**General Information:**
- This document was created for use in the "Bridges to Computing" project of Brooklyn College.
- You are invited and encouraged to use this presentation to promote computer science education in the U.S. and around the world.
- For more information about the Bridges Program, please visit our website at: [http://bridges.brooklyn.cuny.edu/](http://bridges.brooklyn.cuny.edu/)

**Disclaimers:**
- **IMAGES:** All images in this presentation were created by our Bridges to Computing staff or were found online through open access media sites and are used under the Creative Commons Attribution-Share Alike 3.0 License. If you believe an image in this presentation is in fact copyrighted material, never intended for creative commons use, please contact us at [http://bridges.brooklyn.cuny.edu/](http://bridges.brooklyn.cuny.edu/) so that we can remove it from this presentation.
- **LINKS:** This document may include links to sites and documents outside of the "Bridges to Computing" domain. The Bridges Program cannot be held responsible for the content of 3rd party sources and sites.
Agent Based Programming & Simulations

Lecture 2: Multi-Agent Systems
References


- Stan Franklin and Art Graesser (1996); Is it an Agent, or just a Program?: A Taxonomy for Autonomous Agents; Proceedings of the Third International Workshop on Agent Theories, Architectures, and Languages, Springer-Verlag, 1996

Content

- Types of Agents
  - Agent Defined
  - Agent Types
  - Intelligent Agents

- Multi-Agent Systems
  - Defined
  - Properties
  - Research Areas

- Continuing with NetLogo
Agent Defined

- An Agent "is an autonomous entity which observes and acts upon an environment and directs its activity towards achieving goals." Russell & Norvig 2003,

- NOTE: A "software agent" usually implies a program, a "computer agent" can be a hardware/software combination, and an "intelligent agent", can be anything (including people/groups of people).
Components of an Agent

1. **Sensors**
2. **Actuators/Effectors**
3. **Reasoning** (maybe be very simple "if-then")
4. **Learning** (optional, defines an "intelligent agent")
Basic Agent Types

- **Simple reflex agents** -> If condition then action.
  - Can accomplish simple tasks quickly, unlikely to find "optimal" solution
  - If (distance-to-wall < 5 ) { turn right } else { turn left }
  - *NOTE: Algorithm above will eventually find door out of room.*

- **Model-based reflex agents** -> A model-based reflex agent keeps track of the current state of the world using an internal model (create a model). It then chooses an action based on tests performed on the model.
  - Slow
  - (1) Make map of room. (2) Calculate path(s) to door (3) Execute path
  - Can find optimal solution in some cases.
Intelligent Agents

- **Goal-based agents**
  - Goal-based agents are model-based agents which store information regarding situations that are desirable. This allows the agent a way to choose among multiple possibilities, selecting the one which reaches a goal state.

- **Learning agents**
  - Learning has an advantage that it allows the agents to initially operate in unknown environments and to become more competent than its initial knowledge alone might allow.
  - Learning is difficult, time-consuming and has to be done within some other methodology (neural-networks, reinforcement learning, etc.)
"A multi-agent system is one that consists of a number of agents which interact with one another. In the most general case, agents will be acting on behalf of users with different goals and motivations. To interact successfully, they will require the ability to cooperate, coordinate and negotiate with each other, much as people do." (Wooldridge)

A Multi-Agent system is... an environment in which many (well, two or more) agents exist and interact.
Properties of multi-agent systems

1. Individual agents are self-interested
   - Agent pursue their own individual goals
   - There may be team rewards for a group of agents achieving a goal together... but

2. Cooperation is not governed,
   - Cooperation (and altruism) are emergent (if happens at all)

Note: Compare this to "distributed systems", where

1. goals are only group-based
2. cooperation is engineered to be inherent in the system
Multi-Agent Systems… Why?

Are there advantages to using Agent-Based programming? Yes!

1. **Complexity**: Some problems are so complex that they are basically impossible to model as a single program.
   - Examples: Weather, Internet, Travel Systems

2. **Emergent Behavior**: Some types of behavior are very difficult to program in directly but can result as a side effect of individual agents selfishly pursuing their own very simple goals.
The Flocking Program - Emergent Behavior (link)
Research Areas – Multi-Agent Systems

Key Research Areas

1. **Agent design (micro level):** How do we design individual agents to operate (accomplish their own goals) effectively?

