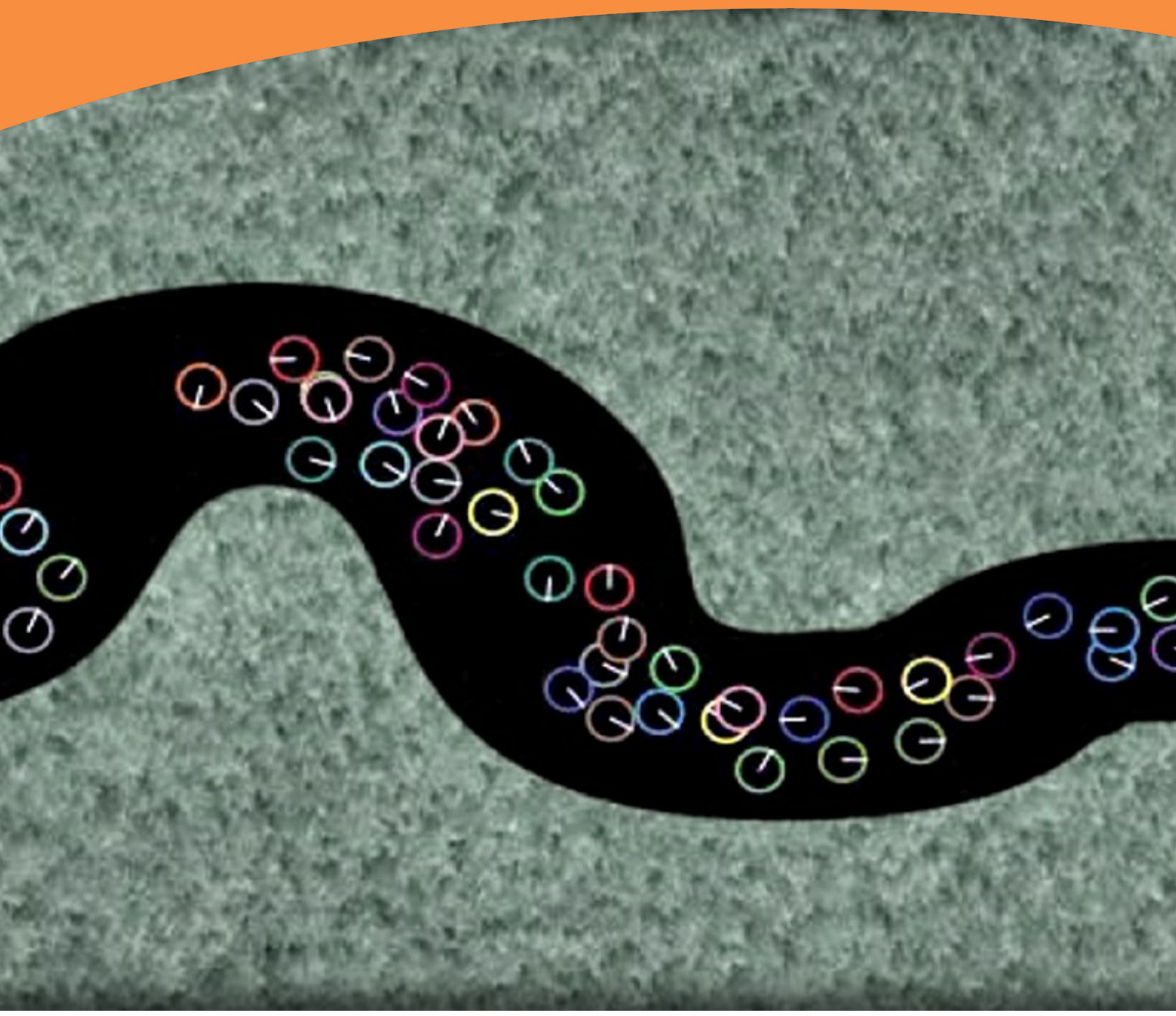
Artificial Intelligence: Exercises II

Agent Behaviour I

William John Teahan



bookboon
The eBook company

William Teahan

Artificial Intelligence:

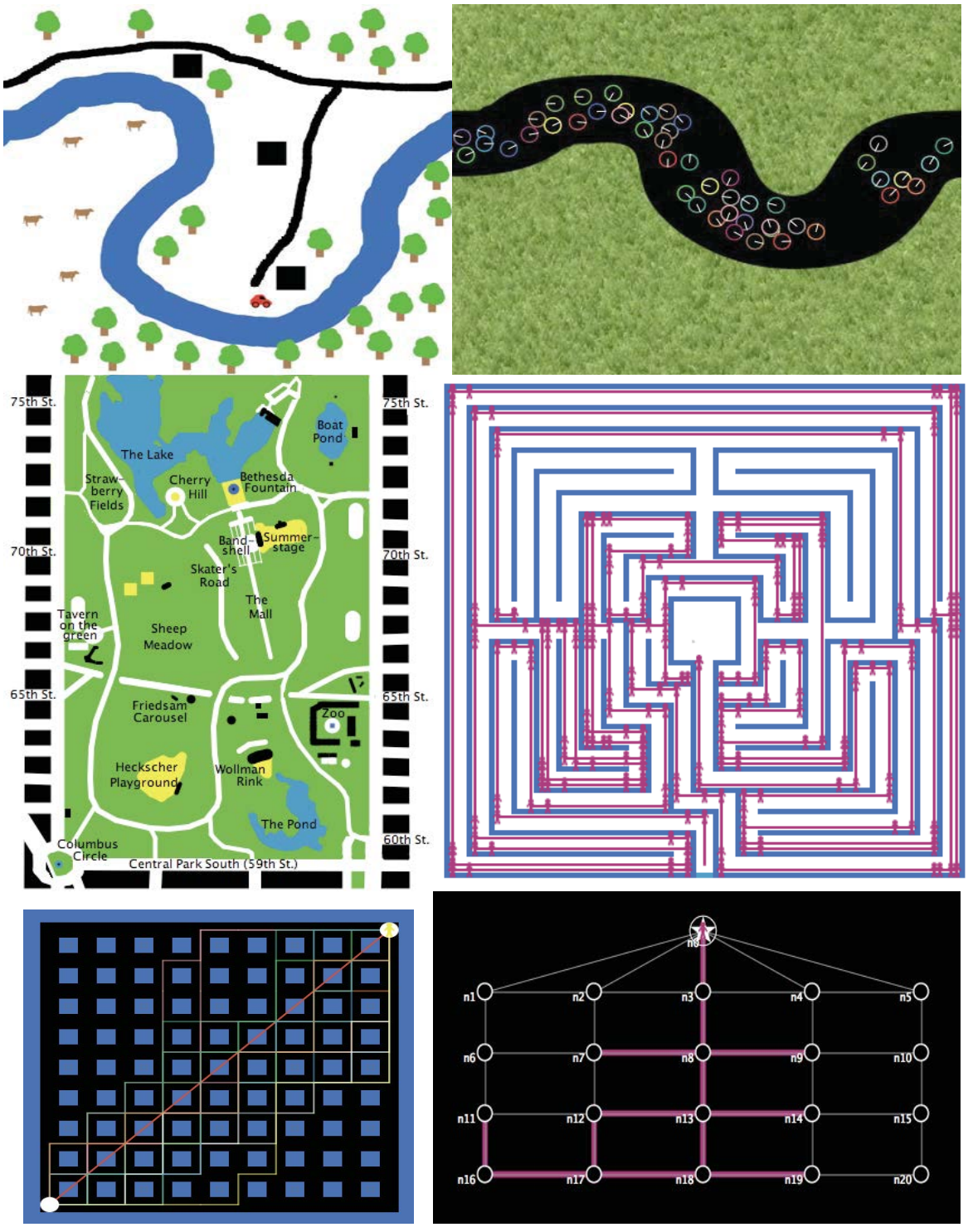
Exercises – Agent Behaviour I

Artificial Intelligence: Exercises – Agent Behaviour I
1st edition

ISBN 978-87-7681-592-9

Contents

Exercises for Artificial Intelligence – Agent Behaviour I



Selection of screenshots taken from NetLogo models described in this book.

Preface

The list of exercises, chapter headings and section, and NetLogo models in this book closely follow what is in the companion "*Artificial Intelligence – Agent Behaviour I*" book. The best way to learn about what is written in the companion book is to try out each of the NetLogo models that are described in the book and in the exercises below. An index of the models used in these books can be found using the following URL:

NetLogo Models for Artificial Intelligence	http://files.bookboon.com/ai/index.html

A table listing all the models described in this book and the companion book is also provided below. Each row in the table lists the name of the model, the exercises where it is described, a short description of the model, and a URL where it can be found. Each of these models have sections in the `Information` tab that provide various documentation, such as: what the model is; how it works; how to use it; the meaning of each of the Interface's buttons, sliders, switches, choosers, monitors, plots and output; important things to notice; things to try out; suggestions for extending the model; explanations of interesting NetLogo features used in the model; credits and references; and links to related models. In particular, the sections on how to use it, things to notice and things to try out provide some suggestions on various things a user can try when playing with the models.

The reader, however, should not restrict themselves to just these suggestions. Due to the complex system nature of many of the simulations that result from the running of these NetLogo models, often unforeseen phenomena emerge as a result of the agent – agent and agent – environment interactions. The reader is encouraged to become an 'explorer' of the virtual environments created by these models by trying out as many of the different combinations of the slider, switch and chooser values as possible while running the simulations many times to ensure that a representative sampling of the possible system behaviours is observed.

Ants (6.5.1, Solution to 6.5.1)

This model simulates a colony of ants foraging for food. In NetLogo's Models Library: Biology > Ants. http://ccl.northwestern.edu/netlogo/models/Ants

Ants (Perspective Demo) (6.4.2)

This is a variation of the Ants model that offers different perspectives. In NetLogo's Models Library: Perspective Demos > Ants (Perspective Demo). http://ccl.northwestern.edu/netlogo/models/AntsPerspectiveDemo

ANZ Continental Drift (6.3.2, Solution to 6.3.2)

This model shifts New Zealand back towards Australia in order to illustrate the process of continental drift. http://files.bookboon.com/ai/ANZ-Contienental-Drift.html

Cars Guessing Game (7.8.1)

This model plays a simple game trying to guess the colour of cars as they drive past. Its purpose is to show how entropy and code length calculations are made given a probability distribution. http://files.bookboon.com/ai/Cars-Guessing-Game.nlogo

Central Park Events (9.8.1)

This model visualises a sequence of events that are necessary for going from the Zoo to the Boat Pond in Central Park, New York. http://files.bookboon.com/ai/Central-Park-Events.html

Being Kevin Bacon (7.5.1)

This model implements various algorithms related to communication amongst agents in a network, such as Dijkstra's algorithm, and communication via word-of-mouth or using blackboards. It also demonstrates some important concepts such as the small world phenomenon, degrees of separation, and super-nodes in peer-to-peer networks. http://files.bookboon.com/ai/Being-Kevin-Bacon.nlogo

Chatbot (10.7.1)

This model implements two basic chatbots – Liza and Harry – using regular expressions. http://files.bookboon.com/ai/Chatbot.html

Colour Cylinder (9.4.3)

This model demonstrates how colour can be represented in 3 dimensions: hue, saturation and brightness. http://files.bookboon.com/ai/Colour-Cylinder.html

Communication T-T Example 2 (7.4.1)

This model simulates the spreading of a message between agents. This is a modification of the Communication T-T Example model in NetLogo's Models Library:
Code Examples > Communication-T-T Example; for modified model used here:
http://files.bookboon.com/ai/Communication-T-T-Example-2.nlogo

Crowd Path Following (6.7.1)

This model is an attempt to recreate boids (see Craig Reynold's work) that use the crowd path following steering behaviour. http://files.bookboon.com/ai/Crowd-Path-Following.html

Entropy Calculator (7.7.1, Solution to 7.7.1)

This model allows the user to calculate the entropy for a specific probability distribution. http://files.bookboon.com/ai/Entropy-Calculator.nlogo

Firebreak (6.1.2, Solution to 6.1.2)

