Investigating Simple Adaptive Agent Cognition through Neuro-Evolution and Competitive Environments

William Daniels

Introduction

Simulating agents to collectively solve complex tasks is a long-known strength of agent-based modeling approaches (Wilensky and Rand, 2015). Where a centralized cognition alone cannot achieve optimal results, individual agent cognition can excel. But even when deciding to rely on individual agent cognition to solve some particular task, there are many frameworks to choose from. Stuart Russel and Peter Norvig detail the four types of agent cognition: reflexive, utility-based, goal-based, and adaptive (learning) (Russel and Norvig, 1995). Adaptive cognition can be reflected upon on the agent-population level, where "fitness" is increased incrementally over time through evolution given some selection pressure. In evolution by natural selection, chance mutations in genes affect the fitness of an individual, which in turn give that individual's genes a higher chance to propagate throughout the environment (Darwin, 1859). Computer scientists have long sought to replicate evolutionary mechanisms in software, giving us metaheuristics such as genetic algorithms (Koza, 1992) and complex decision makers in the form of neural networks optimized through neuroevolutionary techniques (Edmund and March, 1994).

In previous works, researchers have taken advantage of adaptive agent cognition in the form of neuroevolution to complete complex tasks not easily solved through some central agent planner or rule-based agent heuristic. (Dynamic environments are usually an indicator towards the usefulness of adaptive cognition.) Regarding physical environments, roboticists have leveraged neuroevolution to search the cognition space of robot swarms (Stonedahl, 2017), and through using agent-based modeling software, computer scientists have experimented with mutating neural network weights to increase agent fitness in classic population dynamics models (Head et al., 2015). Especially regarding population dynamics, agent-based models determine

agent actions on some type of random behavior, such as choosing a random or pseudo-random direction to move in. This type of agent cognition ”can be brittle in the context of the unpredictable and emergent situations in which agents may find themselves” (Head and Wilensky, 2018). There is therefore a need to investigate flexible agent cognition at the structural level.

In this paper, I present a simple analysis of using and scaling single- and multi-layer neural networks as adaptive cognition for competitive, reward-driven agents. There is, to my knowledge, a lack of literature that probes the effectiveness of different sizes of small neural networks as simple agent cognition, and the limits of expressiveness of these networks in terms of fitness outcomes given a simple competitive environment.

The General Model

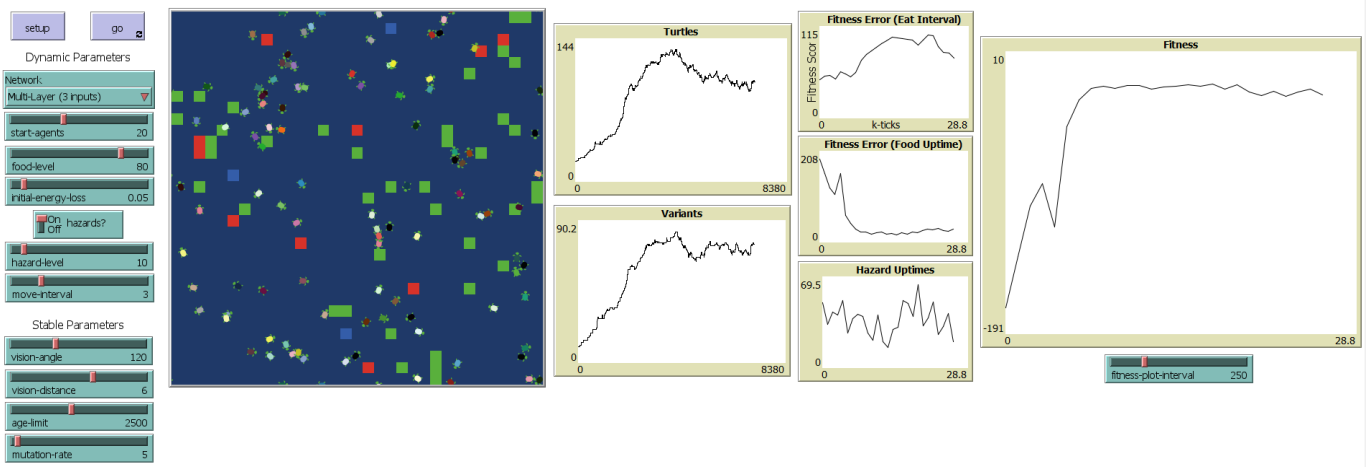


Figure 1: Layout of the model.

