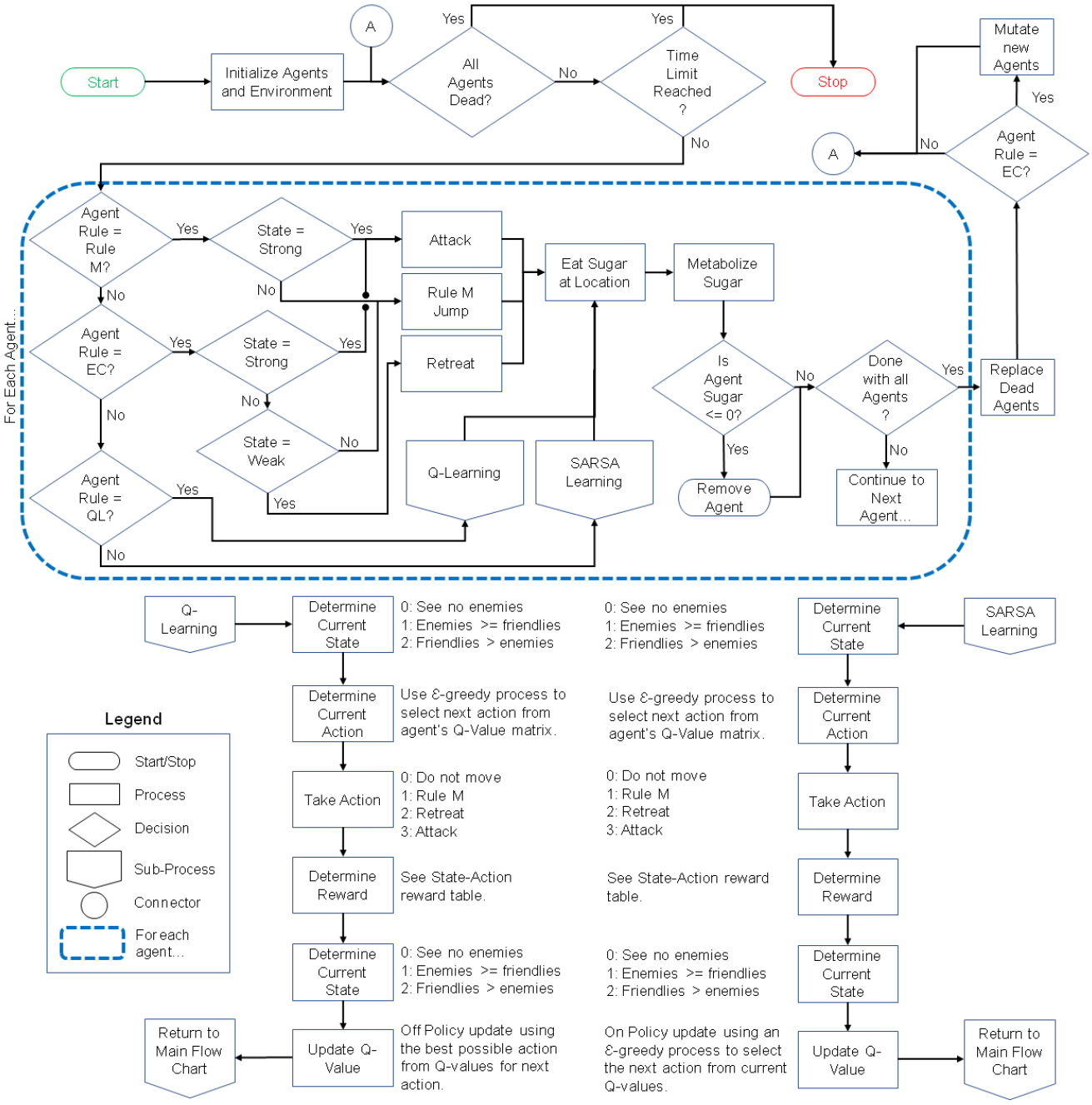

Figure 2 - Model execution flowchart.