This model is an extension of the Fire model that allows users to add firebreaks, extra forest and ignition points. A satellite or aerial image can also be imported into the environment. http://files.bookboon.com/ai/Firebreak.html

Fireflies (6.3.1, Solution to 6.3.1, 6.5.1, Solution to 6.5.1) This model simulates the synchronization of flashing behaviour in fireflies. In NetLogo's Models Library: Biology > Fireflies http://ccl.northwestern.edu/netlogo/models/Fireflies
Flocking (6.3.1, Solution to 6.3.1, 6.5.1, Solution to 6.5.1) This model simulates the flocking behaviour of birds. In NetLogo's Models Library: Biology > Flocking. http://ccl.northwestern.edu/netlogo/models/Flocking
Flocking (Perspective Demo) (6.4.2) This is a variation of the Flocking model that offers different perspectives. In NetLogo's Models Library: Perspective Demos > Flocking (Perspective Demo). http://ccl.northwestern.edu/netlogo/models/FlockingPerspectiveDemo
Flocking With Obstacles (6.7.2) This model is an extension of the Flocking model with obstacles added. http://files.bookboon.com/ai/Flocking-With-Obstacles.html
Follow and Avoid (6.7.3) This model is an attempt to recreate boids (see Craig Reynold's work) that use seeking and fleeing steering behaviours. http://files.bookboon.com/ai/Follow-And-Avoid.html
Heatbugs (6.5.1, Solution to 6.5.1) This model simulates the behavior of biologically inspired agents that attempt to maintain an optimum temperature around themselves. In NetLogo's Models Library: Biology > Heatbugs. http://ccl.northwestern.edu/netlogo/models/Heatbugs
Knowledge Representation (9.8.2) This model visualises the knowledge and reasoning processes for three toy problems using different methods for knowledge representation. http://files.bookboon.com/ai/Knowledge-Representation.html
Language Change (7.3.1) This model shows how social networks amongst language users can affect the course of language change. In NetLogo Model's Library: Social Science > Language Change http://ccl.northwestern.edu/netlogo/models/LanguageChange

Language Modelling (7.9.1)

This model shows how a language model can be constructed from some training text. Its purpose is to show various important features of language models and to visualise them using NetLogo link and turtle agents.
http://files.bookboon.com/ai/Language-Modelling.nlogo

Map Drawing (9.7.1)

Users can create their own maps using this model. http://files.bookboon.com/ai/Map-Drawing.html

Mazes-2 (6.1.1)

This model extends the Mazes model. It gets a simple reactive turtle agent to move around several mazes.
http://files.bookboon.com/ai/Mazes-2.html

Missionaries and Cannibals (8.2.2)

This model applies standard search algorithms to the classic search problem called Missionaries and Cannibals.
http://files.bookboon.com/ai/Missionaries-and-Cannibals.html

Moths (6.5.1, Solution to 6.5.1)

This model demonstrates moths flying around a light in circles each following a set of simple rules. In NetLogo's Models Library: Biology > Moths. http://ccl.northwestern.edu/netlogo/models/Moths

Obstacle Avoidance 1 (6.7.4)

This model is an attempt to recreate boids (see Craig Reynold's work) that employs basic obstacle avoidance steering behaviour. http://files.bookboon.com/ai/Obstacle-Avoidance-1.html

Obstacle Avoidance 2 (6.7.5)

This model is an attempt to recreate boids (see Craig Reynold's work) that employs basic obstacle avoidance steering behaviour. http://files.bookboon.com/ai/Obstacle-Avoidance-2.html

Searching for Kevin Bacon (8.2.3)

This model applies standard search algorithms to the problem of searching for a specific goal node in a network.
http://files.bookboon.com/ai/Searching-for-Kevin-Bacon.html

Searching for Kevin Bacon 2 (8.2.4, 8.3.1, 8.6.2, 8.7.1, Solutions to 8.3.1, 8.6.2, 8.7.1)

This model is an extension to the Searching for Kevin Bacon model. It provides an Output box that allows the user to trace how the search proceeds. http://files.bookboon.com/ai/Searching-for-Kevin-Bacon-2.html

Searching Mazes (8.2.1)

This model applies standard search algorithms to the problem of searching mazes.
http://files.bookboon.com/ai/Searching-Mazes.html

Shannon Guessing Game (7.10.1)

This model shows how a language model can be constructed from some training text and then used to predict text – i.e. play the "Shannon Guessing Game", a game where the agent (human or computer) tries to predict upcoming text, one letter at a time, based on the prior text it has already seen.
http://files.bookboon.com/ai/Shannon-Guessing-Game.nlogo

State Machine Example (6.3.1, Solution to 6.3.1) This model simulates termites creating piles of wood chips. In NetLogo's Models Library: Code Examples > State Machine Example http://ccl.northwestern.edu/netlogo/models/StateMachineExample
State Machine Example 2 (6.3.3, Solution to 6.3.3) This model adds a function and plot to estimate and graph the self-organisation of the termites for the State Machine Example model. http://files.bookboon.com/ai/State-Machine-Example-2.html
Termites (6.3.1, 6.5.1, Solution to 6.3.1) This model simulates termites creating piles of wood chips. In NetLogo's Models Library: Biology > Termites. http://ccl.northwestern.edu/netlogo/models/Termites
Termites (Perspective Demo) 6.4.2) This is a variation of the Termites model that offers different perspectives. In NetLogo's Models Library: Perspective Demos > Termites (Perspective Demo). http://ccl.northwestern.edu/netlogo/models/TermitesPerspectiveDemo
Virus (6.5.1, Solution to 6.5.1) This model simulates how a virus spreads in a human population. In NetLogo's Models Library: Biology > Virus. http://ccl.northwestern.edu/netlogo/models/Virus
Wall Following Example 2 (6.7.6) This is a variation of the Wall Following model where the turtles' wall following behaviour has been split into three sub-behaviours which are executed in random order. In NetLogo's Models Library: Code Examples > Wall Following Example; see modified code at: http://files.bookboon.com/ai/Wall-Following-Example-2.html
Water Flowing Uphill (10.8.1) This model tries to visually simulate one possible solution to the problem of trying to get water to flow uphill. http://files.bookboon.com/ai/Water-Flowing-Uphill.html

6 Behaviour

6.1 What is behaviour?

Exercise 6.1.1:

Try out the different behaviours for the turtle agent in the Mazes-2 NetLogo model:

Mazes-2	http://files.bookboon.com/ai/Mazes-2.html

Figure 6.1.1. Screenshot of the Interface for the Mazes-2 model after the setup button has been pressed followed by the go-forever button, with the chooser and slider values as shown in the image, and with the simulation left running for a short while.

Do this by changing the value of the `turtle-behaviour` chooser in the `Interface`. Characterise each behaviour by the set of actions the turtle carries out when performing that behaviour. Characterise these actions that comprise each behaviour by the set of movements that the turtle performs when carrying out the action. Which behaviour seems to be the most effective on a particular maze, and which seems to be most effective across all mazes? Which behaviours end up with the turtle agent getting stuck in an endless loop in some of the mazes?

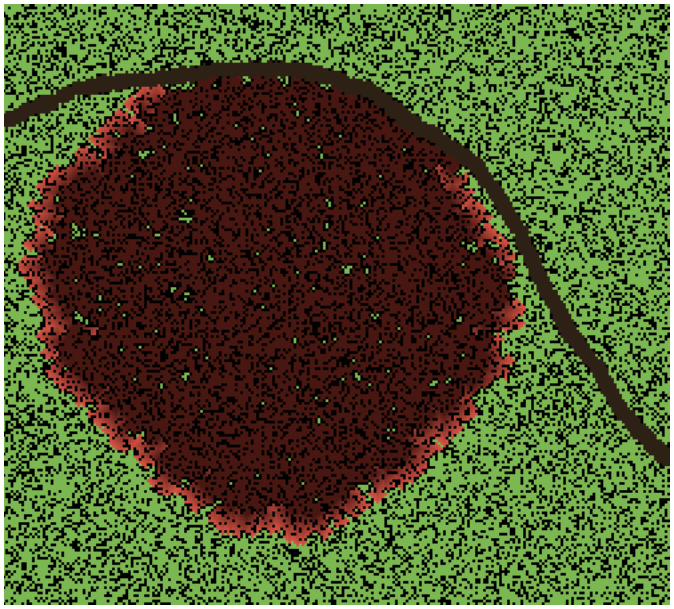
Exercise 6.1.2:

Try out the Fire model in NetLogo:

Fire	http://ccl.northwestern.edu/netlogo/models/Fire

How does the fire behave when the value of the `density` slider is increased or decreased? Characterise the behaviour of the fire by the patterns of behaviour that result from the different slider values – for example, the amount of the forest that is burnt, the pattern of burning, and the pattern of the remaining unburnt trees.

Extend the Fire model by adding a `draw firebreak` button to allow the user to create firebreaks in the forest to stop the spread of the fire. Add two further buttons – `draw forest` and `ignite forest` – for allowing the user to draw extra trees in the forest, and to specify specific locations for where the fire starts rather than having it start in a line on the left of the environment as with the Fire model. For example, in the image below, the firebreak has been drawn as a dark brown curve from top left to bottom right. The fire was ignited in the centre of the burnt-out crimson circle, and is shown to be expanding outwards on the outer rim of the circle except where it is blocked by the firebreak.



Add a fourth button – `import forest` – for importing a forest and its surrounding geography directly from a satellite image or aerial photo so you can use the model to simulate how a fire might spread in a real-life environment.

Use the extended model to investigate how the geography and different types of firebreaks affect the behaviour of the fire.

6.2 Reactive versus Cognitive Agents

Exercise 6.2.1:

An agent wishes to go through a door but the door is locked. An agent adopting the cognitive approach has constructed the following plan in its 'head':

Plan for opening the door:
Go to the place where the key is.
Take the key.
Go to the door.
Open the door with the key.

Suggest some rules that a reactive agent could use to solve the same task. What are the limitations of the approach that the reactive agent uses compared to the cognitive agent?

6.3 Emergence, Self-organisation, Adaptivity and Evolution

Exercise 6.3.1:

Try out the following models in NetLogo:

Flocking	In NetLogo's Models Library: Biology > Flocking http://ccl.northwestern.edu/netlogo/models/Flocking
Fireflies	In NetLogo's Models Library: Biology > Fireflies http://ccl.northwestern.edu/netlogo/models/Fireflies
Termites	In NetLogo's Models Library: Biology > Termites http://ccl.northwestern.edu/netlogo/models/Termites
State Machine Example	In NetLogo's Models Library: Code Examples > State Machine Example http://ccl.northwestern.edu/netlogo/models/StateMachineExample

Explain for each of these models whether you witness any of the following:

- Reactive behaviour (in the turtles);
- Cognitive behaviour (in the turtles);
- Emergence;
- Self-organisation.

In order to answer this question, fill in the rest of the table provided below. Hypothetical answers are provided for the Hypothetical model (which does not exist in the table below).

Model	**Reactive behaviour**	**Cognitive behaviour**	**Emergence**	**Self-oganisation**
Hypothetical	*Yes, because…*	*No, because…*	*No, because…*	*No, because…*
Flocking				
Fireflies				
Termites				
State Machine Example				

Exercise 6.3.2:

Try creating a model in NetLogo to illustrate continental drift between Australia and New Zealand. Hint: Use the `import-pcolors` command to import an image containing the present shapes of Australia and New Zealand. Add a monitor into the model to show an estimate of the years passed as the two countries move slowly back together.

Exercise 6.3.3:

Is there some way of measuring self-organisation when it appears in a model? For example, for the Termites and State Machine Example models, is there some way of measuring how successful the turtle agents have been at moving the woodchips into piles?

Exercise 6.3.4:

Is there some way of determining when an emergent property appears in a model?