2. **Society Design (macro level):** How do we design *groups* of agents, or *societies*, to operate effectively together?

3. **Hive Intelligence (abstract level):** Can simple agents, working together, effectively demonstrate high level reasoning abilities? Can simple agents, working together, accomplish very complex tasks?
Ants & Termites (Hive Intelligence)
continuing with NetLogo
NetLogo Imperative Programming

1. **Sequence** -
   1. Left-Right, Top Down
   2. Buttons can also call procedures.

2. **Selection** -
   1. if condition [ commands ]
   2. ifelse condition [ commands1 ] [ commands2 ]

*Example:*

```logo
ask patches [
  ifelse pxcor > 0
  [ set pcolor blue ]
  [ set pcolor red ]
]
```
NetLogo Imperative Programming (cont)

3. **Repetition** -\( \rightarrow \) (link1)
   
   1. \( \text{while} \left[ \text{reporter} \right] \left[ \text{commands} \right] \)

     Example:
     
     \[
     \text{ask turtles [}
     \text{while} \left[ \text{xcor} < 5 \right] \left[ \right.
     \text{right random 360}
     \text{forward 1}
     \text{]
     \right]
     \]
     
   2. Forever buttons -\( \rightarrow \) The advantage of a forever button is the user can click the button to stop the loop.
NetLogo Procedural Programming

- A procedure (or function) is created using the reserved keyword 'to'.
- The name of the function follows 'to'.
- The end of a procedure is marked by the 'end' keyword.
- Procedures can be called by other procedures or by buttons on the interface.
NetLogo Agents as Objects

- In the previous lab we introduced the "primitives" for the turtle and patch agents.
- Some of those primitives were member variables (facts about the agents) and other primitives were functions (procedures).
- The agents in NetLogo are also objects.
- Unlike regular objects agents in NetLogo can "see" their environment and each other.
NetLogo **Sensors** and **Effectors**

- Agents in Netlogo can use special primitives (built in functions) to "see" their environment.
  - turtles-here (returns a list of turtles nearby)
  - patches-here (returns a list of patches nearby)
  - turtles-at (returns a list of turtles at a specific location)
  - patches-at (returns a list of patches at a specific location)
  - self (returns local state information)

- Agents can also use special primitives (built in functions) to "affect" their environment, including other agents (**ask**).
  - setxy (change a turtles location)
  - set color (change a turtles color)
  - set pcolor (change a patches color)
  - hatch (create NEW turtles)
  - die (deletes a turtle)
Groups, Sets, Lists

- When you use the command `ask turtles []` we are actually sending command to a set (group, or list) consisting of all the turtles in the program.

- In NetLogo we can create what are called AgentSets literally sets (groups, lists) of agents.

- We can then send commands to the agents in that set.

Examples: `(with) (turtles-here) (any?) (in-radius)`

- `ask patches with [pcolor = red]`
- `ask patch 0 0 [ ask turtles-here`
- `if any? turtles with [color = red]`
- `ask turtles [ ask patches in-radius 3]`
Breeds

- Breeds can be thought of as a customizable agentset.

  ```
  ask dogs [ forward 1 ]
  ```

- Officially a breed is like a “subtype” of turtle:
  - it has all the attributes of turtle,
  - plus any new attributes declared for the breed.

- Breeds can be declared and customized in your code

  ```
  breed [ dogs dog]
  dogs-own [ flea-count ]
  ```

- Breed is actually an attribute, which can be reset.

  ```
  show [ breed ] of turtle 0
  ask turtle 0 [ set breed dogs]
  ```
General Program Outline

globals [ ... ] ;; global variables (also defined with sliders, ...)
turtles-own [ ... ] ;; user-defined turtle variables (also <breeds>-own)
patches-own [ ... ] ;; user-defined patch variables

to setup
  clear-all
  ;; setup-patches
  ;; setup-turtles
  ;; setup-plots
end

to go
  ;; update-turtles
  ;; update-patches
  update-plots
  tick
end

;;;

to update-plots
  set-current-plot "my-plot"
  set-current-plot-pen "my-pen"
  plot some-number
end
You will have several labs on NetLogo.

Lost of help is available online:

- Fantastic (short) NetLogo Quick Guide
- Full Netlogo Manual
- NetLogo Dictionary
The End