4. DESIGN CONCEPTS

4.1. Basic principles

The model was developed to demonstrate the differences between traditional rule-based approaches commonly used within agent-based modeling and learning agents and to demonstrate how learning agents can be integrated into an agent-based model. The design of the simple agents, (i.e., *Rule M*), was based on the Sugarscape (Epstein and Axtell, 1996) model. The *EC* agents came from both Sugarscape (Epstein and Axtell, 1996) and Genetic Algorithms (GA) more generally, that were first described by Holland (1975). *Q-Learning* and *SARSA* agents use Reinforcement Learning (Sutton, 1988; Sutton and Barto, 2018). These four approaches are realized at the level of individual agents and are implemented by different sub-models (see Section 7). During a model execution, no more than two *Collectives* (Section 4.10) with each having only one agent type (Section 2.1) such as *Rule M* or *SARSA* compete against each other at the individual level to accumulate wealth (sugar) at the *Collective* level.

4.2. Emergence

A pairwise comparison of the four agent types creates sixteen sets of results over time for each of mean wealth, mean vision, mean metabolism, cumulative combat deaths, cumulative starvation deaths, mean age, and maximum age (as shown on the right of Figure 1). It is from these outcomes a determination can be made for how well the different learning agents fared against each other and against the *Rule M* agents. The emergence of unexpected results, such as decrease of vision distance or lack of combat deaths, is also found in these outcomes.

4.3. Adaptation

All agent types use the same method to assess their current state (see Section 7.5). They examine the *Grid Cells* horizontally and vertically that are directly in line with the agent itself (i.e., they use a von Neumann neighborhood) and counts the number of other agents, friendly and enemy within their vision range (Section 4.7). If an agent sees no other agents, it places them in a No Contact state. If there are the same or less friendly agents than enemy agents within their vison, they are in a Weak state. Lastly, if more friendly agents than enemy agents are seen, they are in a Strong state. Decision-making is the key difference between the actions (e.g. Attack vs. Stay, see Section 7.6 for more actions) of the four types of agents. The *Rule M* agents adopt an Attack action when in a Strong state. Otherwise, they take a Jump action, moving to a location with the highest sugar within its vision range. The *EC* agent does the same but has an additional Retreat action when in a Weak state that moves them back to the spawn area of their *Collective* (see Section 4.10). In addition, the *EC* agents modify the initialization values for metabolism and vision when replacing dead agents, copying values from high wealth agents. Vision effects their ability to see sugar, plus friendly and enemy agents. Metabolism affects their ability to survive without new sugar sources. Both of these agents use explicit knowledge provided by the modeler (see Sections 7.1 and 7.2).

Q-Learning and *SARSA* agents make their decisions based on their Q-Value matrix that hold probabilities for four actions given one of three states. The actions are the same as the previous two agents with a fourth action to Stay in place. This 3x4 matrix is initially empty, but as agents take actions and observe results it is updated with new probability values. At first, the agents have a higher probability of exploring instead of using their best, past decisions. As the agents grow older, they focus

on their past knowledge. The experience gained by each agent is tacit knowledge (see Sections 7.3 and 7.4).

4.4. Objectives

The primary objective measure of this model is the accumulation of wealth in the form of sugar. Agents of all types accumulate sugar individually that is measured as mean wealth over time at the *Collective* level. Agents gather wealth when they move to a *Grid Cell* that contains sugar. They expend sugar based on their individual metabolism.

4.5. Learning

Three of the four agent types (i.e., *EC*, *Q-Learning*, and *SARSA*) use learning to change their decision making over time. The *Rule M* agent does not learn (however, it does adapt - see Section 4.2) and simply follows a set of rules (i.e., explicit knowledge) provided by the modeler. The *EC* agents learn as a *Collective* using evolutionary computing (Holland, 1975). As *EC* agents die and are replaced, their metabolism and vision attribute values are initialized based on those agents in their *Collective* that have the most wealth rather than a random value. The mean vision and metabolism attributes for the *Collective* converge to a local maximum that represents a best choice for wealth accumulation. Reinforcement learning (Sutton and Barto, 2018) drives the remaining two agent types. *Q-Learning* and *SARSA* agent types learn individually by accumulating tacit knowledge, storing their experience in a Q-Value matrix. Initially, these agents explore their world randomly while updating their experience. As they age, they increasingly use their experience to make decisions until doing so 95% of the time. In a sense, as the agents get older, they explore less and follow their tacit knowledge more. The difference between these two agent types is how they update their experience. *Q-Learning* agents make off-policy updates. This allows them to follow their existing policy in their Q-Value matrix, and then update their experience by looking outside the policy. *SARSA* agents make on-policy updates. They follow and update their experience using their existing policy.