Environment, Patches, and Turtles

The spatial environment is modeled by three types of patches: sea (blues patches), food (green patches), and hazards (red patches). Sea patches are the background of the model and have no functional purpose other than to cosmetically represent water, while green patches are representations of 'food' to be competed after by the turtle agents. Hazard patches are patches that should be avoided by the turtle agents as they result in instant energy loss when encountered. Food is randomly placed in the environment every tick until the *food-level* food patches are present. Food patches have their own *patch-birth-tick* variable, denoting the tick they were spawned in at.

Turtle agents have a large number of variables associated with each. The *fitness* variable is the mean of the *eat-times* variable, which is the last *fitness-plot-interval* time intervals the agent has gone without eating. *Weights* stores the agent's personal neural network weights. *Energy* is the current energy the agent has. *Turtle-birth-tick* denotes the tick the agent was

hatched at. Each turtle also has a set of 8 perception variables: *ahead*, *ahead-left*, *ahead-right*, *left-side*, *right-side*, *behind-left*, *behind-right*, and *behind*. These hold the distance of the agent from food and hazards (hazards are represented as negative distance) in each respective perceptive region.

Model Rules

The general **go** loop is as follows: Agents die if they are exhausted or too old. They eat if they come across a food patch. They reproduce (hatch an identical agent that has a chance to mutate their network *weights*) once their *energy* exceeds 100. They perceive their immediate surroundings and move based on their perception inputs. The environment is regulated (the amount of food and hazards in the environment is kept at the *food-level* value and the *hazard-level* value respectively), and turtle *fitness* is plotted. Detailed model rules are present in the appendix.

This loop, at a high level, resembles the generational loop of agents in evolution by natural selection, excluding any explicit predatory aspects (agents preying on other agents or parasitic relationships). Agents must compete to stay alive, which happens to coincide with spreading of genes similar to its own, since staying alive means eating before other agents can, and eating means the agent will reproduce agents more or less similar to them. Having this structure should facilitate progress towards high fitness from the point of view of the population as a whole, which is what I want to investigate with respect to network capacities and environment complexity.

Adaptive Agent Cognition: Neural Network Brains

The type of adaptive agent cognition chosen to investigate is in the form of four structurally different feed-forward neural networks (perceptrons). All network architectures output three possible actions: turn left 20 degrees, turn right 20 degrees, or do nothing. Two networks, a single-layer and a multi-layer (two-layer), take the first 3 perception inputs (*ahead*, *ahead-left*, *ahead-right*), while the other two (again a single- and multi-layer network) take all 8 perception inputs.

