Agent Based Systems

2

Modeling Complex Systems with NetLogo

Dr Budditha Hettige

Contents

Agent based Simulation Using Netlogo

Agent-based system simulation is a powerful approach that allows researchers to model and analyze complex systems by simulating the behavior and interactions of individual agents within the system. NetLogo is a widely used programming language and integrated development environment (IDE) specifically designed for building agent-based simulations. It provides a user-friendly interface and a range of built-in features that make it accessible to both novice and experienced users. NetLogo is widely used in various fields, including social sciences, ecology, economics, and computer science. It enables researchers to explore complex systems, study the effects of different variables and interventions, and gain insights into the behavior and dynamics of agent-based systems. By leveraging the capabilities of NetLogo, researchers can gain a deeper understanding of complex systems and make informed decisions based on the outcomes of agent-based simulations.

Further, Agent-based modeling is a bottom-up modeling approach in which individual agents are defined and programmed to interact with their environment and other agents. Agents can possess various attributes, behaviors, and decision-making rules, which collectively drive the dynamics of the system being modeled.

Steps to Install NetLogo

To install NetLogo please use the following steps;

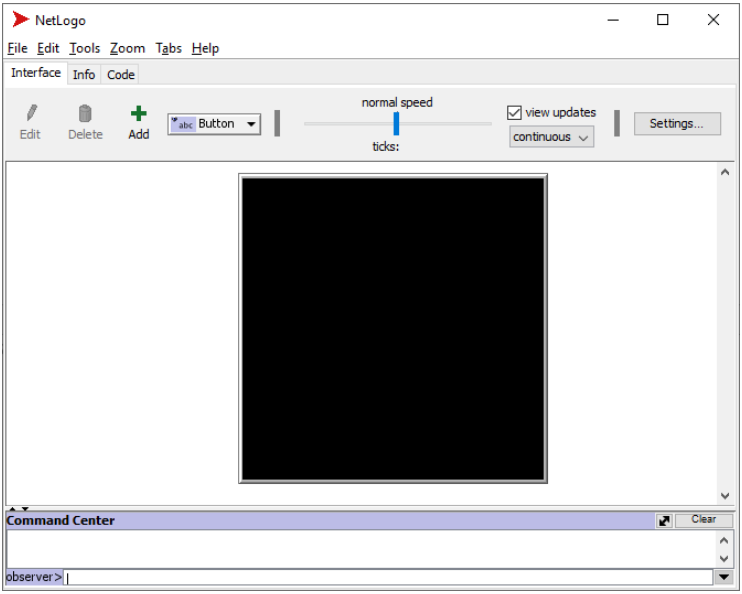
1. Visit the NetLogo website: Go to the official NetLogo website at https://ccl.northwestern.edu/netlogo/.
2. Download the installer: Click on the "Download" link on the website's navigation menu. It will lead you to the download page.

3. Choose your operating system: On the download page, you will see options for different operating systems (e.g., Windows, macOS, and Linux). Select the appropriate version for your system.
4. Download the installer: Click on the link to download the installer file. The file size is relatively small, so the download should complete quickly.
5. Install NetLogo: Once the installer has finished downloading, run it to start the installation process. Follow the on-screen instructions to install NetLogo on your computer. The steps may vary slightly depending on your operating system.
6. Launch NetLogo: After installation, you can launch NetLogo by double-clicking on its icon (for Windows and macOS) or by running it from the Applications folder (for macOS).

NetLogo Interface

The NetLogo interface is designed to provide a user-friendly environment for creating, running, and visualizing agent-based models. It consists of various components that allow users to interact with the model, observe simulation results, and modify the model's behavior

NetLogo GUI



Netlogo User interface

Command Center: The Command Center is the primary area where you can interact with the model using NetLogo's Logo-based programming language. It allows you to enter and execute commands, setup parameters, and control the simulation.

Info Tab: The Info tab provides information about the current model, including its name, author, description, and version. It may also contain additional notes or instructions provided by the model creator.

Code Tab: The Code tab allows you to view and edit the model's source code. The code is written in NetLogo's Logo language, which is a variant of Lisp. Here, you can modify the model's behavior, create custom functions, and add new features.

Interface Tab: The Interface tab provides controls for adjusting model parameters and settings. You can create sliders, buttons, switches, and input boxes to manipulate the model's variables interactively. These interface elements allow you to change the model's initial conditions, run experiments, and observe the effects in real-time.

Graphs and Plots: NetLogo allows you to create graphs and plots to visualize simulation results dynamically. The interface provides options to display agent-related statistics, system-level measures, and other variables over time.

Info and Procedures Tabs: These tabs are hidden by default but can be expanded as needed. The Info tab displays additional information about agents and patches in the model, while the Procedures tab shows the model's built-in procedures, functions, and user-defined code.

Turtle Monitor and Patch Monitor: These are specialized windows that display real-time information about specific agents (turtles) or patches in the simulation. You can use these monitors to observe agent attributes, patches' states, or any other relevant data during the simulation.

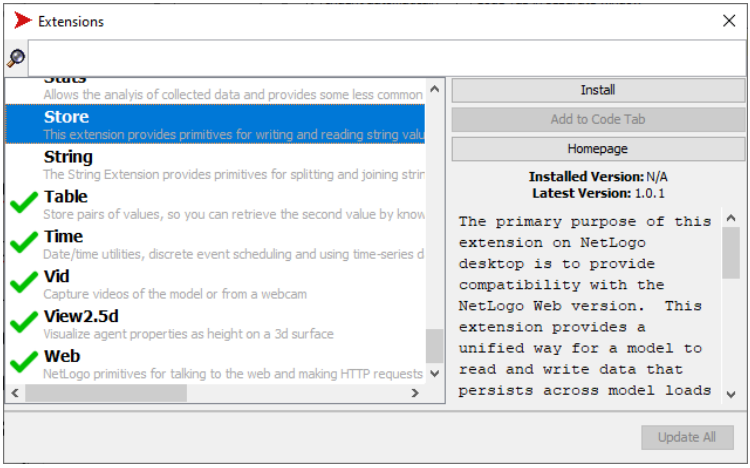
Model Control Buttons: The interface includes various buttons that allow you to control the simulation, such as starting and stopping the simulation, resetting the model, and stepping through the simulation one iteration at a time.