4.6. Prediction

Rule M agents make no predictions of the future, simply living for the moment. The *EC Collective* implicitly predicts that wealthy agents are better role models and shift their vision and metabolism attribute values toward those agents. One of the fundamental concepts of reinforcement learning used by *Q-Learning* and *SARSA* agents is to explicitly calculate future values for all state-action pairs and to update their experience to maximize the best outcome.

4.7. Sensing

Common to all agent types is self-localization to the *Grid Cell* they occupy. From this location, agents can examine other *Grid Cells* above, below, left, and right, up (i.e., von Neumann neighborhood) to a limit held by their vision attribute. In each of these cells, an agent can sense the amount of sugar, friendly agents, and enemy agents with absolute accuracy. The vision attribute values are based on the original Sugarscape model of Epstein and Axtell (1996).

4.8. Interaction

There is one type of direct interaction available to *Rule M* and *EC* agent types that occur when an agent decides to Attack an enemy agent. This combat interaction occurs between agents in different *Collectives*. The decision to Attack is an indirect influence based on observations made of friendly and

enemy agents within sensing range (Section 4.7). If an agent senses more friendly agents than enemy agents it will Attack. Another indirect influence occurs when no enemy agents are sensed by a *Rule M* or *EC* type agent. These agents will Jump to the nearest *Grid Cell* within their sensing range that has the most sugar and eat the sugar. The act of eating the sugar has an indirect influence on future agent decisions. *EC* type agents have an additional indirect interaction if they sense the same or more enemy agents than friendly agents. The agents will Retreat to their *Collective's* spawn area (Section 4.10).

Q-Learning and *SARSA* type agents have all actions available to them as seen in Table 2 and include an additional action, doing nothing, which may indirectly influence other agents. The *Q-Learning* and *SARSA* agents' decision to interact is based on the state they are in and the tacit knowledge stored in the Q-Value matrix. Table 2 summarizes the possible actions for the various agent types, given a state, that each of the agent types can take. Note that *Rule M* cannot discern the difference between Weak and No Contact states and will Jump in either state.

Table 2 - Possible actions given a state by agent type. RM = Rule M, EC = Evolutionary Computing, QL = Q-Learning, and S = SARSA. The Attack is the only direct interaction.

		Action			
		None	Jump	Retreat	Attack
State	No Contact	QL, S	RM, EC, QL, S	QL, S	QL, S
	Weak	QL, S	RM, QL, S	EC, QL, S	QL, S
	Strong	QL, S	QL, S	QL, S	RM, EC, QL, S

4.9. Stochasticity

Stochasticity is used to initialize the attributes of newly created agents, select the activation order of agents, select initial and re-spawn *Grid Cells*, evolve *EC* agents, and make *Q-Learning* and *SARSA* agent action decisions. Uniform distributions are used for all random value selections.

4.10. Collectives

Within the model, the two *Collectives* are labeled Group A and Group B. Either holds exactly one agent type that begin in spawn areas of the southwest most 21x21 *Grid Cells* and northeast most 21x21 *Grid* Cells respectively. The *Collectives* consider each other's agents to be enemies for combat actions.

4.11. Observation

From an observation perspective, the *Grid Cells* provide a view of all agent locations and provide an opportunity for anecdotal observation of interactions. We use plots as shown in Figure 1 (right side) to capture mean wealth, mean vision, mean metabolism, cumulative combat deaths, cumulative starvation deaths, mean age, and maximum age for each *Collective* and the data generated can be collected using NetLogo’s BehaviorSpace tool for later analysis.

5. INITIALIZATION

During initialization, an equal number of agents are created for each *Collective* as determined by user input. Their common state attribute values are drawn from a random uniform distribution for sugar, metabolism, and vision or specific values for age, state, action, epsilon, and the Q-Value matrix. Unique to each *Collective* (Section 4.10) is the agent's color, name, and a stochastically assigned location

restricting an agent to be with others of their *Collective*. The *Grid Cell* psugar attributes are assigned specific values that are drawn from the provided file "symmetric-sugar-map.txt" for the 50x50 grid. These values are symmetrical across the major axis drawn from the northwest to southeast corners.[2] *Tick* is set to zero.