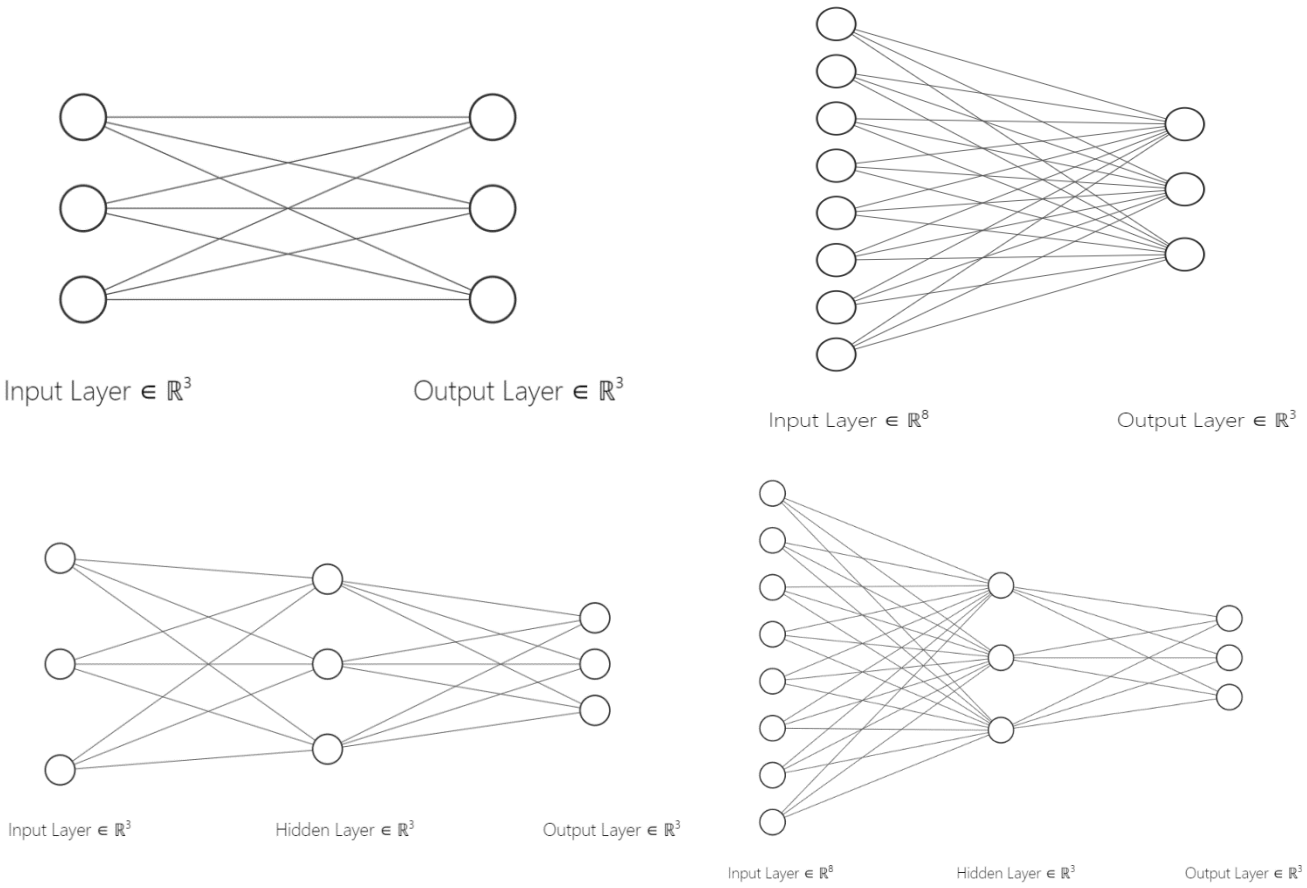


Figure 2: The four network architectures.

Perception inputs are cones of *vision-distance* radius and *vision-angle* angle from the center of the agent. The *ahead* perception looks toward 0 degrees given the agents center, and perception intervals are in increments of 45 degrees. So, the *ahead-left* cone looks towards -45 degrees, *ahead-right* looks towards 45 degrees, *left-side* looks towards -90 degrees, etc.

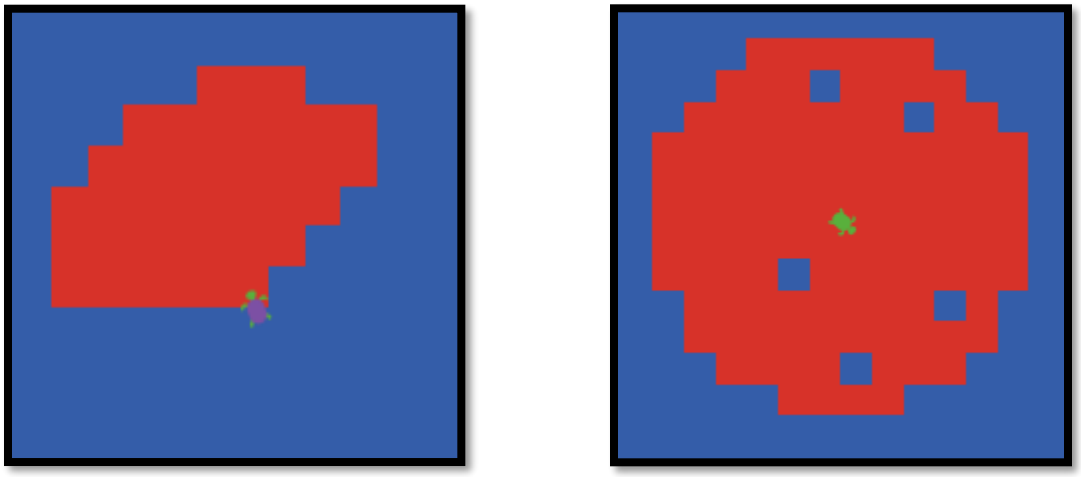


Figure 3: Front-facing vision (3 inputs) (left) and total vision (8 inputs) (right).