Model Preview: The preview area displays the model itself. It shows agents (turtles) represented as individual icons and their interactions with the environment (patches). The model preview updates in real-time as the simulation runs.

NetLogo, Extensions

NetLogo, extensions are external libraries that provide additional functionalities to the modeling environment. They allow users to extend the capabilities of NetLogo by adding new features, data formats, and algorithms. Extensions are typically written in other programming languages (e.g., Java, Scala, Python) and are integrated into NetLogo through the NetLogo Extension API.

NetLogo extensions are useful when the built-in features of NetLogo are not sufficient for a specific modeling task or when you want to use specialized tools or algorithms available in external libraries.



Installing Extensions: To use an extension, you need to install it first. NetLogo provides a set of built-in extensions that can be installed through the Extension Manager. Additionally, third-party extensions can be downloaded from various sources and then installed using the "Extensions" menu in the NetLogo interface.

Importing Extensions: Once an extension is installed, you need to import it into your NetLogo model. This is typically done using the extensions keyword in your NetLogo code. For example, to use the "GIS" extension for geospatial data, you would write extensions [gis].

Accessing Extension Features: After importing an extension, you can access its functions, procedures, and data types in your model code. Each extension has its own set of commands and reporters that can be used to interact with the extension's functionality.

Using Extension Primitives: Extensions often provide new primitives (commands and reporters) that can be used in your NetLogo model. These primitives allow you to perform specific tasks or calculations provided by the extension.

Extension Documentation: Each extension comes with its documentation, which explains the available commands, reporters, and usage examples. You can refer to the documentation to understand how to use the extension effectively.

Third-Party Extensions: Besides the built-in extensions, there are many third-party extensions developed by the NetLogo community. These extensions cover a wide range of topics, such as advanced statistics, network analysis, optimization algorithms, and more. You can find third-party extensions on the NetLogo website or other community platforms.

Some popular built-in extensions in NetLogo include "Goo," "Sound," "Table," and "Matrix." Third-party extensions can greatly expand the modeling capabilities of NetLogo and provide solutions for specific domains or research areas.

Turtle Shape Editor

The "Turtle Shape Editor" allows you to create custom shapes for the turtles, giving you more flexibility in visually representing different agents or objects in your simulation.

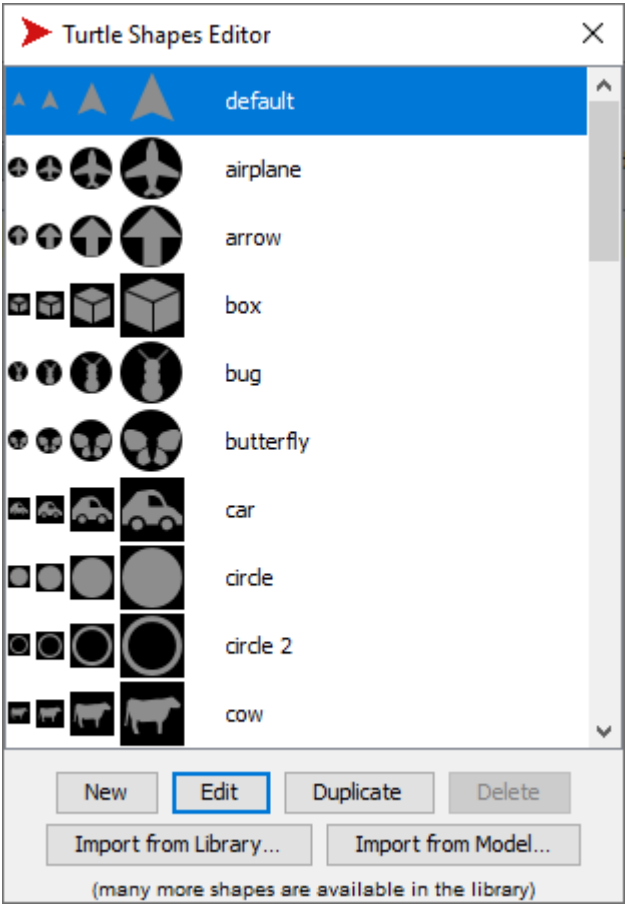
Open the Turtle Shape Editor: To access the Turtle Shape Editor, go to the NetLogo interface and click on the "Turtles" tab. From there, you should see a button labeled "Turtle Shapes." Click on this button to open the Turtle Shape Editor window.

Create or Modify Shapes: In the Turtle Shape Editor, you can draw custom shapes using simple drawing tools like lines, rectangles, circles, and polygons. You can also import image files to use as turtle shapes.

Define Different Turtle States: NetLogo allows you to define multiple turtle shapes corresponding to different states or appearances. For example, you might want turtles to have different shapes when they are active, inactive, or represent different agent types.

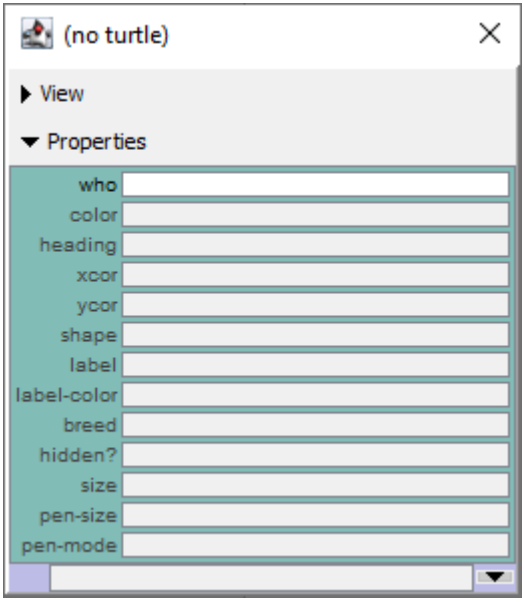
Assign Shapes to Turtles: Once you have created or modified turtle shapes, you can assign these shapes to individual turtles in your model. This can be done using the NetLogo programming language or the Interface tab in the model interface.