6.4 The Frame of Reference Problem

Exercise 6.4.1:

The default mode in NetLogo models is for the observer agent to be looking down on the world from above centered at the origin; this is usually at the co-ordinates (0, 0). The frame of reference in this case is a third-person perspective of the world. That is, the observer agent is external to the world being observed. It is possible in NetLogo to change to a first person perspective using the `follow`, `ride` and `watch` observer commands, and the `follow-me`, `ride-me` and `watch-me` turtle commands. Find out what each of these commands does. Develop a model or models of your own to demonstrate their use.

Exercise 6.4.2:

The models in the Perspective Demos directory of the NetLogo's Models Library illustrate how these commands can be used to change the perspective from third-person to first-person for some of the models in the Models Library. Have a play with the Ants (Perspective Demo), Flocking (Perspective Demo) and Termites (Perspective Demo) models. Look at the code to see how the commands are used in the models.

Ants (Perspective Demo)	In NetLogo's Models Library: Perspective Demos > Ants (Perspective Demo) http://ccl.northwestern.edu/netlogo/models/AntsPerspectiveDemo
Flocking (Perspective Demo)	In NetLogo's Models Library: Perspective Demos > Flocking (Perspective Demo) http://ccl.northwestern.edu/netlogo/models/FlockingPerspectiveDemo
Termites (Perspective Demo)	In NetLogo's Models Library: Perspective Demos > Termites (Perspective Demo) http://ccl.northwestern.edu/netlogo/models/TermitesPerspectiveDemo

Try out these models in both 2D and 3D. Press the `watch one-of-turtles` and `follow one-of turtles` buttons to see what happens. How easy is it to discern the global behaviour of the colony or flock when you are watching a single turtle zoomed most of the way in or when zoomed all of the way in (i.e. when you are in 'ride' mode)? Slow down the speed of the simulation so that you verify the local behaviour of the turtle you are watching from a first person perspective. Conversely, how easy is to discern the local behaviour of each turtle when you are no longer watching one but are the observer looking at the world from above?

6.5 Stigmergy and Swarm Intelligence

Exercise 6.5.1:

Which of the following models in NetLogo's Models Library are examples of swarm intelligence where the agents make use of stigmergic local knowledge?

Ants	In NetLogo's Models Library: Biology > Ants http://ccl.northwestern.edu/netlogo/models/Ants
Heatbugs	In NetLogo's Models Library: Biology > Heatbugs http://ccl.northwestern.edu/netlogo/models/Heatbugs
Fireflies	In NetLogo's Models Library: Biology > Fireflies http://ccl.northwestern.edu/netlogo/models/Fireflies
Flocking	In NetLogo's Models Library: Biology > Flocking http://ccl.northwestern.edu/netlogo/models/Flocking
Moths	In NetLogo's Models Library: Biology > Moths http://ccl.northwestern.edu/netlogo/models/Moths
Termites	In NetLogo's Models Library: Biology > Termites http://ccl.northwestern.edu/netlogo/models/Termites
Virus	In NetLogo's Models Library: Biology > Virus http://ccl.northwestern.edu/netlogo/models/Virus

6.6 Implementing behaviour of Turtle Agents in NetLogo

Exercise 6.6.1: Heatbugs NetLogo Model

Try out the Heatbugs model in NetLogo.

Heatbugs	In NetLogo's Models Library: Biology > Heatbugs http://ccl.northwestern.edu/netlogo/models/Heatbugs

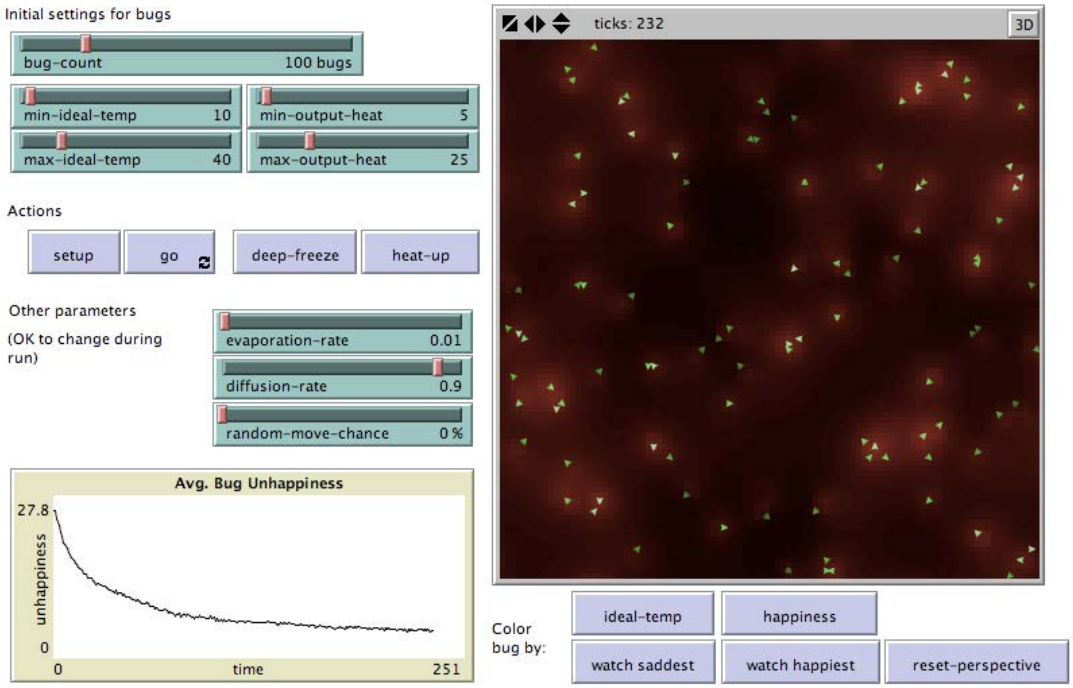


Figure 6.6.1. Screenshot of the Interface for the Heatbugs model after the setup button has been pressed followed by the go button with the slider values as shown in the image.

The model's `Information` states that it has been used as a demonstration model for many agent-based modeling toolkits, with several different kinds of emergent behaviour resulting from simple rules. Run the model multiple times to observe what happens for the different slider values. Make a list of the emergent behaviours you observe.

How is the temperature diffused throughout the environment in this model?

Each tick, all the turtles are directed to perform the `step` procedure. Where is this done in the model? Explain what happens when the `step` procedure is called.

Exercise 6.6.2: Wall Following Events NetLogo Model

Try out the Wall Following Events model in NetLogo.

Wall Following Events	http://files.bookboon.com/ai/Wall-Following-Events.html

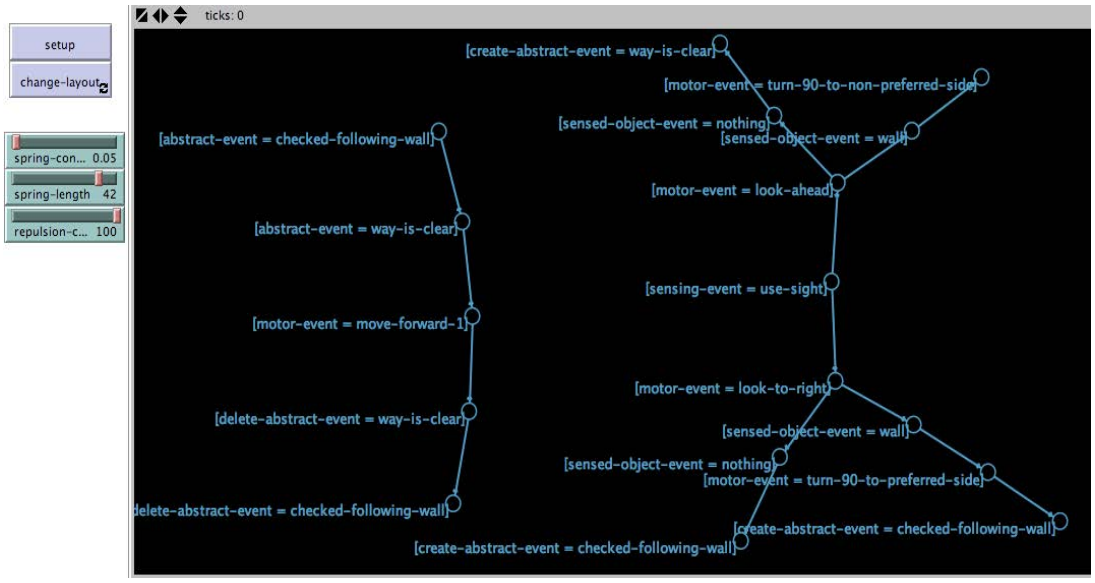


Figure 6.6.2. Screenshot of the Interface for the Wall Following Events model after the setup button has been pressed followed by the change-layout button with the slider values as shown in the image.

WHAT IS IT?

This model visualises a small set of events that an agent can follow in order to perform a modified type of wall following behaviour where sensing, thinking and acting are all done concurrently in no particular order (see the Wall Following Example 2 model for further explanation). The events are shown using an event map representation, where events are linked to other events on separate streams in an ordered sequence. The idea is that an agent processes events simultaneously on separate streams. The approach is similar to the approach adopted for Event Stream Processing (ESP).

WHAT IS ITS PURPOSE?

The purpose of this model is to show how to visualise a series of events using an event map.

HOW IT WORKS

The model uses turtle agents to represent the states in the event map, and uses links agents to represent the paths between states. States own three variables:

- `depth`: The depth in the event map tree.
- `stream`: The stream name (where a stream consists of a sequence of sensory or motor events).
- `event`: The event – either sensory or motor.

A spring layout is used to visualise the event map.

HOW TO USE IT

Press the `setup` button first. This will often produce a cluttered layout. To unfold the clutter, press the `change-layout` button, and then dynamically change the values in the sliders that control the layout. One effective technique is to reduce the value of the `spring-length` slider to 0, then slowly increase it back up again until the desired length is achieved.

THE INTERFACE

The model's `Interface` buttons are defined as follows:

- `setup`: This will clear the environment and variables and (re)-load the event map. Normally, this will appear in a cluttered form and the `change-layout` button needs to be pressed subsequently.
- `change-layout`: This can be used to clear some of the clutter by changing the values in the three sliders.

The model's `Interface` sliders are defined as follows:

- `spring-constant`: This is a value used by the `layout-spring` command. Changing it will usually not affect the visualisation of the event map much.
- `sprint-length`: This modifies the length of the paths between the states of the event map network.
- `repulsion-constant`: This controls how much each of the states repulse each other.

THINGS TO NOTICE

Notice how the `repulsion-constant` slider can be used to "repel" the states away from each other (for larger values) and "attract" the states towards each other (for smaller values).

Notice that the clutter in the network layout can often be removed by setting the value of the `spring-length` slider to zero and then increasing it afterwards.

THINGS TO TRY

Try altering the values of the sliders to see what effect this has on the layout.

EXTENDING THE MODEL

Combine this model with the Wall Following Example 2 model. Then animate the environment and event map at the same time for one of the agents in the environment that is moving around following the walls.

NETLOGO FEATURES

The model uses the `layout-spring` command for modifying the layout of the network of states (turtle agents) and paths (link agents).

RELATED MODELS

See the Wall Following Example 2 model, the Central Park Events model and the Knowledge Representation model. For the Wall Following Example 2 model, see Exercise 6.7.6 below.

6.7 Boids

Exercises 6.7.1 to 6.7.6:

Try out the following models that implement various steering behaviours for boids:

6.7.1 Crowd Path Following;
6.7.2 Flocking with Obstacles;
6.7.3 Follow and Avoid;
6.7.4 Obstacle Avoidance 1;
6.7.5 Obstacle Avoidance 2;
6.7.6 Wall Following Example 2.

Note that to enhance the boids simulation and to be fully compatible with Craig Reynold's boids implementations for these models, this requires the addition of gradual acceleration and deceleration. A faithful implementation should also include some form of steering force implemented using point mass approximation where each boid has a mass and works in relation to forces. However, these NetLogo implementations show how an approximation to Reynolds approach can be achieved relatively easy, and in many cases, the resultant behaviour of the boids is as desired.

