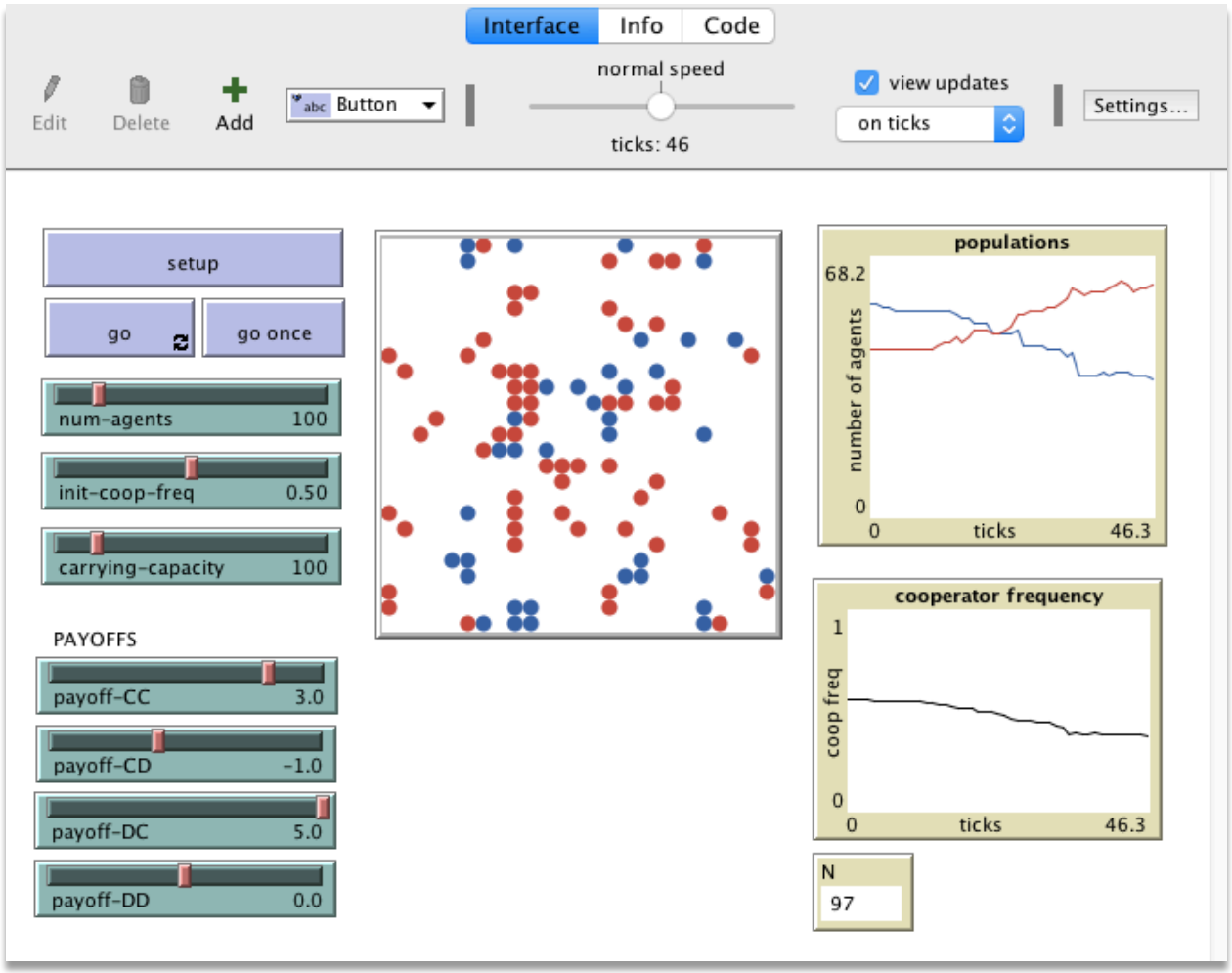
Computational Modeling of Social Behavior

Coding Challenges – Day 3

You will be exploring a model in which agents who move on a 2-D space interact with neighbors for prisoner's dilemma game payoffs. Payoffs accumulate, and can be used to produce offspring. Lifespans are limited by the environment's carrying capacity. In other words, this is a simple evolutionary model. This one is more complicated than the previous ones we've seen in this course, and so the NetLogo code for the baseline model is provided. The model is still described below, in order to help you map between concepts and code. Explore this model. Try to get a sense for how it works, both by inspection of the visualization and by reading through the code. When you are ready attempt to add some of the extensions described later in this document. Note how the model dynamics respond to these changes.