6. INPUT DATA

The model does not use input data to represent time-varying processes.

7. SUBMODELS

There are four sub-models representing each of the agent types (i.e., *Rule M*, *Q-Learning*, *SARSA,* and *EC* agents) and the actions that can be taken. Figure 2 shows how these sub-models are positioned in the process flow. The Sugarscape model, "Sugarscape 2 Constant Growback" (LI and Wilensky, 2009), included in the NetLogo models library was the foundation for the model development. An iterative and incremental programming methodology was used to add new code into the existing model while removing code that was not needed and modifying existing code to support the changes.

7.1. Rule M

Rule M type agents use explicit knowledge in the form of *a priori* rules imposed by the modeler to determine actions (Section 7.6) given its state (Section 7.5). These agents can be in one of two states: 1) Strong or 2) No Contact (This agent type has no rule for a Weak state to more closely match combat in the original Epstein and Axtell (1996) description.). If its state is Strong, it Attacks following the process described in Section 7.6.1. Otherwise, it Jumps following what is discussed in Section 7.6.3. These three steps are listed below:

1. Determine the current state (Section 7.5).
2. If the state is Strong, take action Attack (Section 7.6.1).
3. Otherwise, take action Jump (Section 7.6.3).

7.2. Evolutionary Computing (EC)

EC type agents use explicit knowledge in the form of *a priori* rules imposed by the modeler to determine actions (Section 7.6) given its state (Section 7.5). These agents can be in one of three states: 1) Strong, 2) Weak, or 3) No Contact. If its state is Strong, it Attacks following the process in Section 7.6.1. If Weak, it Retreats uses the process found in Section 7.6.2. Otherwise, it Jumps following Section 7.6.3. These four steps are listed below:

1. Determine the current state (Section 7.5).
2. If the state is Strong, take action Attack (Section 7.6.1).
3. If the state is Weak, take action Retreat (Section 7.6.2).
4. Otherwise, take action Jump (Section 7.6.3).

Once all agents in the two *Collectives* (see Section 4.10) have been selected and followed the above steps, *EC* replacements for dead agents are first initialized then their vision and metabolism attribute values are replaced with new values taken from the two *EC* agents with the highest sugar. A random uniform distribution is used to select one of each of the two values for these attributes.

[2] The original sugar topology in Epstein and Axtell (1996) is not symmetrical as can be seen in Figure II-1 using a visual diagnostic (Epstein and Axtell, 1996, p. 22).

7.3. Q-Learning

Q-Learning agents use reinforcement learning (Sutton, 1988; Sutton and Barto, 2018) to gather tacit knowledge over time and store their experience in a 3x4 Q-Value matrix. The matrix is an intersection of three states (Section 7.5) and four actions (Section 7.6). This agent type makes its decision, takes the appropriate action, and updates its experience using the following procedure:

1. Determine the current state (Section 7.5).
2. Calculate the probability (Equation 1) of exploring alternate actions in lieu of the best action. This probability ϵ is based on the age of the agent so younger agents have a higher probability than older agents do. There is a hard limit of 5% for exploration that an agent will not go below. This calculation occurs in every *tick* so *t* is constantly increasing monotonically.

$$\epsilon = \frac{1}{1 + e^{\frac{t-5000}{1000}}} \quad (1)$$

where *t* is the age of the agent in *ticks*

3. Create a four-value probability vector (one for each action) with each holding the value ϵ divided by four. The sum of the probability vector is less than one. Examine the Q-Value matrix for the current state (from Step 1) and find the action with the highest value. If there is more than one action with the same highest value, one is selected using a random uniform distribution. The corresponding action in the probability vector is changed to A_{new} (Equation 2) using the best action value from the Q-Value matrix. The sum of the probability vector is now one.

$$A_{new} = QV_{max} + 1.0 - \epsilon \quad (2)$$

where ϵ is from Equation 1 and QV_{max} is the best Q-Value matrix value for the state