Exercise 6.7.1: Crowd Path Following NetLogo Model

Crowd Path Following	http://files.bookboon.com/ai/Crowd-Path-Following.html

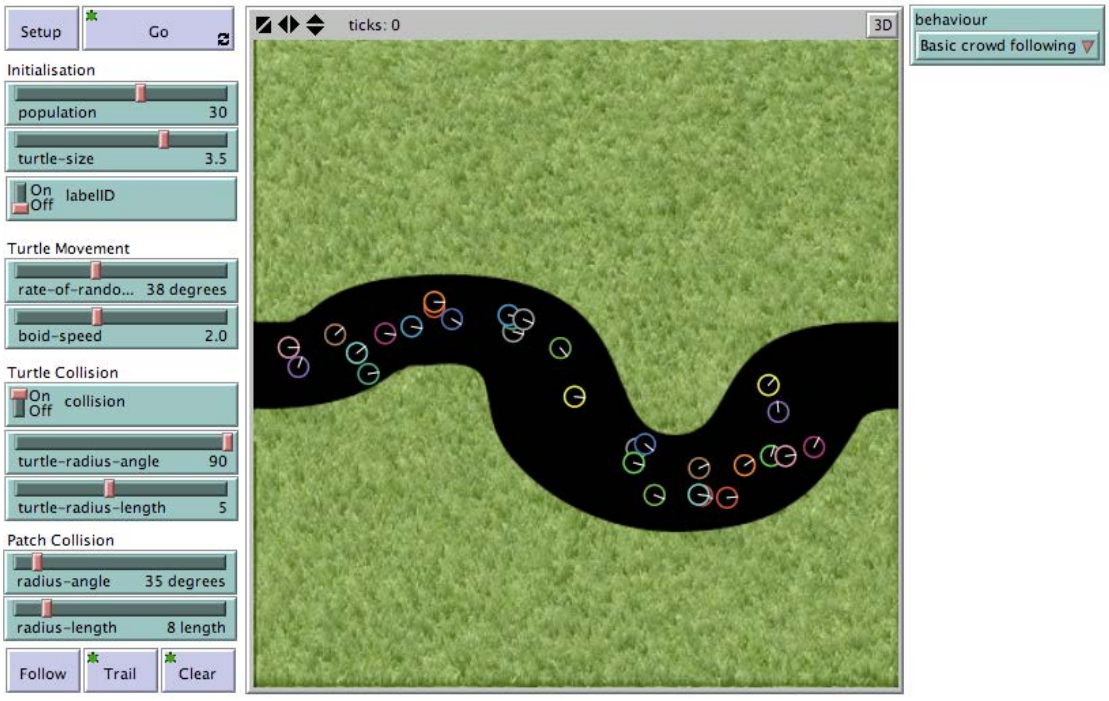


Figure 6.7.1. Screenshot of the Interface for the Crowd Path Following model after the setup button has been pressed followed by the go button when the behaviour has been set to "Basic crowd following".

WHAT IS IT?

This model is an attempt to recreate boids (see Craig Reynold's work) that use the crowd path following steering behaviour.

HOW IT WORKS

Each turtle (boid) has two levels of vision – a forward facing cone and a boundary radius. The Cone looks out for coloured patches (i.e. *not* black) in front and if a patch is detected it runs a simple routine to navigate the path. The radius detection simply looks out for other turtles at a given distance threshold. When a turtle bumps into another turtle it does not move forward but instead turns right; this causes a slight pause in its forward motion which prevents it from going through the colliding turtle. As an addition, to highlight the collision effect, each turtle is given a variable speed based on its `who` number. This makes the first turtle the slowest and the last the fastest, making the turtles more likely to bump into one another. The faster turtles try to get past the slower ones.

HOW TO USE IT

Decide on the size of the population then press the `setup` button, then press the `go` button. The slider bars allow various parameters to be altered. To draw the boid trails, press the `trail` button; to clear them, press the `clear` button. To follow one particular boid, press the `follow` button.

THE INTERFACE

The model's `Interface` buttons are defined as follows:

- `Setup`: This sets up the environment with a black path across the middle, and the rest shown as a green texture to represent the surrounding grass. A number of boids (determined by the `population` variable) are created and placed at a random location within the black path.
- `Go`: The boids start moving along the path and after a while, most of them end up heading in the same direction.
- `Follow`: The simulation follows the path of one particular boid.
- `Trail`: This draws the paths made by all boids.
- `Clear`: This clears the trails.

The model's `Interface` sliders, switches and chooser are defined as follows:

1. For initialising the simulation:

- `population`: This specifies the number of boids.
- `turtle-size`: This specifies the size of the boids.
- `labelID`: If this is set to `On`, then the boids are labelled with their `who` numbers.

2. For controlling the movement of the boids:

- `rate-of-random-turn`: This controls how much the wandering boid turns each time tick. The boid has a tendency to head in a right turning direction as the rate of random turn to the right (as specified by the slider) is twice that of the rate of random turn to the left.
- `boid-speed`: This controls the speed of the boid i.e. how much it moves forward each tick.

3. For specifying parameters that control boid collision avoidance behaviour:

- `collision`: This turns on collision detection and avoidance if set to `On`.
- `turtle-radius-angle`: This defines the radius angle of the boid's vision cone in relation to other boids.
- `turtle-radius-length`: This defines the radius length of the boid's vision cone in relation to other boids.

4. For specifying parameters that control patch collision avoidance behaviour (where the patches are the boundaries of the path):

- `radius-angle`: This defines the radius angle of the boid's vision cone in relation to the edges of the path.
- `radius-length`: This defines the radius length of the boid's vision cone in relation to the edges of the path.
- `behaviour`: This defines the boid's behaviour. It can be set to the following values:

"Basic crowd following": This is very basic crowd path following behaviour without collision avoidance.
"With collision avoidance": This combines crowd path following behaviour with collision avoidance if the collision slider is set to On.

THINGS TO NOTICE

Notice how the turtles come close to the sides but manage to avoid them. How is this achieved, and what is the mechanism for keeping them in track?

Notice how the speed variances are produced. Each turtle has a unique speed setting.

Notice that the colours are randomised based on the chromatic point at which it was created. This allows a random looking range of colours to be produced while avoiding very dark and black colours, which would clash with the path.

Notice the Follow turtle button, and how a single turtle can be analysed more closely; the 3D button also is of interest here.

Notice that the turtles only appear on the black patches. How is this done?

Notice that when the boid-speed variable is set to a large number (5.0, say), some of the boids start moving off the path and end up going across the surrounding green grass. Why is this? Is there some way we can code the turtle agent's movement to prevent this from happening?

THINGS TO TRY

Try altering the random rate of turn. How does this affect the turtles' ability to follow the path? Why does it change?

Change the radius-angle and radius-length. What does this do to the model? Why does this change?

Change the turtle collision parameters. How does this change the model? What are the optimum settings for accurate collision detection?

Try altering the turtle speed, why does this change the ability to stay in the path?

Click on the trail button (here is the real reason for variable colours).

EXTENDING THE MODEL

Adding a `turtles-own` variable for speed, we can make the turtles accelerate and decelerate. This will enable a colliding turtle to simply slow down and alter direction to avoid a collision.

NETLOGO FEATURES

The `who`, `myself`, `in-cone`, `in-radius`, `heading`, `sprout` and turtle editor features are used in this model.

RELATED MODELS

See the following models: Crowd Path Following, Flocking With Obstacles, Follow and Avoid, Obstacle Avoidance 1, Obstacle Avoidance 2, Vision Cone Example 2, Wall Following Example 2. These are basic implementations of various Craig Reynold's steering behaviours for boids.

Another boid related model is the Biology/Flocking model in the Models Library.

Exercise 6.7.2: Flocking with Obstacles NetLogo Model

Flocking With Obstacles	http://files.bookboon.com/ai/Flocking-With-Obstacles.html

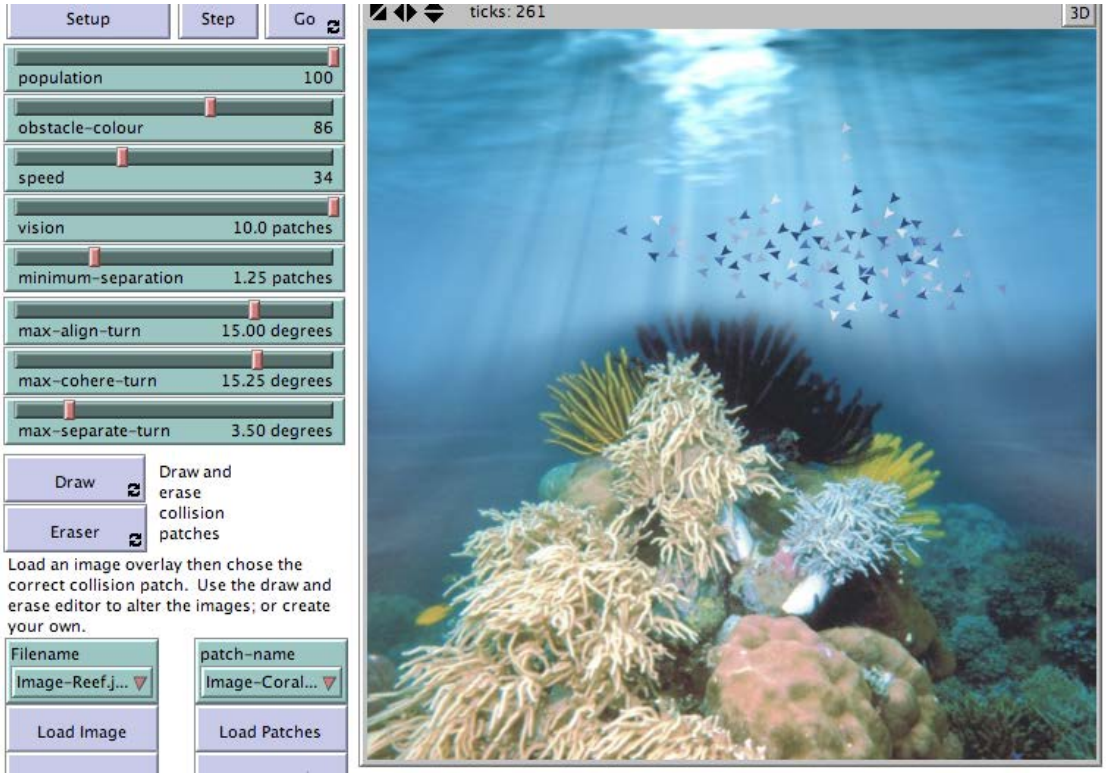


Figure 6.7.2. Screenshot of the Interface for the Flocking with Obstacles model after the setup button has been pressed followed by the go button.

WHAT IS IT?

This model is an attempt to recreate the flocking behaviour of Craig Reynold's boids. It was created by Uri Wilensky and modified by Thomas Christy and William John Teahan. Most of the `Information` in the model is similar to that provided by Uri Wilensky, with some modifications as included below.

THE INTERFACE

The buttons in the `Interface` are defined as follows:

- `Setup`: Clears all variables and creates the population of birds at random locations in the environment.
- `Step`: Runs a single tick of the simulation.
- `Go`: Runs the simulation continuously.
- `Draw`: This allows the user to draw collision patches to provide some obstacles for the birds to avoid hitting.
- `Erase`: This allows the user to erase the drawn obstacles.

- `Load Image`: This loads an image from the file named by the `Filename` slider into the background. Although this makes the simulation look more impressive, the bird agents cannot "see" this image. If you want to have the birds react to what appears in the image, you also need to load the patches for the same image using the `Load Patches` button. Note: Since the `Load Image` button will overwrite patches, make sure you synchronise both the `Filename` and `patch-name` (see below) so that they match, otherwise the birds will react to patches that can't be seen by us as the observers but are there in fact in the environment.
- `Load Patches`: This will load the patches for a given image specified by the `patch-name` slider so that the bird agents are able to "see" and react to them.