The included *network.py* file contains the network class and helper functions. There are two classes, one for a single-layer network and one for a network that contains a hidden layer. Bias is removed and gaussian initialization ($N(0, 2.5)$) is used for all network layers. For the multi-layer, a Tanh nonlinearity is applied to the first layer, and for both networks the SoftMax function is applied to the output tensor.

All networks share the following two helper functions: The *mutate* function takes a network copy and alters each weight with *mutation-rate* chance according to the normal distribution: $N(0, 5)$. The *get-action* function takes the *weights* agent variable and gives it as input to the agent's personal neural network cognition, returning a probability distribution over

which of the three possible actions the agent should take. In the NetLogo code, the maximum value from this distribution is taken as the action for the agent to take.

Agent cognition in this format is implemented into NetLogo through the *py* extension, as PyTorch is used as the standard library for implementing the neural networks.

Evaluation

The Fitness Metric

To evaluate agent performance, a fitness metric is compiled from agent interactions with the environment. The first component is the recent (the previous *fitness-plot-interval* times) average time turtles have gone without eating, or the *fitness-error*. The second and third components are the average uptime of food and hazard patches, respectively. Using the average time turtles have gone without eating is a good representation of how good turtles are at pathing towards food patches but may not necessarily be totally indicative of fitness as turtles will have to path around hazards as they are spawned into the world. Therefore, the average amount of time individual food and hazard patches have existed in the environment is considered as well. This last variation of the fitness formula is scaled by how many turtles there are in the world, as fitness could potentially be biased towards higher population counts with the current formulation.

The final metric can be computed as follows:

$$fitness = \frac{WA(HU) - 5 * WA(FU) - WA(FE)}{\sqrt{T}}$$

Where *WA*, a weighted average function, takes the input *x* as a list:

$$WA(x) = median\ x * 0.9 + mean\ x * 0.1$$

And where:

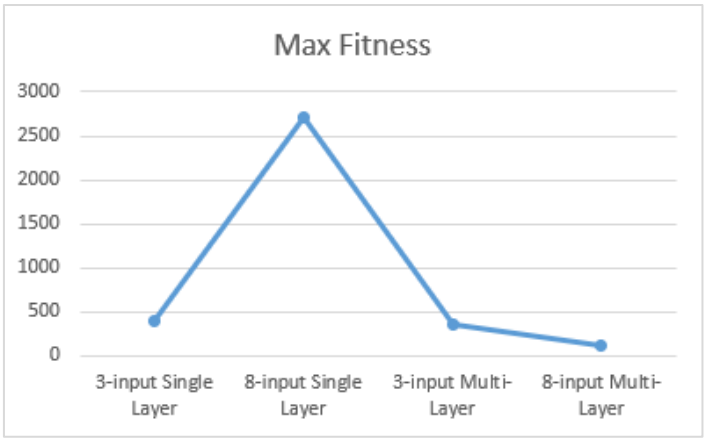
$$HU = Hazard\ uptimes;\ FU = Food\ uptimes;\ \ FE = Fitness\ error;\ \ T = Turtle\ count$$

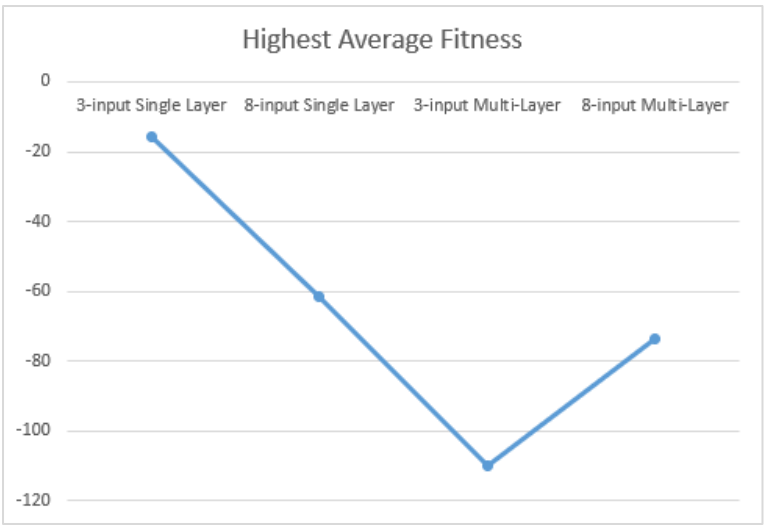
Experiments and Results

To compare the four different adaptive cognition structures, a series of simulations were run. For each network structure, 20 simulation runs were recorded, which amounted to 80 runs in total. To test the fitness of agents to changes in the environment and increasing selection pressures over time, the *food-level* and *hazard-level* parameter were altered over time. *Food-level* was set to decay from 100 to 20, and *hazard-level* to increase from 0 to 20 over 100,000 ticks. The *mutation-rate* was set to 5 for the 8-input multi-layer network, and 10 for all others. Other parameters are set as follows:

start-agents	*initial-energy-loss*	*move-interval*	*vision-angle*	*vision-distance*	*age-limit*
20	**0.01**	**1**	**120**	**6**	**2500**

Simulations are run with the NetLogo BehaviorSpace software, which allows measuring of custom metrics over a wide parameter space (Wilensky, 1999). Results of the experiments are gathered and plotted:





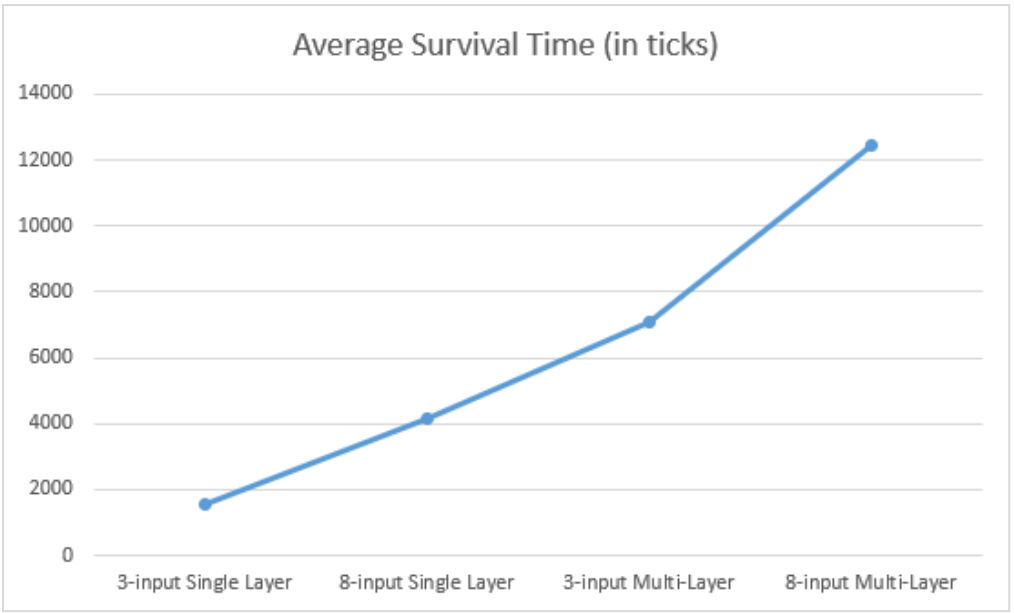


Figure 4: Fitness and survival values plotted for over 20 runs per each network.

Analysis and Discussion

Interpretation of results

The “Max Fitness” graph plots the max fitness achieved per network at any time over the 20 runs, while the “Average Max Fitness” graph takes the mean of the maximum fitness value per network in every one of the 20 runs. The “Highest Average Fitness” graph takes the highest average fitness per network out of the 20 runs, and “Average Survival Time” records the average time turtles stay alive per network over the 20 runs.

Focusing on peak fitness values, single-layer networks are able to generally become very optimized for fitness at some time in the environment, as fitness values up to 2705.77 are reached with the 8-input perceptron. The same trends are seen when considering the averages of multiple maximums (where a maximum is the highest fitness value in a run). But when we look past the peak fitness values at any given time, and instead look at the average fitness over entire runs, a trend arises where more complex networks tend to do worse, except for the most complex network puzzlingly.

Combining this information with the average survival time graph, however, gives a clearer picture as to the competency of each particular network in different areas of expertise. More complex networks eventually find better ways to navigate the environment as selection pressures mount but perform worse on the given “fitness” metric, as the metric favors easier environment spaces. Shifting the focus to less complex networks, these preform much worse in terms of adapting to changes in the environment, but much more easily take advantage of easier environments than the more complex networks.

This is to say that, at least in this particular competitive paradigm, there is a tradeoff between short-term learning speed and long-term population stability. If placed in the same environment, the agents with the simplest cognition would out-compete the agents with more sophisticated cognitive structure but would not have the network capacity to survive harsher environments where, paradoxically, the more complex agents would.

