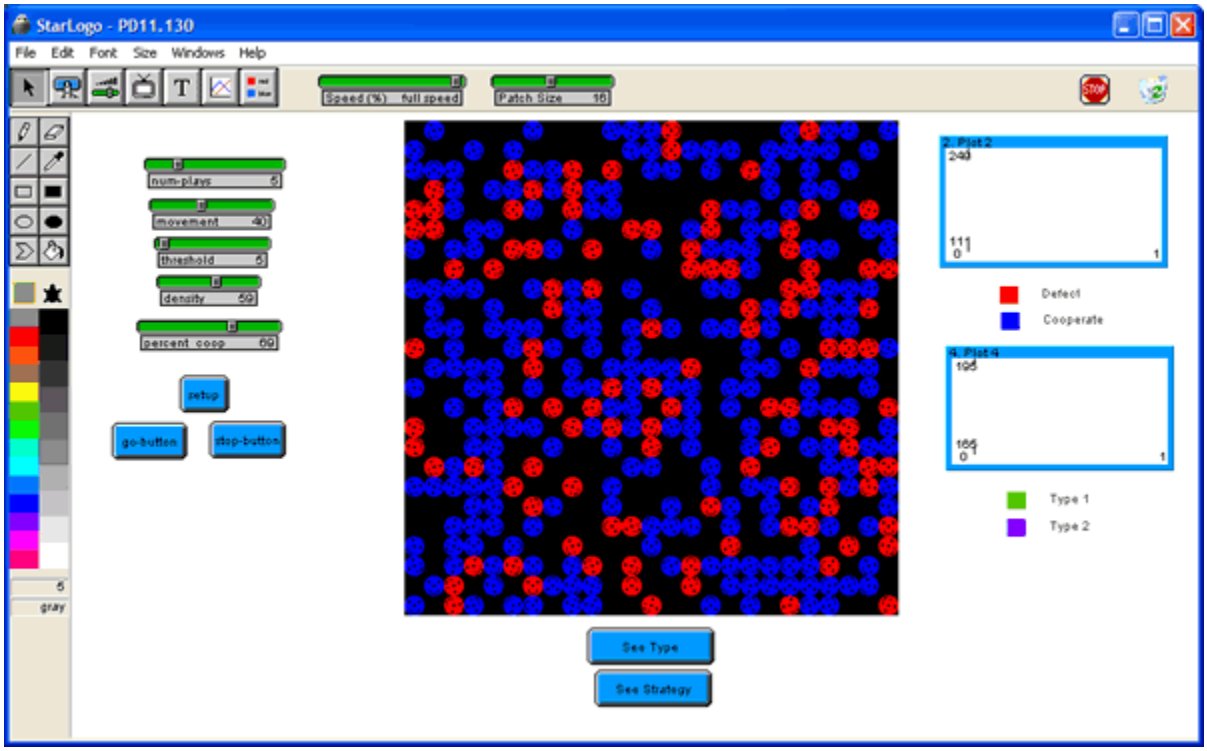
StarLogo - Competing Prisoners Competition

Overview

Think back to the Prisoner's Dilemma participatory simulation that we did last week. What kinds of strategies did you use and under what conditions? Did other people use more successful strategies. The goal of this project is for you to program strategies that can outcompete your opponents.

StarLogo Project

Load the PD11.124 project into StarLogo. It should look like the picture below.



There are several pieces of the user interface that you can interact with. The sliders control several variables as follows:

- **num-plays** = number of times each agent plays with another agent before moving on
- **movement** = tenths of steps that each agent takes each move
- **threshold** = the number of points that each agent must have before reproducing
- **density** = initial density of agents (only one agent per spot)
- **percent_coop** = initial percentage of population that cooperates.

There are several buttons that control the simulation as follows:

- **setup** = begin a new round
- **go-button** = starts simulation
- **stop-button** = stops simulation at end of next round
- **see type** = shows which agents are type 1 (green) and which are type 2 (purple)
- **see strategy** = shows which agents are currently cooperating (blue) and defecting (red)

Additionally the two graphs on the right hand side show the current numbers of cooperators and defectors (top graph) or type 1 and type 2 individuals (bottom graph).

Creating Strategies

Your goal is to create a strategy that will outcompete your opponents over a variety of conditions (varying movement rates, neighborhoods, and numbers of meetings). By default all of the agents play the same strategy for life, and give birth to agents that also play that strategy. The rules of the game are as follows:

- Payoffs

	Cooperate	Defect
Cooperate	1	-2
Defect	3	-1

- Initial Energy = 5
- Birth Cost = 3

You must modify the decide procedures (you will be using one and your opponent will be using the other) to outcompete your opponents. To do so you may use the following commands:

- **if *condition* [*result*]**
  - does *result* if *condition* is true
  - e.g. if times-played > 0 [setstrategy coop]
    - cooperates if you just played with this agent
- **ifelse *condition* [*result1*] [*result2*]**
  - does *result1* if *condition* is true, otherwise *result2*
  - e.g. ifelse times-played > 0 [setstrategy coop] [setstrategy defect]
    - plays cooperate if you just played, otherwise plays defect
- **set*variable value***
  - makes the value of the *variable value*
  - e.g. setstrategy coop
    - makes the current strategy cooperat
- **last *listname***
  - last item in *listname*
  - e.g. last my-play
    - last strategy you played
- **item *number listname***
  - item *number* from *listname*

    - e.g. item 5 op-play
        - the fifth strategy that was played against you
- **length *listname***
    - returns the current number of items in the list *listname*
- **repeat *number [instructions]***
    - repeats the *instructions number* of times
- **random *number***
    - returns a random number between 0 and number
        - e.g. if (random 100) < 75 [setstrategy coop]
        - sets strategy to cooperate 75% of the time
- **let [*:variable value*]**
    - creates a local variable called *:variable* and sets it to *value*
- **:my-opponent**
    - id number of your opponent (colon is important)
- **score**
    - current score
- **score-of :my-opponent**
    - score of your opponent
- **result**
    - result of last interaction
- **last-patch**
    - color of last patch
- **times-played**
    - number of times played current opponent in a row
- **plays**
    - total number of plays
- **last-time :my-opponent**
    - strategy your opponent played against you last (-1 if never played)
- **my-play**
    - list of your plays
- **op-play**
    - list of plays against you
- **op-list**
    - list of opponents
- **result-list**
    - list of results

      You are not allowed to directly look at or modify your opponents values. You are also not allowed to modify any of the other procedures in the program, or falsely modify your own values.

- This might be useful if you want to know if your opponent thinks they've played you before (sometimes because of the way id number sare recycled you might think you've played them, but you haven't). Paste this at the bottom of the window and then use the command compute-op-plays :my-opponent to return the number of times that they played you.
    - to compute-op-plays :my-opponent
      let [:n (length (op-list-of :my-opponent))]
      let [:last-time :n + 1]
      repeat :n
      [
      if who = (item :n (op-list-of :my-opponent))
      [set:last-time :n set :n (:n - 1)]

```
]
output ((length op-list-of (:my-opponent)) - :last-time + 1)
end
```

- There was an error in the original program, a line was missing
  - to meet
  - ....
  - set times-played 0
  - set times-played-of-partner 0 ;; this line was missing