The sliders in the `Interface` are defined as follows:

- `population`: This is the number of bird agents created at the beginning.
- `obstacle-colour`: This is the colour used for drawing the obstacles using the `Draw` button.
- `speed`: This controls the speed of the bird agents.
- `vision`: This defines the radius of the bird agent's cone of vision. (The angle is 360 degrees, so the bird can "see" in all directions).
- `minimum-separation`: This defines the minimum separation for the flockmates before the flockmates will start to separate.
- `max-align-turn`, `max-cohere-turn`, `max-separate-turn`: These sliders control the maximum angle a bird can turn as a result of each rule – alignment, cohesion and separation, respectively.

The choosers in the `Interface` are defined as follows:

- `Filename`: This is the name of the image file which is loaded into the background when the `Load Image` button is pressed.
- `patch-name`: This is the name of the image file which is converted into patches in the environment when the `Load Patches` button is pressed.

THINGS TO NOTICE

Around obstacles, sometimes the flock splits in two to head around different sides of an obstacle, but then returns to a single flock. The same behaviour can be seen in Craig Reynold's simulations. What are the conditions that flock splitting seems to occur in? Does it sometimes occur that the flock remains split in two?

THINGS TO TRY

What happens if the `in-cone` command is used instead of the `in-radius` command? Does substantially reducing the vision cone angle affect the behaviour of the birds?

EXTENDING THE MODEL

Try adding your own images to the model. How is the behaviour of the turtle agents affected by the different environments?

Many of the birds get stuck inside or near to an obstacle. This detracts a little from the believability of the simulation. Is there some way this "bug" can be removed from the model?

NETLOGO FEATURES

The model uses the command `in-radius` to implement the boids' cone of vision.

Exercise 6.7.3: Follow and Avoid NetLogo Model

Follow and Avoid	http://files.bookboon.com/ai/Follow-And-Avoid.html

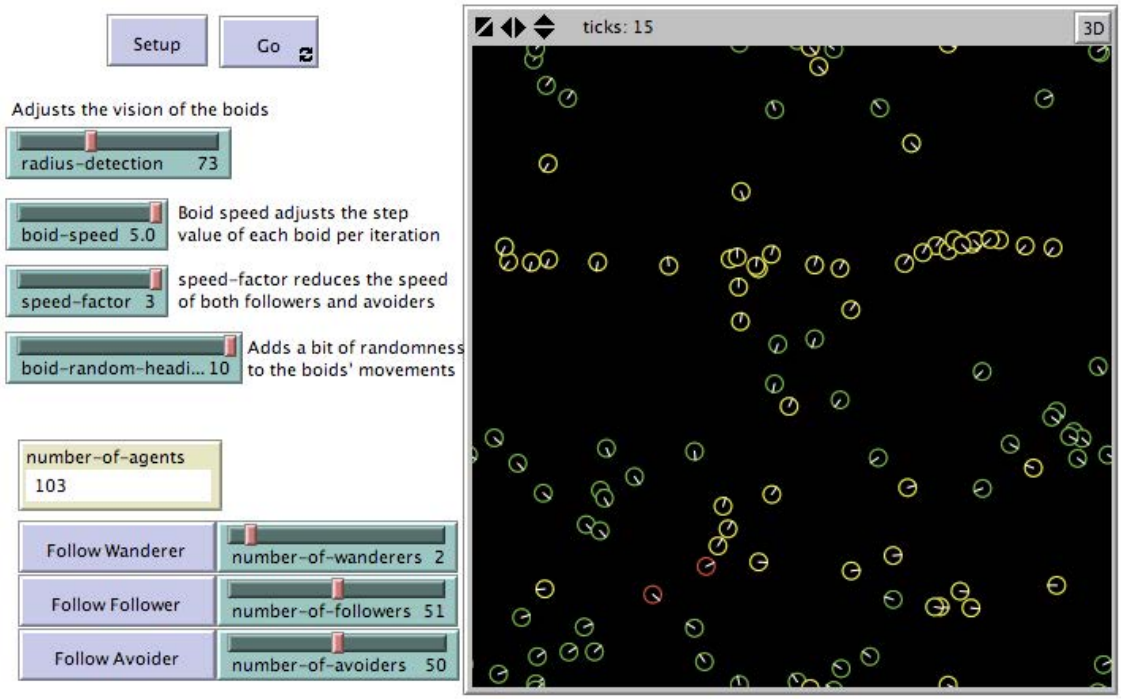


Figure 6.7.3. Screenshot of the Interface for the Follow and Avoid model after the setup button has been pressed followed by the go button.

WHAT IS IT?

This model attempts to simulate the seeking and fleeing steering behaviours for boids (as devised by Craig Reynolds).

HOW IT WORKS

It does this by generating random Wanderer turtles (boid) that simply wander. Followers (seeking), on detecting Wanderers within their radius, turn to face the nearest Wanderer and move forward. Avoiders (fleeing) do the same action but reverse their direction by 180 degrees, thus moving away from the nearest Wanderer.

HOW TO USE IT

Setup the initial world. Then use the speed bars during operation to adjust the speed of all the turtles.

THE INTERFACE

The buttons in the `Interface` are defined as follows:

- `Setup`: This clears the environment and creates new Wanderer, Follower and Avoider agents.
- `Go`: This starts the simulation.
- `Follow Wanderer`: This will follow one of the Wanderer agents.
- `Follow Follower`: This will follow one of the Follower agents.
- `Follow Avoider`: This will follow one of the Avoider agents.

The sliders and monitor in the `Interface` are defined as follows:

- `radius-detection`: This adjusts the vision of the boids.
- `boid-speed`: This adjusts the forward step value of each boid per iteration.
- `speed-factor`: This reduces the speed of both the Follower and Avoider agents.
- `boid-random-heading`: This adds a bit of randomness to the boids' movements.
- `number-of-agents`, `number-of-wanderers`, `number-of-followers`, `number-of-avoiders`: These are the number of agents in each respective category.

THINGS TO NOTICE

Adjusting the radius bar alters the distance at which both Followers and Avoiders can see. This alters the way they behave, such as the distance that the Avoider stays away from Wanderers.

The Avoider sometimes appears to move towards a Wanderer; this is an affect resulting from an environment that wraps around.

THINGS TO TRY

Try changing the number of Wanderers, Followers and Avoiders.

Try adjusting the scale bar to see what happens to the Follower.

Try changing the shape of the Avoider to be rotatable (if they are not already); this highlights the affects of the wrap around problem.

EXTENDING THE MODEL

The model could be extended to add gradual acceleration and deceleration. This would enhance the boids simulation.

NETLOGO FEATURES

The model uses the command `in-radius` to implement the boids' cone of vision.

RELATED MODELS

Another boid related model is the Biology/Flocking model.

Exercise 6.7.4: Obstacle Avoidance 1 NetLogo Model

Obstacle Avoidance 1	http://files.bookboon.com/ai/Obstacle-Avoidance-1.html

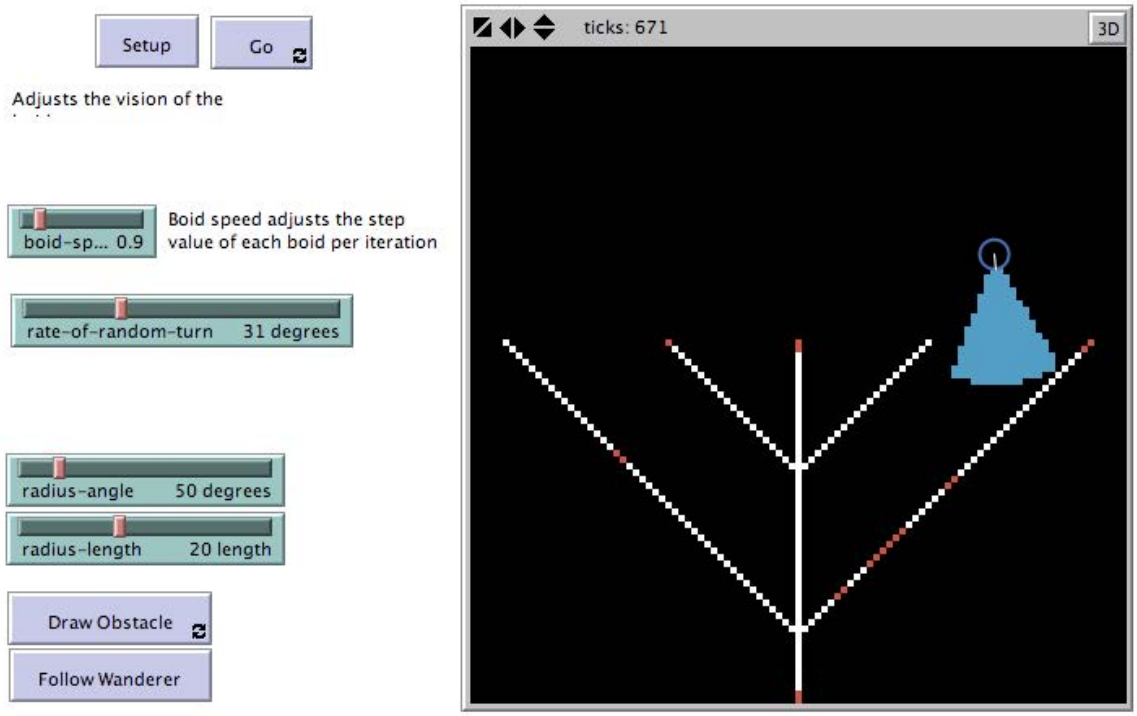


Figure 6.7.4. Screenshot of the Interface for the Obstacle Avoidance 1 model after the setup button has been pressed followed by the go button.

WHAT IS IT?

This model implements a boid (see Craig Reynold's work) that employs basic obstacle avoidance steering behaviour.

HOW IT WORKS

It does this by generating a wanderer turtle (boid) that simply wanders around randomly in the environment avoiding the obstacles that are drawn with white patches. The boid is implemented using NetLogo's `in-cone` command that implements a turtle with a cone of vision.

INTERFACE

The model's Interface buttons are defined as follows:

- `Setup`: This sets up the environment with a tree-like arrangement of obstacles drawn with white patches. One turtle agent (the wanderer boid) is created and placed at a random location.
- `Go`: The boid starts wandering around the environment avoiding obstacles.
- `Draw Obstacle`: The user can draw further obstacles in the environment. These are also coloured white.
- `Follow Wanderer`: This allows the perspective of the visualisation to be altered so that it is centred on the wanderer.

The model's Interface sliders are defined as follows:

- `boid-speed`: This controls the speed of the boid i.e. how much it moves forward each tick.
- `rate-of-random-turn`: This controls how much the wandering boid turns each time tick. The boid has a tendency to head in a right turning direction as the rate of random turn to the right (as specified by the slider) is twice that of the rate of random turn to the left.
- `radius-angle`: This defines the radius angle of the boid's vision cone.
- `radius-length`: This defines the radius length of the boid's vision cone.

HOW TO USE IT

Press the `Setup` button first, then press `Go`.

You can draw extra obstacles by pressing the `Draw Obstacle` button and by then holding down the mouse at the point where you want the obstacles to be drawn. You can change the frame of reference so that the visualisation is centred around where the boid currently is situated by pressing the `Follow Wanderer` button.

THINGS TO NOTICE

Setting the boid's vision cone `radius-length` or `radius-angle` will result in the boid not seeing the obstacles and as a result will run right over the top of them. Why? What needs to be fixed so that even without any sensing ability, it will still bounce off the obstacles?

Increasing the `radius-length` (while keeping the other variables the same) discernibly changes the behaviour of the boid. (Try doing this dynamically while moving the `radius-length` slider back and forth). Instead of covering most of the environment, when the radius length is large then the boid covers a much smaller more constrained area usually at the top of the environment. Sometimes it can get stuck, seemingly trapped in the same place.

THINGS TO TRY

Try adjusting the boid's speed, radius angle and radius length to see how this affects the boid's behaviour. Also try changing the `Interface Settings` to see if this has any affect.

Try adding obstacles to see how this affects the boid's ability to cover the entire environment. For example, add obstacles in the form of a maze. Try to create "black spots" where the boid never visits. Alternatively, try to trap the boid into a small area.

EXTENDING THE MODEL

The model could be extended to add gradual acceleration and deceleration. This would enhance the boids simulation.