Run the Simulation: After setting up the turtle shapes, you can run your simulation to see the customized turtle appearances in action.



Turtle Monitor

In NetLogo, a "Turtle Monitor" is a specialized window that displays real-time information about specific turtles (agents) in your simulation. It allows you to observe and analyze individual turtle attributes, such as their position, heading, color, or any other custom variables you define for them.



Setup: Before using a Turtle Monitor, you need to make sure you have defined and created the turtles in your model. This can be done in the "Setup" procedure or using the Interface tab to set the initial number of turtles and their attributes. The Turtle Monitor is a helpful tool for debugging and gaining insights into the behavior of individual turtles in your agent-based models. It allows you to observe how specific turtles interact with their environment, change their states, and respond to changes in the model's conditions.

Opening a Turtle Monitor: To create a Turtle Monitor, go to the NetLogo interface and click on the "Turtles" tab. From there, you should see a button labeled "Turtle Monitor." Click on this button to open the Turtle Monitor window.

Selecting Turtles: In the Turtle Monitor, you can select the turtles whose attributes you want to observe. You can choose to observe all turtles or a subset of them based on certain conditions.

Observing Attributes: Once you have selected the turtles, the Turtle Monitor will display their attributes in real-time as the simulation runs. You can choose which attributes to display and customize the appearance of the monitor.

Updating the Monitor: As the simulation progresses, the Turtle Monitor will update automatically, showing the latest values of the selected turtle attributes at each iteration.

Customizing the Monitor: You can adjust the layout, font size, and other settings of the Turtle Monitor to suit your preferences and analysis needs.

NetLogo Environment

NetLogo provides a graphical user interface (GUI) where users can build and run agent-based simulations. The interface consists of a world grid where agents move and interact, along with a set of tools for defining agents, specifying their behaviors, and visualizing simulation outcomes.

Defining Agents

In NetLogo, agents can represent various entities, such as individuals, groups, or objects. Agents can be defined with specific attributes, such as position, size, color, and variables representing their state or characteristics. The behavior of agents is typically defined through procedures or rules that dictate their actions and interactions.

Interactions and Communication

Agents in NetLogo can interact with their environment and other agents through a variety of mechanisms. They can move, change their attributes, communicate through messages or signals, and influence the behavior of other agents. These interactions can be based on proximity, spatial relationships, or predefined rules.

Simulation Control

NetLogo allows users to define the rules and parameters that govern the simulation. Users can specify the duration of the simulation, control the speed of execution, and define the initial conditions and environmental constraints. NetLogo also provides tools for collecting and analyzing data during the simulation.

Experimentation and Analysis

NetLogo facilitates experimentation and analysis by allowing users to run multiple simulations with different parameter settings. Users can observe the emergent behavior of the system, collect data on agent-level interactions, and analyze simulation results using built-in statistical functions or by exporting data for further analysis in external tools.

Programming Language

NetLogo has its own programming language, which is simple and easy to learn. It is a variant of Logo, which was popular in the 1980s and 1990s as a language for teaching programming to beginners. NetLogo's language includes primitives for controlling agent behavior, accessing environment information, and managing simulation parameters.

Extension Libraries

NetLogo offers a range of extension libraries that provide additional functionality for modeling specific domains or phenomena. These extensions can include advanced visualization tools, social network analysis, geographic information system (GIS) integration, or integration with external data sources.

Simulation Environment

The simulation environment is composed of four main components: turtles, patches, links, and the observer. Each of these components plays a crucial role in the agent-based modeling process, enabling the simulation of complex systems and interactions. Here's a brief overview of each component:

Turtles: Turtles are the mobile agents in NetLogo. They represent individual entities that can move around the simulation world, interact with each other, and change their properties over time. Turtles can have attributes (variables) that determine their characteristics and behavior. For example, in a model simulating a predator-prey system, turtles could represent predators and prey. Each turtle can have its own unique set of variables that dictate its actions and interactions with the environment and other turtles.

Patches: Patches form the static grid-like world in which the turtles move. They are arranged in a two-dimensional grid and can be thought of as cells or individual units of space. Patches do not move, but they have attributes that can be accessed and modified by turtles. These attributes can represent environmental features, resources, or any other relevant information. Patches are particularly useful for modeling spatially explicit systems, and their properties can influence the behavior of the turtles that occupy them.

Links: Links represent connections or relationships between turtles in NetLogo. They are used to model networks or

associations between agents. For example, in a social network model, links could represent friendships between individuals. Links have attributes just like turtles and patches, allowing researchers to model complex interactions and information exchange between connected agents.

The Observer: The observer is not an agent in the simulation but rather an entity that oversees the entire simulation. It can be thought of as the "controller" or "manager" of the simulation. The observer can access and modify global variables, control the flow of the simulation, and execute procedures to start, stop, or modify the simulation run. Additionally, the observer can provide an interface for the user to interact with the simulation, input parameters, and view the results.

These four components together create a flexible and powerful environment for agent-based modeling in NetLogo. Modelers can define the rules and behaviors of turtles and links, access and modify patch attributes, and use the observer to run experiments and observe emergent patterns in the system. This combination allows for the exploration of a wide range of phenomena in various scientific and social domains.

Turtles

Turtles are the individual agents in NetLogo, representing entities that can move around the simulation world, interact with each other, and change their properties over time. They are a fundamental building block for creating agent-based models in NetLogo.

Creating Turtles: You can create turtles using the create-turtles command followed by the number of turtles you want to create. For example, create-turtles 100 will create 100 turtles in the simulation.

In NetLogo, turtles can be visually represented using different shapes, allowing modelers to give agents distinctive appearances in the simulation. NetLogo provides a variety of built-in shapes, and you can also create custom shapes using vector graphics.

Built-in turtle shapes

Default: The default shape is a simple triangle pointing upwards. It is the default shape assigned to turtles if no other shape is specified.