Designing the model

Agent and their world

The world is a toroidal square grid with dimensions 101 x 101 patches.
Agents are turtles that play the prisoner's dilemma (PD) game with neighbors, and attempt movement when unable to find a co-player. Each game encounter represents an opportunity to gain resources in the form of energy units. If an agent gains enough energy, it attempts to reproduce. If its energy falls to zero, it dies. All individuals play pure strategies of cooperate (C) or defect (D), and produce offspring of the same type. Cooperators are visualized as blue, defectors as red.

Initialization

- First, all variables and agentsets are cleared.
- Next, any variables that need to be initialized, are.
- Next, agents are initialized. An agent is a cooperator with probability *init-coop-freq*, otherwise it is a defector. Its color is assigned correspondingly. Each agent is placed on a patch by itself. Its initial energy is randomly drawn from a uniform distribution between 1 and 51.
- Finally, we reset the tick counter.

At each tick

- If all agents are the same type (all C or all D), or if all agents have died, stop the simulation.
- Set all agents' *played?* variable to false. This will ensure that each agent plays the PD game at most once per tick.
- For each agent:
    - If energy is below zero, die.
    - If the agent hasn't played this tick, attempt to **interact**:
        - The agent examines all turtles in its Moore neighborhood (the 8 closest patches) and finds one that hasn't played yet this tick, if one exists. If successful, both agents' *played?* variables are set to true. They play the PD game receive payoffs according to their respective strategies. The payoffs are added to each agent's energy.
    - If the agent's energy is above *reproduce-threshold*, attempt to **reproduce**.
        - The agent checks to see if there is a patch in its Moore neighborhood (8 closest neighboring patches) that is empty (does not have an agent on it). If so, the agent "hatches" a new agent onto that patch. The new agent inherits the strategy of its parent. The cost of reproduction, *reproduce-cost* is deducted from the focal agent's energy, and the newly hatched offspring's energy is set to *reproduce-cost*.
    - The agent checks to see if the current population is above the environmental carrying capacity, *carrying-capacity*. If so, a random turtle has its energy reduced by 10, reflecting the costs of overcrowding.
    - If the agent could not find a co-player among its neighbors, it attempts to **move**.
        - The agent chooses a patch from among its Moore neighborhood. If that patch is not occupied, it moves there. Otherwise, it stays where it is.

Plotting

Two plots show somewhat redundant information. The first plot shows the absolute population sizes of the two strategy types as a function of time. The second plot shows the fraction of the population who are cooperators. A monitor shows the instantaneous total population size.

CHALLENGES:

The most important thing is to explore this model and make sure you understand how it works, and how the code translates into agent behavior. While you are doing this, think about questions you might have about the population dynamics in the model, and what you could potentially learn about cooperation as defined by PD interactions. How would you need to modify the model to explore some of those questions. You can do whatever you like here, but you might try some of the following suggestions. For each one, make sure you can use sliders, choosers, and/or switches to recover the original model as well as to vary relevant parameters.

Dispersal

In the original model, offspring were initialized very close to their parent. This facilitates a type of positive assortment, since offspring have the same phenotype as their parent. In some human cultures and many non-human species, at least some offspring disperse far from their parents before they are themselves of reproductive age. Create a switch that causes offspring to be placed on a random patch on the grid. Make sure this patch is unoccupied. A more advanced challenge is to allow a slider to determine the maximum distance an offspring can be placed from its parent.

Mutation

In the original model, offspring always inherited the same phenotype as their parent. Add a variable that corresponds to the probability that an offspring is a "mutant," inheriting a different strategy than its parent.

Movement strategies

In the original model, a moving agent simply chooses one of its 8 neighboring patches at random to move to. However, how an agent moves can matter. A strategy that mostly moves forward will tend to explore the space widely, while a strategy that mostly turns around to move will tend to stay close to where it starts. Create sliders or choosers that control how agents move. For added complexity, make separate choosers or sliders for cooperators and defectors, so that each game strategy can correspond to a different movement strategy. Might some movement strategies favor defectors more than cooperators, and vice versa?

Occupation and embodiment

In the original model, only one agent can occupy a patch at a time. However, although individuals cannot in reality occupy the exact same space, a patch could also represent a space that can accommodate multiple individuals. Create a switch that allows multiple agents to occupy a single patch. You may want to change how agents find co-players so that they can search on their own patch.

New strategies: Walkaway

In the original model, a agent only moves if it cannot find a co-player. This allows a cooperator to be severely exploited by a defector, or mutual defection to last a long time. Add a strategy variant called *walkaway* that moves if its co-player's most recent game play was to defect.

New strategies: Probabilistic strategies

In the original model, all agents play pure strategies of always cooperate or always defect. However, strategies may often be better characterized as probabilistic—what someone *usually* does. Allow agents to be defined not by a discrete strategy, but by a probability of cooperating. You may want to visualize its color in a spectrum between red and blue. Your plots may be better served by plotting the average cooperation probability, and/or the distribution of phenotypes.