NETLOGO FEATURES

The code uses the `in-cone` command to simulate the boid's cone of vision.

RELATED MODELS

See the following models: Crowd Path Following, Flocking With Obstacles, Follow and Avoid, Obstacle Avoidance 1, Obstacle Avoidance 2, Vision Cone Example 2, Wall Following Example 2. These are basic implementations of various Craig Reynold's steering behaviours for boids.

Another boid related model is the Biology/Flocking model in the Models Library.

Exercise 6.7.5: Obstacle Avoidance 2 NetLogo Model

Obstacle Avoidance 2	http://files.bookboon.com/ai/Obstacle-Avoidance-2.html

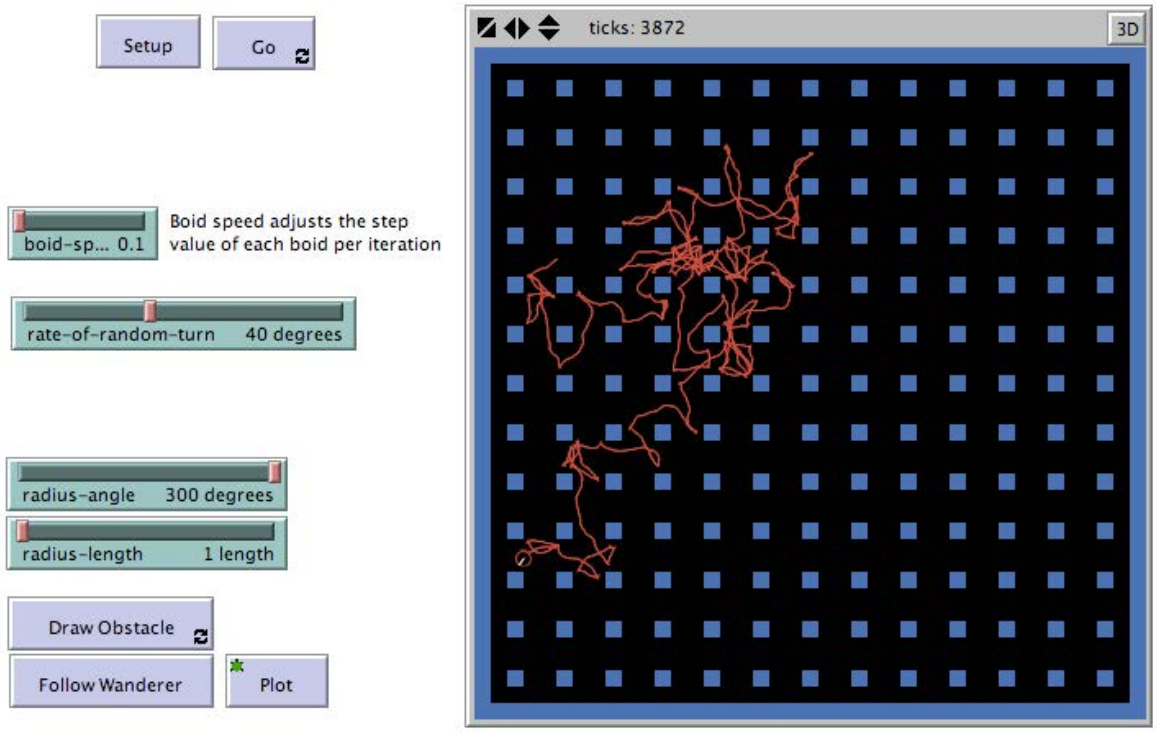


Figure 6.7.5. Screenshot of the Interface for the Obstacle Avoidance 2 model after the setup button has been pressed followed by the go and plot buttons.

WHAT IS IT?

This model implements a boid (see Craig Reynold's work) that employs basic obstacle avoidance steering behaviour.

HOW IT WORKS

It does this by generating a wanderer turtle (boid) that simple wanders around randomly in the environment avoiding the obstacles. The boid is implemented using NetLogo's `in-cone` command that implements a turtle with a cone of vision.

INTERFACE

The model's `Interface` buttons are defined as follows:

- `Setup`: This sets up the environment with a grid of obstacles and an outside border (drawn using blue patches). One turtle agent (the wanderer boid) is created and placed at a random location.
- `Go`: The boid starts wandering around the environment avoiding obstacles.
- `Draw Obstacle`: The user can draw further obstacles in the environment. These are coloured brown.
- `Follow Wanderer`: This allows the perspective of the visualisation to be altered so that it is centred on the wanderer.
- `Plot`: This instructs the wanderer turtle agent to put its pen down when wandering. Hence this will draw the path it has taken while wandering around.

The model's `Interface` sliders are defined as follows:

- `boid-speed`: This controls the speed of the boid i.e. how much it moves forward each tick.
- `rate-of-random-turn`: This controls how much the wandering boid turns each time tick. The boid has a tendency to head in a right turning direction as the rate of random turn to the right (as specified by the slider) is twice that of the rate of random turn to the left.
- `radius-angle`: This defines the radius angle of the boid's vision cone.
- `radius-length`: This defines the radius length of the boid's vision cone.

HOW TO USE IT

Press the `Setup` button first, then press `Go`. To see where the boid wanders, press `Plot`. You can draw extra obstacles by pressing the `Draw Obstacle` button and then holding down the mouse at the point where you want the obstacles to be drawn. You can change the frame of reference so that the visualisation is centred around where the boid currently is situated by pressing the `Follow Wanderer` button.

THINGS TO NOTICE

Setting the `boid-speed` to 0.1, `rate-of-random-turn` to 40, `radius-angle` to 300, `radius-length` to 1, and pressing the `Plot` button once, followed by moving the speed slider (just below the Information tab in the `Interface`) from `"normal speed"` to `"faster"` will result in the boid rapidly covering the entire environment while reliably avoiding the blue obstacles.

Increasing the `radius-length` to 5 (while keeping the other variables the same) discernibly changes the behaviour of the boid. Instead of covering most of the environment, the boid covers a rectangular path of width 4 to 5 around the outside of the environment but indented by about 3 to 4 patches in from the outer boundary. The boid seems to refrain from going inside of the rectangular path. Sometimes the boid can get stuck spinning around one of the obstacles.

THINGS TO TRY

Try adjusting the boid's speed, radius angle and radius length to see how this affects the boid's behaviour. Also try changing the `Interface Settings` to see if this has any affect.

Try adding obstacles to see how this affects the boid's ability to cover the entire environment. For example, add obstacles in the form of a maze. Try to create "black spots" where the boid never visits. Alternatively, try to trap the boid into a small area.

EXTENDING THE MODEL

The model could be extended to add gradual acceleration and deceleration. This would enhance the simulation of the boids model.

NETLOGO FEATURES

The code uses the `in-cone` command to simulate the boid's cone of vision.

RELATED MODELS

See the following models: Crowd Path Following, Flocking With Obstacles, Follow and Avoid, Obstacle Avoidance 1, Obstacle Avoidance 2, Vision Cone Example 2, Wall Following Example 2. These are basic implementations of various Craig Reynold's steering behaviours for boids.

Another boid related model is the Biology/Flocking model in the Models Library.

Exercise 6.7.6: Wall Following Example 2 NetLogo Model

Wall Following Example 2	In NetLogo's Models Library: Code Examples > Wall Following Example; see modified code at: http://files.bookboon.com/ai/Wall-Following-Example-2.html

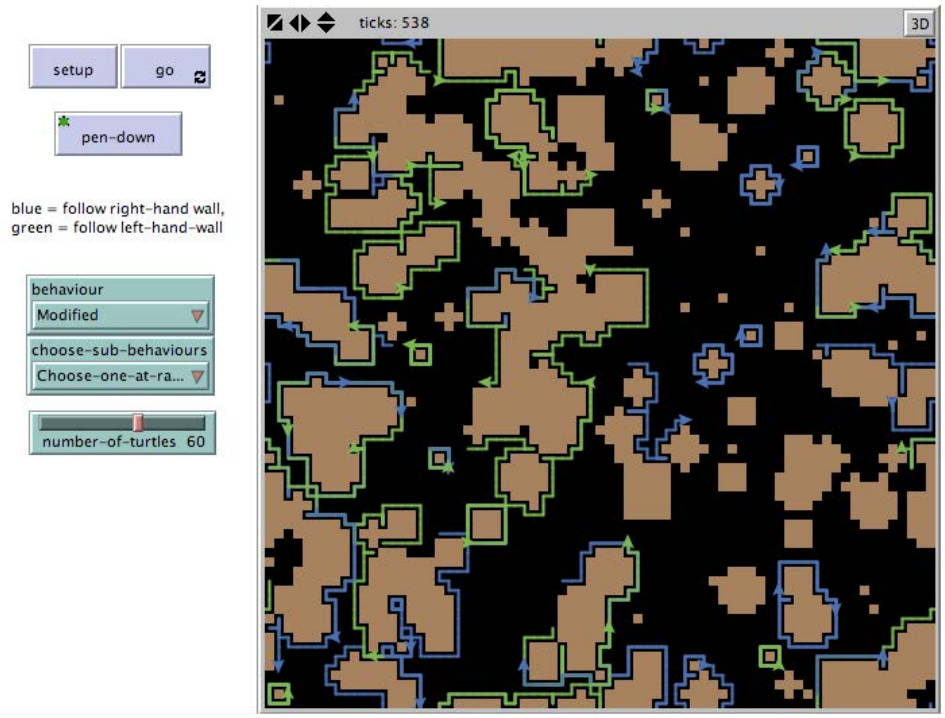


Figure 6.7.6. Screenshot of the Interface for the Wall Following Example 2 model after the setup button has been pressed followed by the go and pen-down buttons.

WHAT IS IT?

The turtles in this example follow walls made out of colored patches. The blue turtles try to keep the wall on their right; the green turtles keep the wall on their left. Hence, the blue turtles end up heading in a clockwise direction, and the blue turtles end up in an anti-clockwise direction.

(Only the `Information` in the model that is different to the Wall Following Example model provided by Uri Wilensky is included below.)

HOW IT WORKS

The model implements the standard unmodified behaviour as provided by Uri Wilensky's original model, but also provides an alternative modified behaviour where the actions have been split into three independent parts as follows:

1. `walk-modified-1`: The turtle turns right if necessary.
2. `walk-modified-2`: The turtle turns left if necessary.
3. `walk-modified-3`: The turtle moves forward.

These modified sub-behaviours are executed concurrently in no particular order. The purpose is to show a "Sense & Think & Act" type behaviour where sensing, thinking and acting are done concurrently rather then a "Sense – Think – Act" type behaviour where sensing, thinking and acting are done one after the other.

THE INTERFACE

The `Interface` buttons are defined as follows:

- `setup`: This initialises the environment with randomly shaped blocks, and places the turtle agents at random locations in the intervening black spaces.
- `go`: This runs the model.
- `pen-down`: This gets the turtle agents to put their pen downs, thereby drawing the paths they take as they move around.

The `Interface` choosers and sliders are defined as follows:

- `behaviour`: Setting this to `Standard` sets the behaviour to that devised in Uri Wilensky's original model. Setting this to `Modified` sets the behaviour to that explained above in the `"How it works"` section.
- `choose-sub-behaviours`: When the behaviour variable is set to `Modified`, setting this to `choose-all-in-random-order`, then the three modified behaviours (see above) will be executed together but in no particular order. Otherwise, just one of them will be randomly chosen to be executed.
- `number-of-turtles`: This is the number of turtle agents that are created at setup.

THINGS TO NOTICE

Changing the behaviour variable from `Standard` to `Modified` in the `Interface` does not seem to change what the turtles seem to be doing. However, when the behaviour is set to Modified, see if you can notice slight variations in behaviour to the `Standard` behaviour, especially when the turtle agent ends up in a cul-de-sac or there is a narrow alley-way between two buildings. In the latter case, the `Modified` behaviour results in the turtles having the ability to explore further than their `Standard` counterparts.