- Turtle: The turtle shape is similar to the default shape but has a more stylized appearance.
- Circle: Turtles can be represented as circles.
- Square: Turtles can be represented as squares.
- Arrow: The arrow shape is an arrow pointing upwards.
- Person: Turtles can be represented as little stick-figure-like people.
- Robot: The robot shape depicts a little robot character.
- Car: The car shape represents a simple car.
- Fish: Turtles can be represented as fish.
- Bug: The bug shape is a small bug-like creature.

- Butterfly: Turtles can be represented as butterflies.
- Gosper: The Gosper shape is a complex shape resembling a fractal pattern.

Attributes and Variables

Turtles can have attributes, which are represented as variables. These variables can hold numerical values, strings, or Boolean values. Modelers can define custom variables for turtles to represent their characteristics, states, or any other relevant information.

In NetLogo, turtles are born with several built-in attributes, also known as turtle variables, which provide them with essential properties and behaviors. These attributes allow turtles to interact with their environment, other turtles, and the simulation as a whole. Here are the 13 built-in attributes that every turtle is born with in NetLogo:

Who: This attribute uniquely identifies each turtle with an integer value. The who attribute is automatically assigned by NetLogo, and each turtle is given a distinct "who" number when it is created.

Color: The color attribute determines the color of the turtle's shape. It can be set using a color value (e.g., "red," "blue," "green") or an RGB color value.

Shape: The shape attribute specifies the shape of the turtle. It can be set to one of the built-in shapes, such as "default," "turtle," "circle," "square," or to a custom shape loaded from an SVG file.

Size: The size attribute defines the size of the turtle's shape. It can be set to a numerical value to control the size of the turtle on the simulation's graphical interface.

Heading: The heading attribute represents the current orientation or direction the turtle is facing. It is measured in degrees, with 0 representing north, 90 representing east, and so on.

Label: The label attribute is used to display text labels on turtles. It allows you to provide additional information about each turtle on the simulation interface.

Xcor and Ycor: These attributes represent the turtle's current X and Y coordinates in the grid-based world. The xcor and ycor variables determine the turtle's position in the simulation environment.

Hidden?: The hidden? attribute determines whether the turtle is hidden (not visible) on the simulation interface. A hidden turtle still exists in the simulation but is not shown.

Breed: The breed attribute specifies the breed to which the turtle belongs. Breeds are user-defined groups of turtles with shared characteristics and behavior.

Label Color: The label-color attribute sets the color of the turtle's label. It can be different from the color of the turtle's shape.

Label-Size: The label-size attribute defines the font size of the turtle's label text.

Label-Position: The label-position attribute determines the position of the turtle's label relative to the shape. It can be set to "above," "below," "left," or "right."

Label-Space: The label-space attribute specifies the distance between the turtle's shape and its label.

These built-in attributes provide turtles with various characteristics and properties, which can be used to model complex agent-based systems in NetLogo. Additionally, NetLogo allows you to create custom attributes and variables for turtles,

making the modeling capabilities even more versatile and powerful.

Moving Turtles: Turtles can move around the simulation world using the forward and backward commands to move along the current heading, or right and left commands to change their heading (orientation). For more complex movements, you can use mathematical calculations with setxy, fd, bk, rt, and lt commands.

Moving turtles is a fundamental aspect of agent-based modeling. Turtles can be programmed to move around the simulation world, responding to various conditions and interacting with the environment and other turtles. Here are some common methods for moving turtles in NetLogo:

fd (Forward) and bk (Backward) Commands:

The fd command allows turtles to move forward in the direction they are facing.

The bk command allows turtles to move backward.

Both commands take a numerical value as an argument, indicating the number of steps the turtle should move.

Example:

```
fd 10;
```

setxy Command:

The setxy command sets the turtle's position to specific X and Y coordinates in the simulation world.

It is commonly used when you want to move a turtle to a precise location.

Example:

```
Netlogo to move-turtle-to-location
setxy 5 10
end
```

rt (Right) and lt (Left) Commands:

The rt command turns the turtle to the right by a specified number of degrees. The lt command turns the turtle to the left. Both commands take an angle (in degrees) as an argument.

Example:

Netlogo to turn-right

rt 90

end

Random Movement:

Turtles can be programmed to move randomly within a certain range using the random function. By combining random values with movement commands, turtles can explore the environment in a stochastic manner.

Example:

netlogo

to move-randomly

fd random 10 ; Move forward a random distance between 0 and 10.

```
rt random 180  ; Turn right by a random angle between 0 and 180 degrees.
end
```

Slope-Based Movement:

Turtles can move along specific paths determined by slope values and distance calculations.

For example, you can use trigonometric functions to move turtles along circular paths or other geometric shapes.

Example:

```
Netlogo to move-in-circle
  let radius 5
  let angle 10
  rt angle
  fd radius * sin angle
  lt angle
end
```

These examples demonstrate different ways to move turtles in NetLogo. The movement of turtles is a crucial component of agent-based models, allowing you to explore the emergent behavior and patterns that arise from the interactions and movements of individual agents. You can combine these movement techniques with conditional statements and interactions with other turtles and the environment to create more complex and realistic simulations.

Interactions between Turtles

Turtles can interact with each other through procedures (functions) and links. They can access and modify the properties of other turtles, and they can also exchange information, resources, or coordinate actions.

Interactions between turtles are at the core of agent-based modeling in NetLogo. Turtles can interact with each other based on their positions, attributes, and behaviors, leading to emergent patterns and collective behaviors within the simulation. Here are some common ways in which turtles can interact with each other in NetLogo:

Proximity-Based Interactions

Turtles can interact with other turtles that are within a certain distance from them in the simulation world. You can use the in-radius primitive to check for turtles within a specified radius of the current turtle.

Example:

```
Netlogo to interact-with-neighbors
  let nearby-turtles turtles in-radius 5
  ask nearby-turtles [
    ; Perform interactions with neighboring turtles here
  ]
End
```

Breeds and Group-Based Interactions

Turtles can interact with turtles of the same breed or specific breeds. Breeds allow you to group turtles based on their attributes, making it easier to perform interactions within specific subgroups.

Example:

netlogo

```
to interact-with-same-breed
  ask turtles with [breed = "predators"] [
    ; Interact with other predators here
  ]
end
```

Communication and Information Sharing:

Turtles can communicate with each other by sharing information or sending messages. This can be achieved by using global variables or creating custom variables to represent shared information.