4. Use the probability vector to select an action. The previous step will slightly increment the best action to a higher probability as the agent grows older.
5. Take the action using the appropriate sub-model from Section 7.6.
6. Determine the Reward the agent receives for its action given its state. The Reward table (Table 3) is the means by which the modeler influences the reinforcement learning agent's actions. They are established *a priori* as part of the model design. Here it is based on the amount of sugar in the *Grid Cell* (Patch Sugar) and the maximum amount of sugar the *Grid Cell* can hold (Max Sugar) with extra encouragements and discouragement for some actions.
7. Determine the new state (Section 7.5) now that an action has taken place.
8. Given the new state, find the new best action in the Q-Value matrix.
9. Update the Q-Value matrix.
    a. Calculate the future Q-Value using the new state and new action from steps 7 and 8, the Reward from Step 6, and hyper-parameter γ as shown in Equation 3.

$$QV_{future} = QV_{future} \times \gamma + Reward \quad (3)$$

where the hyper-parameter γ is the future discount and QV_{future} is a value obtained from the Q-Value matrix using new state and new action

    b. Calculate the difference (error) between the future Q-Value (Step 9.a) and the current Q-Value using current state and action using Equation 4.

$$E = QV_{future} - QV_{current} \quad (4)$$

where QV_{future} is a value obtained from the Q-Value matrix using new state and new action, $QV_{current}$ is a value obtained from the Q-Value matrix using current state and current action

c. Set the Q-Value for the current state and action to that value plus the learning rate times the error from Step 9.b as shown in Equation 5.

$$QV_{current} = QV_{current} + \lambda \times E \quad (5)$$

where $QV_{current}$ is a value obtained from the Q-Value matrix using current state and current action, the hyper-parameter λ is the learning rate, and *E* is the error from the previous step

10. Finally, the agent's state is set to the new state.

Table 3 - State-Action Rewards

Rewards		Action			
		Stay	**Jump**	**Retreat**	**Attack**
State	**No Contact**	Patch Sugar	Patch Sugar	-100	-100
	Weak Position	Patch Sugar - Max Sugar	Patch Sugar + Max Sugar	Patch Sugar + Max Sugar x 10	Patch Sugar - Max Sugar x 2
	Strong Position	Patch Sugar	Patch Sugar + Max Sugar	Patch Sugar - Max Sugar x 2	Patch Sugar + Max Sugar x 10

7.4. SARSA (State-Action-Reward-State-Action)

SARSA agents use reinforcement learning (Sutton, 1988; Sutton and Barto, 2018) to gather tacit knowledge over time and store their experience in a 3x4 Q-Value matrix. *SARSA* is an extension of *Q-Learning* with an additional State-Action that uses on-policy decision making instead of off-policy. The matrix is an intersection of the three states (Section 7.5) and four actions (Section 7.6). This agent type makes its decision, takes the appropriate action, and updates its experience using the following procedure:

1. Determine the current state (Section 7.5).
2. Calculate the probability (Equation 1) of exploring alternate actions in lieu of the best action. This probability ϵ is based on the age of the agent so younger agents have a higher probability than older agents do. There is a hard limit of 5% for exploration that an agent will not go below. This calculation occurs in every *tick* so *t* in Equation 1 is constantly increasing monotonically.
3. If this agent is aged zero (i.e., brand new) then:
    a. Create a four-value probability vector (one for each action) with each holding the value ϵ divided by four. The sum of the probability vector is less than one. Examine the Q-Value matrix for the current state (from Step 1) and find the action with the highest value. If there is more than one action with the same highest value, one is selected using a random uniform distribution. The corresponding action in the probability vector is changed to A_{new} (Equation 2) using the best action value from the Q-Value matrix. The sum of the probability vector is now one.
    b. Use the probability vector to select an action. The previous step will slightly increment the best action to a higher probability as the agent grows older.
4. Take the action using the appropriate sub-model from Section 7.6.

