Simulating Language: Lab 3 Worksheet

Download the file `signalling2.py` from WebCT and save it your usual working directory. To open this file in IDLE, choose **File-Open** from the menu, and find the file you just saved. Opening the signalling2.py file in IDLE gives you access to the rather handy syntax highlighting and automatic indentation that IDLE provides, and is better than editing the code in something like notepad.

This worksheet extends the first signalling program by measuring communicative accuracy among a population of agents. Most of the code is the same as the first, with the exception of the `pop_update` function; we also store more information about each of the agents, as described in the comments below.

Make sure you understand the complex embedded list structure described:

- ✤ a population is a list of agents;
    - ✤ an agent is a list containing three items: a production system, a reception system, and a set of scores;
        - ✤ a production system is a matrix of association weights (i.e. a list of lists);
        - ✤ a reception system is a matrix of association weights (i.e. a list of lists);
        - ✤ the set of scores is a list of four integers.

```
"""
Simple innate signalling simulation - communication in a population

pop_update takes a list of agents and picks two at random to be
producer and receiver for a random meaning. Each agent consists of
a production system, a reception system and a list of 4 scores: the number of
times they have successfully been understood, the number of times they have
spoken, the number of times they have successfully understood an interaction,
and the number of times they have heard an interaction, respectively.

Usage example:

population = [[[[3, 1], [0, 2]], [[1,0], [2,4]], [0, 0, 0, 0]],
              [[[1, 0], [0, 1]], [[2,0], [0,1]], [0, 0, 0, 0]],
              [[[0, 1], [1, 0]], [[0,1], [1,0]], [0, 0, 0, 0]]]

for i in range(10000): pop_update(population)

print population

will pick one of these three agents to be speaker and another to be
hearer, and update scores accordingly, for 10000 interactions.

NOTE: there are two returns after the for loop in this usage example:
otherwise, Python will wait for you to add more code to the body of the for
loop.
"""
```

How would you access the production matrix for the first agent in the population? How about the set of scores for the last agent in the population?

```
import random
import matplotlib.pyplot as plt

def wta(items):
    maxweight = max(items)
    candidates = []
    for i in range(len(items)):
        if items[i] == maxweight:
            candidates.append(i)
    return random.choice(candidates)

def communicate(speaker_system, hearer_system, meaning):
    speaker_signal = wta(speaker_system[meaning])
    hearer_meaning = wta(hearer_system[speaker_signal])
    if meaning == hearer_meaning:
        return 1
    else:
        return 0

# ----- new code below -----

def pop_update(population):
    speaker_index = random.randrange(len(population))
    hearer_index = random.randrange(len(population) - 1)
    if hearer_index >= speaker_index: hearer_index += 1     # ensure speaker
                                                        # and hearer are different
    speaker = population[speaker_index]
    hearer = population[hearer_index]
    meaning = random.randrange(len(speaker[0]))
    success = communicate(speaker[0], hearer[1], meaning)
    speaker[2][0] += success
    speaker[2][1] += 1
    hearer[2][2] += success
    hearer[2][3] += 1
```

Have a look `pop_update` and check you understand how it works.

How does the program ensure that the same agent does not play both roles?
What scores are updated after a communication event, and why?

Work through the following questions: these go from easy to hard, everyone should answer 1 and 2, 3-5 are optional and can involve as much or as little coding as you like.

1. The two ways of scoring an agent's success depend on being understood (the first number), and understanding (the third number). What are the ecological interpretations of these scores? Which do you think are evolutionarily significant, and why?

2. Can you construct a population where every agent gets approximately the same score for being understood, but different scores for understanding? What about the other way round?

3. How would you adjust this code to keep a trial-by-trial record of the communicative accuracy of the population by trial number? Hint: look at how this was achieved in signalling1.py.

4. Who communicates with who in a population? What other ways could you model this, and how would you start adjusting the code to implement your model? Hint: what if people only talked to people who were 'near' them?

5. Rather than explicitly providing a population to evaluate, can you come up with some code that generates a population of a specified size with random production and reception matrices?