Example:

Netlogo to communicate-with-neighbors

```
  let nearby-turtles turtles in-radius 5
  ask nearby-turtles [
    set shared-variable value   ; Set a shared variable for communication
  ]
```

```
end
```

Competition and Cooperation:

Turtles can compete or cooperate with each other, leading to changes in their attributes or behaviors.

Interactions might involve competing for resources, cooperating in collective tasks, or engaging in predator-prey dynamics.

Example:

Netlogo to compete-for-resources

```
 let target one-of turtles with [breed = "food" and distance myself < 5]
  if target != nobody [
    ; Turtles compete for resources here
  ]
end
```

Link-Based Interactions:

Turtles can be connected through links, representing relationships or connections between them. Link-based interactions are useful for modeling social networks, transportation networks, and other relational structures.

Example:

Netlogo to interact-through-links

```
 ask turtles [
   let linked-turtles my-links  ; Get turtles connected to the current turtle
   ask linked-turtles [
     ; Interact with linked turtles here
   ]
 ]
end
```

Interactions between turtles in NetLogo allow you to model a wide range of phenomena, from social dynamics and ecological systems to traffic flow and economic behaviors. By defining rules and procedures for turtle interactions, you can observe the emergence of complex patterns and behaviors at the system level, which are the hallmark of agent-based modeling.

Turtle States and Behaviors

Turtle States and Behaviors: Turtles can have different states throughout the simulation, and their behaviors may change based on their current state. Modelers define procedures that dictate how turtles respond to stimuli, make decisions, and interact with their environment and other agents.

In NetLogo, turtle states and behaviors are essential components of agent-based modeling that define how individual turtles behave and how their actions change based on the state of the simulation or their own attributes. Turtle states represent the

current condition or situation of a turtle, while turtle behaviors are the actions and interactions that turtles exhibit in response to their states and the environment. Here's an overview of turtle states and behaviors in NetLogo:

Turtle States:

Turtle states refer to the different conditions or attributes that turtles can have during the simulation. These states can be based on various factors, such as the turtle's energy level, age, position in the environment, or its relationships with other turtles. Modelers can define custom states for turtles to represent specific aspects of the system being modeled. States are typically represented by turtle variables that hold numerical values, strings, or Boolean values.

Example:

```
Netlogo turtles-own [
  energy
  is-hungry?
  is-healthy?
]
```

In this example, turtles have three states: energy, is-hungry?, and is-healthy?.

Turtle Behaviors:

Turtle behaviors refer to the actions and interactions that turtles perform in response to their states and the environment. Behaviors are typically defined as procedures (functions) that dictate what turtles do under certain conditions or in specific

situations. Turtle behaviors can include moving, reproducing, consuming resources, interacting with other turtles, and responding to changes in their states or the environment.

Example:

```
Netlogo to move-turtle
  fd 1
end
to reproduce
  if energy > reproduction-energy [
    hatch 1 [
      set energy 0
    ]
  ]
end
to consume-food
  let target one-of turtles with [breed = "food" and distance myself < 3]
  if target != nobody [
    ask target [
      die
    ]
    set energy energy + energy-gain-from-food
  ]
end
```

In this example, we have defined three behaviors for turtles: move-turtle, reproduce, and consume-food.

Using States to Drive Behaviors:

The states of turtles often influence their behaviors. For example, turtles might move differently when they are hungry compared to when they are not. Conditional statements (e.g., if, ifelse) are commonly used to implement behaviors based on the current states of turtles.

Example:

```
Netlogo to move-turtle
  if is-hungry? [
    fd 0.5
  ] else [
    fd 1
  ]
end
```

In this example, the turtle moves at half speed (fd 0.5) when it is hungry (is-hungry?), and at full speed (fd 1) otherwise.

By defining turtle states and behaviors, modelers can create dynamic simulations where individual turtles respond to their internal states, the state of the environment, and the actions of other turtles. These interactions lead to emergent behaviors and patterns at the system level, providing valuable insights into the dynamics of complex systems.

Conditional Actions

Turtles can use conditional statements (e.g., if, ifelse) to make decisions based on their attributes or the state of the simulation. This allows for more sophisticated and dynamic behaviors.

Conditional actions in NetLogo involve making decisions based on specific conditions or rules. These conditions are often represented using Boolean expressions or comparisons, and they control the flow of the model, allowing turtles to take different actions depending on the current state of the simulation or their own attributes. Conditional actions are implemented using if statements (if), if-else statements (ifelse), and sometimes nested conditions. Here's how they work:

if Statement:

The if statement allows you to execute a block of code if a specified condition is true. If the condition is false, the block of code is skipped, and the model continues to the next statement after the if block. The if statement is followed by the condition in square brackets. If the condition is true, the code inside the square brackets is executed.

Example:

```
Netlogo to hungry-turtles-feed
  ask turtles [
    if energy < 50 [
      fd 1
    ]
  ]
end
```

In this example, turtles with energy less than 50 will move forward by one step.

ifelse Statement

The ifelse statement provides an alternative action when the condition of the if statement is false. If the if condition is true, the code inside the first set of square brackets is executed. Otherwise, the code inside the second set of square brackets is executed.

Example:

```
Netlogo to hungry-turtles-feed
  ask turtles [
    ifelse energy < 50 [  fd 1] [      rt 45 ]
  ]
end
```

In this example, turtles with energy less than 50 will move forward by one step, and if their energy is greater than or equal to 50, they will turn right by 45 degrees.

Nested Conditions:

You can nest if and ifelse statements within each other to create more complex conditions and behaviors. The inner if or ifelse statements will only be evaluated if the conditions of the outer statements are true.

Example:

```
Netlogo to hungry-turtles-feed
  ask turtles [
    if is-hungry? [
      if energy < 50 [
        fd 1
      ] else [
        rt 45
      ]
    ]
  ]
end
```

In this example, only turtles that are hungry will evaluate the inner condition. If their energy is less than 50, they move forward; otherwise, they turn right by 45 degrees.