5. Determine the Reward the agent receives for its action given its state. The Reward table (Table 3) is the means by which the modeler influences the reinforcement learning agent's actions. They are established *a priori* as part of the model design. Here it is based on the amount of sugar in the *Grid Cell* (Patch Sugar) and the maximum amount of sugar the *Grid Cell* can hold (Max Sugar) with extra encouragements and discouragement for some actions.
6. Determine the new state (Section 7.5) now that an action has taken place.
7. Create a four-value probability new state vector (one for each action) with each holding the value ϵ divided by four. The sum of the probability vector is less than one. Examine the Q-Value matrix for the new state (from Step 6) and find the action with the highest value. If there is more than one action with the same highest value, one is selected using a random uniform distribution. The corresponding action in the probability new state vector is changed to A_{new} (Equation 2) using the best action value from the Q-Value matrix. The sum of the probability vector is now one.
8. Use the probability new state vector to select a new action. The previous step will slightly increment the best action to a higher probability as the agent grows older.
9. Update the Q-Value matrix.
    a. Calculate the future Q-Value using the new state and new action from steps 6 and 8, the Reward from Step 5, and hyper-parameter γ as shown in Equation 3.
    b. Calculate the difference (error) between the future Q-Value (Step 9a) and the current Q-Value using current state and action using Equation 4.
    c. Set the Q-Value for the current state and action to that value plus the learning rate times the error from Step 9b as shown in Equation 5.
10. The agent's state is set to the new state.
11. The agent's action is set to the new action.

7.5. States

The state an agent is in either determines (for *Rule M* and *EC*) or suggests (for *Q-Learning* and *SARSA*) possible actions that can be taken. There are three possible states: Strong, when an agent senses more friendly agents than enemy; Weak, when an agent senses the same or more enemy agents than friendly; and, No Contact, when an agent senses no enemy agents.

7.6. Actions

All agents can take an action based on their current state. Table 2 provides a breakdown of these actions.

7.6.1. Action: Attack

Once the decision to Attack is made, an agent senses (Section 4.7) the enemy agents within their vision range and selects the closest one occupying a *Grid Cell* with the most sugar. If there is more than one such *Grid Cell*, the agent selects one using a random uniform distribution. The agent then moves to the selected *Grid Cell*, removes (kills) the enemy agent, and receives a sugar bonus.

7.6.2. Action: Retreat

An agent executing a Retreat selects an empty *Grid Cell* in its *Collective's* (Section 4.10) spawn area and moves to that location. If there are multiple available locations, the agent selects one using a random uniform distribution.

7.6.3. Action: Jump

Based on the original Epstein and Axtell (1996) movement for *Rule M*, an agent selects the closest unoccupied *Grid Cell* with the most sugar and moves to that location. If there are multiple available locations, the agent selects one using a random uniform distribution.

7.6.4. Action: Stay

The agent remains in its current *Grid Cell* and eats any sugar that has re-grown at its location.

REFERENCES

Epstein, J. M., and Axtell, R. (1996). *Growing Artificial Societies: Social Science from the Bottom Up*. Washington, DC, USA: The Brookings Institute.

Grimm, V., Railsback, S. F., Vincenot, C. E., Berger, U., Gallagher, C., DeAngelis, D. L., . . . Radch. (2020). The ODD Protocol for Describing Agent-Based and Other Simulation Models: A Second Update to Improve Clarity, Replication, and Structural Realism. *Journal of Artificial Societies and Social Simulation,* 23(2): 7. Available from: http://jasss.soc.surrey.ac.uk/23/2/7.html

Holland, J. H. (1975). *Adaptation in Natural and Artificial Systems*. Ann Arbor, Michigan, USA: The University of Michigan Press.

LI, J., and Wilensky, U. (2009). NetLogo Sugarscape 2 Constant Growback model. Evanston, IL, USA: Center for Connected Learning and Computer-Based Modeling, Northwestern University.

Stevens, S. S. (1946). On the Theory of Scales of Measurement. *Science*, 103(2684): 677 - 680.

Sutton, R. S. (1988). Learning to Predict by the Methods of Temporal Differences. *Machine Learning*, 3(1): 9-44.

Sutton, R. S., and Barto, A. G. (2018). *Reinforcement Learning: An Introduction* (Second ed.). Cambridge, Massachusetts, USA: The MIT Press.

Wilensky, U. (1999). NetLogo. Evanston: Center for Connected Learning and Computer-Based Modeling, Northwestern University.