Emergent Behavior

Emergent behavior is observed in certain circumstances. Given agent genes (network weights) become homogenous enough, agents can seem to be following "leaders" or creating pathways that other agents follow *even though they actually have no perception of other agents* at all.

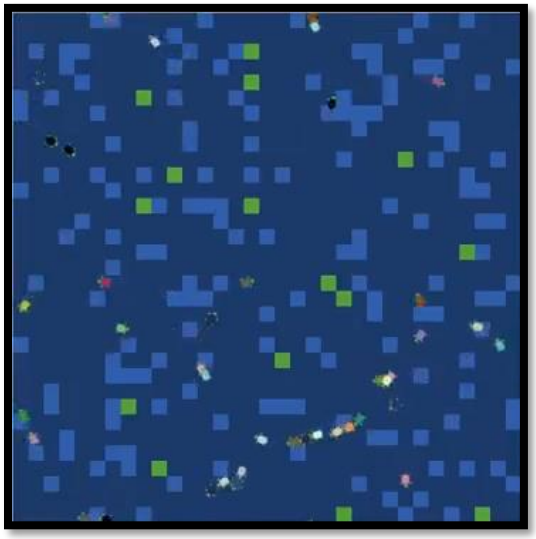


Figure 4: Turtle agents "following" each other (pictured at bottom).

Through evolution of population weights distribution over time, emergent behavior is most evident in the strategies turtles exhibit when obtaining food while avoiding hazards. Agents with multi-layer networks often times adapt behaviors of turning intricately when hazards are plentiful, and agents with total (8-input) perception exhibit behavior of backtracking for food in the same area multiple times. These behaviors, if observed in a vacuum without prior knowledge of agent rules, can easily be rationalized by assuming that turtles almost have a "human" movement to them, when in-fact these behaviors are emergent from population drift given a very simple adaptive cognition.

Conclusions

In this paper I've attempted to answer some pressing questions about very simple adaptive cognition in the context of evolving agent populations. The concept of brittle cognition that has a high short-term adaption speed contrasted with a cognitive structure robust to complex environment change should be an important note for those trying to harness the power of adaptive agent cognition. This is to stress that tradeoffs between agents that can "get off the ground fast" and agents that are "future-proof" present a barrier to successful adaptive cognition that can be cleared given one is adequately equipped with knowledge of the suitable cognitive structure to implement.

For example, future work would entail not only a neuroevolution of network weights, but a *structural* neuroevolution where the cognition structure itself can be adaptive. Given that the bane of learning/adapting is immutable frameworks and given the static nature of the neural

network structures used here, the next steps may be to allow the learning process to take advantage of mutating network capacity also.

References

[1] Wilensky, U., Rand, W. An Introduction to Agent-Based Modeling: Modeling Natural, Social, and Engineered Complex Systems with NetLogo. MIT Press (2015)

[2] Russell, Stuart J., Peter Norvig, and Ernest Davis. Artificial Intelligence: A Modern Approach. 1st ed. Upper Saddle River, NJ: Prentice Hall. (1995)

[3] Darwin, Charles, 1809-1882. On the Origin of Species by Means of Natural Selection, or Preservation of Favoured Races in the Struggle for Life. London :John Murray. (1859)

[4] J. Koza. Genetic Programming: On the Programming of Computers by Means of Natural Selection. MIT Press, Cambridge, MA, USA, (1992)

[5] Ronald, Edmund, Schoenauer, March. "Genetic Lander: An experiment in accurate neuro-genetic control", Parallel Programming Solving from Nature, pp. 452–461. (1994)

[6] Stonedahl, Forrest & Stonedahl, Susa & Cheboi, Nelly & Tazyeen, Danya & Devore, David. Novelty and Objective-based Neuroevolution of a Physical Robot Swarm. 382-389. (2017)

[7] B. Head, A. Hjorth, C. Brady and U. Wilensky, "Evolving agent cognition with Netlogo LevelSpace," 2015 Winter Simulation Conference (WSC), pp. 3122-3123 (2015)

[8] Head, B. and Wilensky, U. Agent Cognition Through Micro-simulations: Adaptive and Tunable Intelligence with NetLogo LevelSpace: Proceedings of the Ninth International Conference on Complex Systems, pages 71-81. (2018)

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