Conditional actions are powerful tools for creating dynamic and adaptive agent-based models in NetLogo. By incorporating conditional statements, turtles can respond to their own states, the states of other turtles, and the state of the environment, leading to a wide variety of behaviors and interactions in the simulation.

Breed Specific Behavior

You can define multiple breeds of turtles with unique behaviors and characteristics using the breed keyword. Breeds allow you to organize different types of turtles and apply specific rules to each group. Breed-specific behavior is a feature in NetLogo that allows

you to define different behaviors and rules for turtles belonging to different breeds. Breeds are user-defined groups of turtles with shared characteristics and attributes. By creating multiple breeds and specifying unique procedures for each breed, you can simulate heterogeneous populations of agents in your model. Breed-specific behavior enhances the flexibility and realism of agent-based models, as different types of agents can behave differently, interact distinctively, and respond to the environment in unique ways.

Defining Breeds:

First, you need to define the breeds you want to use in your model using the breed keyword. For example, you can define breeds for predators and prey in a predator-prey simulation:

Netlogo

```
breed [predators predator]
breed [prey preys]
```

Creating Turtles of Different Breeds:

To create turtles of a specific breed, use the breed name followed by the create- keyword.

For example, to create five predators and ten prey:

Netlogo

```
create-predators 5
create-preys 10
```

Breed-Specific Procedures:

After creating the breeds, you can define procedures (functions) that are specific to each breed. Use the ask command to apply a procedure to all turtles of a particular breed. You can use the with keyword to filter turtles based on breed or other conditions.

Example:

```
to move-predators
  ask predators [
    ; breed-specific movement for predators
    fd 1
  ]
end
to move-prey
  ask preys [
    ; breed-specific movement for prey
    rt random 50 - 25
    fd 1
  ]
end
```

In this example, move-predators is a procedure specific to the "predators" breed, and move-prey is specific to the "prey" breed. Predators move forward, while prey move randomly within a range.

Breed-Specific Variables:

You can also define breed-specific variables that hold different values for each breed. These variables can influence breed-specific behaviors or represent breed-specific attributes.

Example:

```
breed [predators predator]
breed [prey preys]
predators-own [
  hunting-range ; Breed-specific variable for predators
]
prey-own [
  hiding-spot ; Breed-specific variable for prey
]
to setup
  clear-all
  create-predators 5 [
    set shape "wolf"
    set color red
    set hunting-range 5
  ]

  create-preys 20 [
    set shape "rabbit"
```

```
    set color blue
    set hiding-spot one-of patches
  ]
  reset-ticks
end
```

In this example, we define breed-specific variables hunting-range for predators and hiding-spot for prey.

By utilizing breed-specific behavior, you can model diverse interactions and dynamics among agents in your NetLogo model. Each breed can have its own attributes, rules, and interactions, enabling you to explore complex systems where different types of agents exhibit unique behaviors in response to their environment and other agents.

Appearance

Turtles can be visually represented using various shapes, colors, and sizes. NetLogo provides built-in shapes, and you can create custom shapes using vector graphics. The appearance of turtles and patches can be customized to create a visually appealing and informative simulation. Appearance settings allow you to control the color, shape, size, and labels of turtles and patches on the simulation's graphical interface. By adjusting these appearance settings, you can enhance the visual representation of your agent-based model and convey information more effectively. Here are some of the appearance-related features in NetLogo:

Turtle Shapes:

Turtles can be visually represented using different shapes, such as triangles, circles, squares, or custom shapes in SVG format. The set shape command is used to change the shape of turtles.

Example:

set shape "circle"

Turtle Colors:

The color of turtles can be customized to distinguish between different breeds or represent various states of the agents. The set color command is used to change the color of turtles.

Example:

set color red

Turtle Size:

The size of turtles can be adjusted to make them more visible and prominent on the graphical interface. The set size command is used to change the size of turtles.

Example:

set size 2

Labels:

Labels can be added to turtles to display additional information about them. Labels are useful for showing attributes

or states of turtles directly on the graphical interface. The set label command is used to add labels to turtles.

Example:

set label "Agent 1"

Label Color and Size:

The color and font size of the labels can be customized to improve readability. The set label-color and set label-size commands are used to change the label color and size, respectively.

Example:

set label-color white

set label-size 12

Patches Colors:

Patches, which make up the grid-based environment in NetLogo, can have different colors to represent different states or conditions. The set pcolor command is used to change the color of patches.

Example:

set pcolor green

Patch Colors Based on Values:

Patches can be colored based on their values, allowing you to visualize data on the grid. For example, you can use the set pcolor scale-color command to color patches based on numeric values.

Example:

```
set pcolor scale-color blue [value] [0 100]
```

These appearance settings, combined with other aspects of NetLogo's visualization capabilities, allow you to create visually engaging and informative agent-based models. Proper visualization enhances the explanatory power of your model and facilitates a better understanding of the simulation's dynamics and emergent patterns.

Linking Turtles

Turtles can be connected to each other using links. Links represent relationships or connections between turtles and are useful for modeling networks, social interactions, and other forms of relationships. Linking turtles in NetLogo allows you to establish connections or relationships between individual turtles, creating networks or social structures within the model. The process of linking turtles involves creating links (edges) between turtles to represent interactions, dependencies, or other connections among them. In NetLogo, links can be directed or undirected and can have attributes that store additional information about the relationship between turtles. Here's how you can link turtles in NetLogo:

Defining Link Breeds:

Before linking turtles, you need to define the types of links you want to use in your model using the breed keyword for links. You can define multiple breeds of links to represent different types of relationships.

Example:

breed [predators predator]

breed [prey preys]

breed [predator-prey-links predator-prey-link]

Creating Links:

To create a link between two turtles, use the create-link-to or create-links-with command. You can specify the breed of the link and the target turtle(s) as arguments.

Example:

```
ask predator 0 [
  create-link-to prey 0
]
```

In this example, we create a link from the predator with who number 0 to the prey with who number 0.

Directed and Undirected Links:

By default, links are undirected, meaning that they do not have a specific direction. If you want to create directed links, you can use the create-link-from command to establish links from one turtle to another.