When the behaviour variable is set to `Modified`, note that the speed of the turtles is much faster when the `choose-sub-behaviours` variable is set to `choose-all-in-random-order` rather than `choose-one-at-random`. Why?

Exercise 6.7.7:

Look at the code in the NetLogo models described in Exercises 6.7.1 to 6.7.6. How is each of the models performing obstacle avoidance? (i.e. Locate and explain the piece of code in the models that performs obstacle avoidance). Or are there any models not performing any obstacle avoidance at all?

Exercise 6.7.7:

For the Follow and Avoid model described in Exercise 6.7.3, the behaviour of the three breeds of agents is defined by the `wanderers`, `followers` and `avoiders` procedures. Explain how these three procedures work. Which implements boid-like behaviour and which doesn't?

Exercise 6.7.8:

For the Flocking with Obstacles model described in Exercise 6.7.2, explain how the code works to produce flocking. In particular, explain the behaviour that is being modelled by the following procedures in the model:

- `separate`;
- `align`;
- `cohere`.

Also detail any emergent phenomena that you witness (an obvious example is the flocking, but there are others when various variables are changed using the `Interface` sliders).

7 Communication

7.1 Communication, Information and Language

Exercise 7.1.1:

Compare the different meanings of the following terms by finding out how they are being defined in the literature:

- communication;
- information;
- language.

7.2 The diversity of human language

Exercise 7.2.1:

How many human languages are being used in the world today? Which continent has the greatest number of languages in use?

7.3 Communication via communities of agents

Exercise 7.3.1: Language Change NetLogo Model

Try out the Language Change model in NetLogo:

Language Change	In NetLogo's Models Library: Social Science > Language Change http://ccl.northwestern.edu/netlogo/models/LanguageChange

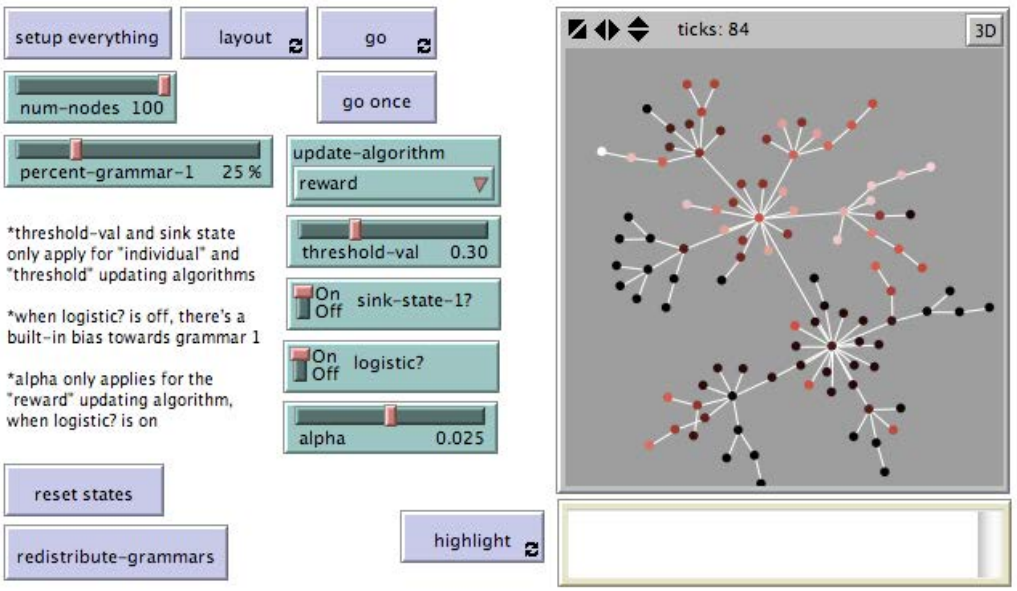


Figure 7.3.1. Screenshot of the Interface for the Language Change model after the setup everything button has been pressed followed by the layout button to unclutter the network, and then the go button, with the slider and switch values as shown in the image.

Read the `What is it?` and `Things to Notice` sections in the `Information` for the model. Then try out the exercises suggested in the `Things to try` and `Extending the Model` sections. In particular, the `Information` in the model states that language users tend to arrive at using just one grammar but in some cases, they may not all converge to the same grammar. Try to find out what conditions cause the language users to converge to a single grammar, and what conditions result in multiple grammars being used.

Exercise 7.3.2:

What happens in the Language Change model when the agents communicate with each other? In particular, explain how the following procedures work in the model:

- `communicate-via`;
- `speak`;
- `listen`.

7.4 Communicating Behaviour

Exercise 7.4.1: Communication-T-T Example 2 NetLogo Model

Try out the Communication-T-T Example 2 model in NetLogo:

Communication-T-T Example 2	For original model, see NetLogo's Models Library: Code Examples > Communication-T-T Example; for modified model described here: http://files.bookboon.com/ai/Communication-T-T-Example-2.nlogo

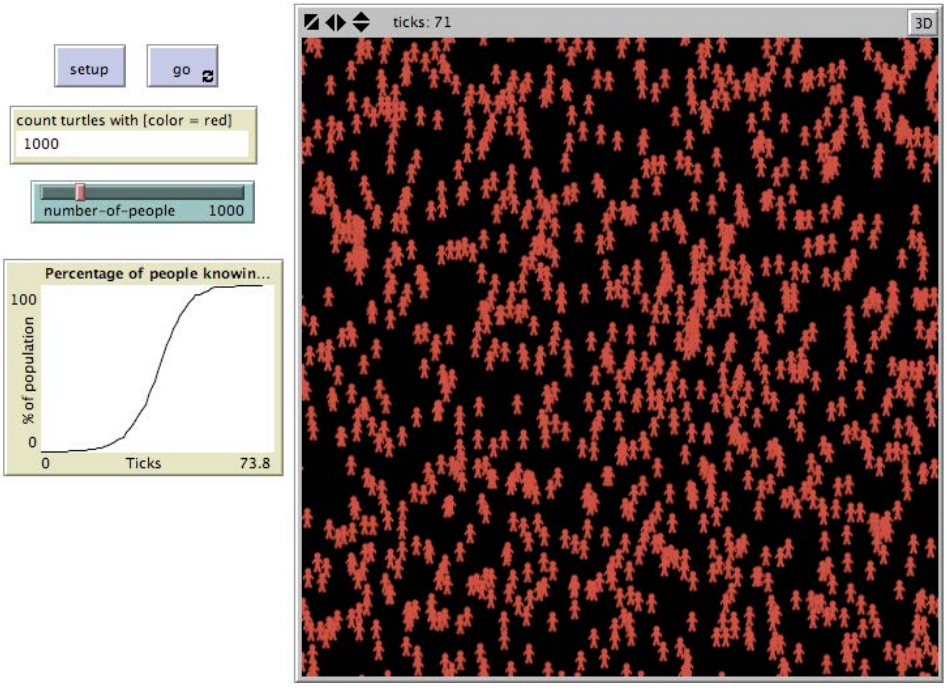


Figure 7.4.1. Screenshot of the Interface for the Communication T-T Example 2 model after the setup button has been pressed followed by the go button, with the number of people set at 1000.

Read the `What is it?` section in the `Information` for the original model. Then try out the modified model, whose modified `Information` is detailed below.

HOW TO USE IT

Select the number of people you want in the simulation using the `number-of-people` slider, press the `setup` button, then press the `go` button.

THE INTERFACE

The buttons in the `Interface` are defined as follows:

- `setup`: This clears the variables, and resets the simulation creating persons at random locations whose number is specified by the `number-of-people` slider.
- `go`: This starts the simulation.

The monitor, slider and plot in the `Interface` are defined as follows:

- `count turtles with [color = red]`: This counts the number of turtles who know the message.
- `number-of-people`: This is the number of persons who will be created next time the `setup` button is pressed.
- `Percentage of people knowing message`: This plots the percentage of the people who know the message against the number of ticks.

THINGS TO NOTICE

Notice that when the number of people is set to maximum, the message literally spreads like fire.

THINGS TO TRY

Try running the simulation with different number of people. Notice what happens in the plot when the number of people is small rather than large. It takes a lot longer to spread the message when the population size is small. Why is this (since the number of people do not know the message at the beginning is a lot less)?

EXTENDING THE MODEL

Try changing the model so multiple messages are being spread. Alternatively, try having two different breeds of people – those that spread messages, and those that don't.

Try a further variation where there is a random probability (as defined by a slider value) that the message may mutate when it is passed on. Set various conditions, such as how many times does the message need to be heard from different people, that determine when the message is finally "believed". How long does it now take before the message is spread?

7.5 The Small World Phenomenon and Dijkstra's algorithm

Exercise 7.5.1: Being Kevin Bacon NetLogo Model

Try out the Being Kevin Bacon model in NetLogo:

Being Kevin Bacon	http://files.bookboon.com/ai/Being-Kevin-Bacon.nlogo

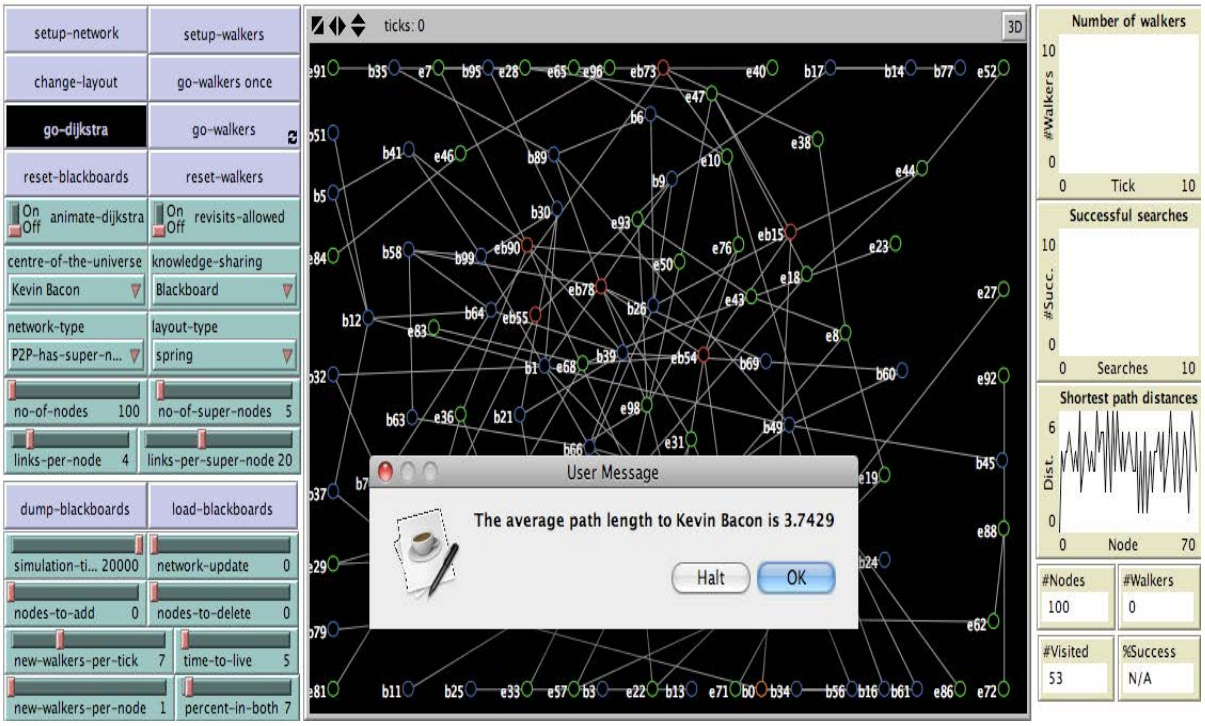


Figure 7.5.1. Screenshot of the Interface for the Being Kevin Bacon model after the setup-network button has been pressed followed by the go-dijkstra button to execute Dijsktra's algorithm.

WHAT IS IT?

This model applies various algorithms to the problem of searching for a specific goal node in a network. The problem is that you do not know where the node is, or how the network is configured. To solve this problem, search needs to be employed to try out the different paths that might lead to the goal node. This model implements some multi-agent based solutions where knowledge is shared between agents in a dynamic way. These can be applied to this problem separately to see how they perform against each other.