Example:

```
ask predator 0 [
  create-link-from prey 0
]
```

In this example, we create a directed link from the prey with who number 0 to the predator with who number 0.

Link Attributes:

Links can have attributes that store additional information about the relationship between turtles. You can define and access link variables in a similar way to turtle variables.

Example:

```
breed [predators predator]
breed [prey preys]
breed [predator-prey-links predator-prey-link]
predator-prey-links-own [
  strength
]
to setup
  clear-all
  create-predators 5
  create-preys 10
```

```
  ask predators [
    create-link-to one-of prey [
      set strength random 100
    ]
  ]
  reset-ticks
end
```

In this example, we define a link attribute called strength to represent the strength of the relationship between predators and prey.

Linking turtles in NetLogo is useful for modeling social networks, predator-prey interactions, transportation systems, and various other relational structures within your agent-based model. Links provide a powerful mechanism for representing complex relationships between agents, which can lead to more sophisticated and realistic simulations.

NetLogo Examples

Example 1

Example 1: create turtles and have them move forward

In this example, the code is very minimal. When you run this code in NetLogo, it will create 10 turtles and place them at the center of the canvas. The turtles will then move forward by 1 unit at each tick.

Instructions to use the code:

Open NetLogo and create a new model.

Copy the above code and paste it into the Code tab of the NetLogo interface.

Click on the "Setup" button to create 10 turtles at the center of the canvas.

Click on the "Go" button to start the turtles' movement. They will move forward at each tick, but since the code does not include turning commands, they will all move in the same direction.

```
to setup
  clear-all
  create-turtles 10
  reset-ticks
end

to go
  ask turtles [
    fd 1  ; Move the turtle forward by 1 unit
  ]
  tick
end
```

Foraging scenario

Where turtles move around the canvas, collecting food from patches. The patches will represent food sources, and turtles will search for and eat the food, leaving a trail behind as they move.

In this example, we have added behaviors to the turtles to make them seek out and eat food represented by the green patches on the canvas. When a turtle finds a food patch within a radius of 3 units, it moves towards the patch, eats the food (changes the patch color to white), and leaves a label "+" on the patch to indicate that it was consumed.

Here's how you can use this code:

1. Open NetLogo and create a new model.
2. Copy the above code and paste it into the Code tab of the NetLogo interface.
3. Click on the "Setup" button to create 10 turtles at random positions and set up the food sources (green patches) on the canvas.
4. Click on the "Go" button to start the simulation. Turtles will move around, forage for food, and leave a trail of white patches with "+". The food patches will disappear as they get consumed.

You can further expand this simulation by introducing more complex behaviors, adding obstacles, varying the distribution of food sources, or implementing different movement strategies for the turtles. NetLogo's flexibility allows you to create various scenarios and explore emergent behaviors in multi-agent systems.

Example 3

We'll create a scenario where turtles forage for food, but they also need to avoid obstacles represented by red patches. Additionally, we'll add a condition for the turtles to reproduce when they have collected a certain amount of food.

```
to setup
  clear-all
  ask patches [
    set pcolor green   ; Set the color of all patches to green (representing food)
    set plabel ""      ; Clear any existing labels on patches
  ]
  ask n-of 50 patches [
    set pcolor red    ; Set the color of 50 patches to red (representing obstacles)
  ]
  create-turtles 5 [
    set shape "turtle"
    set color blue    ; Set the color of each turtle to blue
    set size 1.5      ; Set the size of each turtle
    setxy random-xcor random-ycor ; Place turtles at random positions on the canvas
    set food-collected 0 ; Initialize the food-collected variable for each turtle
  ]
```

```
  reset-ticks
end


to go
  ask turtles [
    ; Check for nearby food patches
    let target one-of patches in-radius 3 with [pcolor = green]

    ; Check for nearby obstacles
    let obstacle one-of patches in-radius 2 with [pcolor = red]

    if obstacle = nobody [
      if target != nobody [
        face target        ; Face the target patch
        fd 1               ; Move the turtle toward the target
        set pcolor white   ; Change the color of the patch to white (food eaten)
        set plabel "+"     ; Label the patch to indicate that food was consumed
        set food-collected food-collected + 1 ; Increment food-collected by 1
      ]
    ] else [
      ; If an obstacle is nearby, turn away from it
```

```
    rt random 180
  ]

  ; Reproduce if the turtle has collected enough food
  if food-collected >= 5 [
    hatch 1 [
      set shape "turtle"
      set color blue
      set size 1.5
      set food-collected 0 ; Reset food-collected for the new turtle
      rt random 360       ; Randomly turn the new turtle
      fd 2                ; Move the new turtle forward by 2 units
    ]
    set food-collected 0 ; Reset food-collected for the current turtle
  ]
 ]
 tick
end
```

In this enhanced example, we've added a few new features:

Obstacles: We added red patches as obstacles. Turtles will turn away from these obstacles if they are within a radius of 2 units.

Food Collection: When turtles find food patches (green), they move towards them and consume the food. The food-collected variable keeps track of the number of food items each turtle has eaten.

Reproduction: When a turtle has collected at least 5 units of food, it reproduces by creating a new turtle near its location. The new turtle inherits the parent's color, shape, and size, and it starts with a food-collected value of 0.

Here's how you can use this code:

Open NetLogo and create a new model.

Copy the above code and paste it into the Code tab of the NetLogo interface.

Click on the "Setup" button to create 5 turtles at random positions, set up the food sources (green patches), and add obstacles (red patches) on the canvas.

Click on the "Go" button to start the simulation. Turtles will move around, forage for food, avoid obstacles, and reproduce when they have collected enough food.

You can further experiment and modify the code to add more complex behaviors and interactions between turtles, food, and obstacles. NetLogo provides a powerful platform for modeling and exploring emergent behavior in multi-agent systems.

Examples

Random Walk:

Description: Agents move randomly on a 2D grid.

```
globals [num-agents]
to setup
  clear-all
  set num-agents 50
  create-turtles num-agents [
    setxy random-xcor random-ycor
  ]
reset-ticks
end

to go
```

```
  ask turtles [
    rt random 360
    fd 1
  ]
  tick
end
```

Step-by-step guide:

Create a new NetLogo model and clear the existing code.

Copy and paste the above code into the Code tab.

Click the "setup" button to initialize the simulation.

Click the "go" button to start the random walk simulation.

NetLogo is a powerful multi-agent modeling environment used for simulating complex systems. Below are ten example models with step-by-step guides to get you started. For each example, I'll provide a brief description, followed by the NetLogo code and step-by-step instructions on how to set up and run the simulation.

Please note that the code examples below assume you have NetLogo installed on your computer. If you don't have it, you can download it from the official website (ccl.northwestern.edu/netlogo/).

1. Random Walk: Description: Agents move randomly on a 2D grid.

NetLogo Code:

```
netlogo
globals [num-agents]

to setup
  clear-all
  set num-agents 50
  create-turtles num-agents [
    setxy random-xcor random-ycor
  ]
  reset-ticks
end

to go
  ask turtles [
    rt random 360
    fd 1
  ]
  tick
end
```

Step-by-step guide:

- Create a new NetLogo model and clear the existing code.
- Copy and paste the above code into the Code tab.
- Click the "setup" button to initialize the simulation.
- Click the "go" button to start the random walk simulation.

2. Boids: Description: Simulate flocking behavior inspired by bird flocking.

NetLogo Code:

```
netlogo
breed [boids boid]

to setup
  clear-all
  create-boids 100 [
    set shape "fish"
```

```
    set color blue
    setxy random-xcor random-ycor
    set heading random-float 360
  ]
  reset-ticks
end

to go
  ask boids [
    align
    cohere
    separate
    fd 1
  ]
  tick
end

to align
  ; alignment code here
end

to cohere
  ; cohesion code here
end

to separate
  ; separation code here
end
```

Step-by-step guide:

- Create a new NetLogo model and clear the existing code.
- Copy and paste the above code into the Code tab.
- Implement the `align`, `cohere`, and `separate` procedures (flocking rules) inside the respective placeholders.
- Click the "setup" button to initialize the simulation.
- Click the "go" button to start the Boids simulation.

3. Predator-Prey (Foxes and Rabbits): Description: Simulate a predator-prey interaction between foxes and rabbits.

NetLogo Code:

```
netlogo
breed [rabbits rabbit]
breed [foxes fox]

to setup
  clear-all
  create-rabbits 50 [
    setxy random-xcor random-ycor
  ]
  create-foxes 20 [
    setxy random-xcor random-ycor
  ]
  reset-ticks
end

to go
  ask rabbits [
    move
    reproduce-rabbits
  ]
  ask foxes [
    move
    reproduce-foxes
    hunt
  ]
  tick
end

to move
  ; movement code here
end

to reproduce-rabbits
  ; reproduction code here
end

to reproduce-foxes
  ; reproduction code here
end

to hunt
  ; hunting code here
end
```

Step-by-step guide:

- Create a new NetLogo model and clear the existing code.
- Copy and paste the above code into the Code tab.
- Implement the `move`, `reproduce-rabbits`, `reproduce-foxes`, and `hunt` procedures inside the respective placeholders.
- Click the "setup" button to initialize the simulation.
- Click the "go" button to start the predator-prey simulation.

4. Forest Fire Spread: Description: Simulate the spread of a forest fire.

NetLogo Code:

```
netlogo
globals [prob-spontaneous-growth prob-lightning]

to setup
  clear-all
  set prob-spontaneous-growth 0.01
  set prob-lightning 0.001
  ask patches [
    ifelse random-float 1 < 0.2
      [set pcolor green]
      [set pcolor brown]
  ]
  reset-ticks
end

to go
  ask patches with [pcolor = green] [
    if any? neighbors with [pcolor = red]
      [ifelse random-float 1 < prob-lightning
        [set pcolor red]
        [ifelse random-float 1 < prob-spontaneous-
growth
          [set pcolor green]
          [ask one-of neighbors with [pcolor = red]
[set pcolor red]]
        ]
```

```
      ]
  ]
  tick
end
```

Step-by-step guide:

- Create a new NetLogo model and clear the existing code.
- Copy and paste the above code into the Code tab.
- Click the "setup" button to initialize the simulation.
- Click the "go" button to start the forest fire spread simulation.

5. Schelling's Segregation Model: Description: Simulate residential segregation based on Schelling's model.

NetLogo Code:

```
netlogo
globals [threshold]

to setup
  clear-all
  set threshold 40
  ask patches [
    ifelse random-float 1 < 0.7
      [set pcolor white]
      [set pcolor black]
  ]
  ask n-of (count patches * 0.3) patches [set
pcolor gray]
  reset-ticks
end

to go
  ask patches [
    if pcolor != gray [
      let same-color count neighbors with [pcolor =
[pcolor] of myself]
      let different-color count neighbors with
[pcolor != [pcolor] of myself]
      ifelse (same-color + different-color) > 0 [
```

```
        if (same-color / (same-color + different-
color)) < threshold / 100 [
          move-to one-of patches with [pcolor =
gray]
        ]
      ] [
        move-to one-of patches with [pcolor = gray]
      ]
    ]
  ]
  tick
end
```

Step-by-step guide:

- Create a new NetLogo model and clear the existing code.
- Copy and paste the above code into the Code tab.
- Click the "setup" button to initialize the simulation.
- Click the "go" button to start the Schelling's segregation simulation.

6. Virus Spread: Description: Simulate the spread of a virus in a population.

NetLogo Code:

```
netlogo
globals [infection-radius infection-probability re-
covery-time]

to setup
  clear-all
  set infection-radius 2
  set infection-probability 0.5
  set recovery-time 20
  create-turtles 100 [
    setxy random-xcor random-ycor
    set color blue
    set shape "person"
    set size 1
  ]
  ask n-of 5 turtles [set color red]
  reset-ticks
```

```
end

to go
  ask turtles [
    if color = red [
      spread-virus
      recover
    ]
  ]
  tick
end

to spread-virus
  ask turtles in-radius infection-radius with
[color = blue and random-float 1 < infection-proba-
bility] [
```

***