The model uses "Kevin Bacon" in its title as Kevin Bacon has been chosen as the "centre" of the Hollywood Universe based on the catch-phrase "degrees of separation from Kevin Bacon". This catch-phrase is based on the related mathematical pastime of measuring the degrees of separation of mathematicians from Paul Erdös, a well-known Hungarian mathematician who was prolific at publishing papers. Paul Erdös has been chosen as the focal point for measuring how well connected a person is amongst the mathematical community. The degrees of separation measure is based on whether as person has published a paper with Paul Erdös (which equates to an Erdös number of 1), or if not then published a paper with someone who has published a paper with Paul Erdös (an Erdös number of 2), or if not then published a paper with someone who has published a paper with someone who has published a paper with Paul Erdös (an Erdös number of 3), and so on.

The model allows the user to select either Paul Erdös or Kevin Bacon as the "centre of the universe". Then it can measure the average degree of separation using Dijkstra's algorithm, or it can search for the node associated with these persons using various distributed algorithms for sharing of knowledge between agents, such as word-of-mouth.

WHAT IS ITS PURPOSE?

The purpose of this model is to demonstrate some important concepts in relation to communication amongst agents in a network such as the small world phenomenon, degrees of separation, super-nodes in peer to peer networks and communication via word-of-mouth.

HOW IT WORKS

The model implements the search algorithms in a novel way by making use of an agent-oriented approach which is relatively easy to implement in NetLogo. The model adopts a purely agent-oriented solution where the information is distributed amongst NetLogo agents. Unlike the traditional search algorithms (such as uninformed searches like Breadth-first Search, and Depth-first Search, or the informed searches such as Greedy Best-first Search and A* that are implemented in the Searching for Kevin Bacon model), these agents conduct the search by sharing knowledge with other agents that are concurrently conducting the search. In the word-of-mouth knowledge sharing method, if agents have found the goal, then they share their knowledge with other agents they meet while returning to their start destination. With the blackboard method, the agents share their knowledge by writing what they know into communal "blackboards" that are found at each node. This can then subsequently be read by agents who visit the node latter on. Two further knowledge sharing methods combine these two approaches – word-of-mouth and blackboard – in slightly different ways to create two new hybrid knowledge sharing schemes.

A breed of `walker` turtle agents is used for the agents that move throughout the maze trying to get to the goal. These searcher agents maintain information about the current state of the search such as the path they have taken. Each searcher agent expands the search by following the outgoing paths that lead out of the current node that were suggested as a result of the method of knowledge sharing they adopt.

A breed of `dwalker` agents are used to walk the network during the execution of Dijkstra's algorithm. This can be used to calculate the degree of separation for every node in the network from the goal node.

HOW TO USE IT

To initialise the search and create a random network (whose type is specified by the `network-type` chooser), press the `setup-network` button in the `Interface`. To setup a `walker` agent to search the network, first press the `reset-blackboards` and `reset-walkers` buttons to clear blackboards and remove any existing walkers, then press the `setup-walkers` button to create new ones. Several `walker` turtle agents (whose number is specified by the `new-walkers-per-tick` slider) will then be created and randomly located through the network. The `setup-walkers` button can be pressed multiple times if you wish more walkers to start the search.

To start the simulation, either press the `go-walkers once` button to make the search proceed one step at a time, or press the `go-walkers` button to make the search proceed continuously. The walkers will continue walking the network until they find the goal node or they expire without success if they have exceeded the `time-to-live` slider value. The links that the `walker` agents traverse across are briefly thickened to show the paths the agents are taking as they proceed in their search.

To plot the shortest path distances and find the average degree of separation from the goal node using Dijkstra's algorithm, press the `go-dijkstra` button.

THE INTERFACE

The model's `Interface` buttons are defined as follows:

- `setup-network`: This will clear the environment and all variables, and create a random network, whose type is specified by the `network-type` chooser. Its layout is specified by the `layout-type` chooser. The four sliders `no-of-nodes`, `no-of-super-nodes`, `links-per-node` and `links-per-super-node` control the number of nodes and super-nodes, and the number of links between them.
- `change-layout`: If the layout is of type `spring`, then this may help to remove some of the clutter.
- `go-dijkstra`: This plots the shortest path distances and find the average degree of separation from the goal node using Dijkstra's algorithm. The goal node is specified using the `centre-of-the-universe` chooser.
- `reset-blackboards`: This resets the blackboards that are associated with each node in the network. This is used by the `Blackboard` knowledge sharing mechanism.
- `setup-walkers`: This creates new walkers as specified by the number in the slider `new-walkers-per-tick`. These walkers will then start walking around the network trying to find the goal node.
- `go-walkers-once`: This will make the simulation proceed one step at a time.
- `go-walkers-forever`: This will make the simulation proceed continuously until it has reached the goal node or it gets stuck.
- `reset-walkers`: This kills off all existing searchers and creates new ones to restart the search as specified by the number in the slider `new-walkers-per-tick`.
- `dump-blackboards`: This will dump the contents of all the blackboards in the network (this can be used for debugging purposes).
- `load-blackboards`: This pre-loads the blackboards for each node using the shortest paths found using the Dijkstra algorithm. (This will improve the performance of the `Blackboard` knowledge sharing mechanism).

The model's `Interface` choosers, sliders, and switches are defined as follows:

- `animate-dijkstra`: This will animate the execution of Dijkstra's algorithm (that runs when the `go-dijkstra` button is pressed).
- `revisits-allowed`: If set to `On`, this will allow the agents to revisit already visited nodes.
- `centre-of-the-universe`: This sets the goal node that the agents are trying to find. It can be set to either `Paul Erdos` or `Kevin Bacon`. (These nodes are randomly created when the network is created).
- `knowledge-sharing`: This specifies the type of knowledge sharing the agents employ while walking around the network. The types of knowledge-sharing is as follows:
  `"None"`: The agents do not exchange any knowledge with other agents. As a result, the agents end up randomly walking around the network.
  `"Word-of-mouth"`: If the agents have found the goal, then they share their knowledge with other agents they meet while returning to their start destination.
  `"Blackboard"`: The agents share their knowledge by writing what they know into communal "blackboards" that are found at each node. This can then subsequently be read by agents who visit the node latter on.
  `"Combined 1"` and `"Combined 2"`: These combine the knowledge of the word-of-mouth and blackboard approaches in slightly different ways to create two new hybrid knowledge sharing schemes. The first scheme checks the blackboard first, then defaults to the word-of-mouth scheme if no knowledge exists in the blackboard. The second scheme does the reverse – it will apply word-of-mouth knowledge first before defaulting to the blackboard knowledge.
- `network-type`: This selects the type of network that is created when the `setup-network` button is pressed. The type of networks are as follows:
  `"P2P-no-super-nodes"`: This simulates a peer-to-peer network with no super-nodes.
  `"P2P-has-super-nodes"`: This simulates a peer-to-peer network with super-nodes.
  `"P2P-random-single-link"`: This simulates a peer-to-peer network where each node has only one link with another random node.
  `"P2P-incremental"`: This creates the simulated peer-to-peer network by incrementally building the links to other random nodes one at a time.
  `"P2P-incremental-1"`: This creates the simulated peer-to-peer network by incrementally building links to other random nodes.
  `"Star-central-hub"`: This creates a network with one node (the `Kevin Bacon` node; i.e. the goal node) as the central hub and all other nodes linked to it and not to any other node.
  `"Hierarchical"`: This creates a tree network with the `Kevin Bacon` goal node at the root.
- `layout-type`: This specifies how the network should be laid out when it is visualised.
- `no-of-nodes`: This is the number of nodes to place in the network.

- `no-of-super-nodes`: This is the number of super-nodes to place in the network. (These are nodes that usually will have significantly more links than standard nodes as specified by `links-per-super-node` slider).
- `links-per-node`: This specifies the maximum number of links a standard node will have. The actual number chosen for a particular node will be a random number between 1 and this number.
- `links-per-super-node`: This specifies the maximum number of links a super-node will have. The actual number chosen for a particular super-node will be a random number between 1 and this number.
- `simulation-ticks`: This specifies how long the simulation should run for.
- `network-update`: If non-zero, this will cause the network to be updated at an interval according to a random number from 0 up to the value of this slider.
- `nodes-to-add`: The number of nodes specified by this slider will be added to the network when the next update occurs according to the `network-update` slider.
- `nodes-to-delete`: The number of nodes specified by this slider will be deleted from the network when the next update occurs according to the `network-update` slider.
- `new-walkers-per-tick`: This sets the number of `walker` agents that are added each tick of the simulation.
- `time-to-live`: This sets how long the `walker` agents should remain active if they have not found the goal node before being killed off.

- `new-walkers-per-node`: This sets the number of `walkers` that are created to continue the search in multiple different directions when each `walker` visits each node. Setting this number higher than 1 will quickly flood the network in most circumstances.
- `percent-in-both`: When the network is created, a number of nodes will be created that are connected to the `Paul Erdos` node (i.e. a path exists that will eventually lead to Paul Erdos), and a number will be created that are connected to `Kevin Bacon`. This slider controls how many of the nodes will be doubly connected (i.e. paths exist from the node that will lead to both the Paul Erdos and Kevin Bacon nodes).

The model's `Interface` plots monitors (shown on the right) are defined as follows:

- `Number of walkers`: This plot shows the number of `walkers` per tick.
- `Successful searches`: This plot shows the number of successful searches (i.e. `walkers` that have found the goal node) per tick.
- `Shortest path distances`: This plot is plotted when the `go-dijkstra` button is pressed. It plots the shortest path distances from each node to the goal node.
- `#Nodes`: This monitor reports the number of nodes in the network. It will change if the network is updated when the `network-update`, `nodes-to-add` and `nodes-to-delete` sliders are all non-zero.
- `#Walkers`: This monitor reports the number of active `walker` agents in the network.
- `#Visited`: This monitor reports the number of nodes visited by the `walker` agents.
- `%Success`: This monitor reports the percentage of `walker` agents that have been successful in reaching the goal node.

THINGS TO NOTICE

Notice how well the agents perform at finding the goal node when the knowledge sharing method is set to `None`. Compare this with the other knowledge sharing methods.

Notice that the blackboard knowledge sharing mechanism usually outperforms the word of mouth mechanism in terms of the percentage of successful searches.

Notice that pre-loading the blackboards with shortest path information gained from execution of Dijkstra's algorithm can significantly improve the search.

Notice that when there is a learning effect due to the knowledge sharing mechanism, the `Successful Searches` plot will start to curve upwards rather then continue as a straight line. Why is this? What settings and type of networks does this occur for? Can you get the curve to bend downwards instead of upwards? (Hint: Try changing the value of one or two of the sliders mid-stream).

Notice how the effectiveness of the search deteriorates as the network changes when nodes are either added or deleted. Which type of knowledge sharing method seems to cope best with a dynamic network?

Notice the effect of turning the flag `allow-revisited-states?` `On` and `Off`. Why is turning it `Off` so effective at dramatically reducing the search for the different search behaviours?

Notice how the agents shown when the Dijkstra's algorithm is animated seem to jump all around the network. Why is this?

THINGS TO TRY

Which knowledge sharing behaviour seems to be the best at this problem? Which seems to be the worst? Try out each of the different knowledge sharing behaviours to see which one is the most effective.

Try setting low and high values for the `time-to-live` slider. How does this affect the percentage of successful searches and the number of nodes visited as a result?

Try pressing the `setup-walkers` button multiple times. This will start the search with more than one walker.

Try out all the different types of networks with different slider settings. Which types of network causes problems for the various knowledge sharing methods, and which do not? i.e. Which seem to be more difficult to search?

Try switching the knowledge sharing behaviour mid-stream to see if there is any noticeable effect. For example, try switching from a `Word-of-Mouth` sharing method to a `Blackboard` sharing method then back again.

EXTENDING THE MODEL

Try adding other knowledge sharing mechanisms, or adding your own variations to the ones implemented in the model.

Try adding your own type of network, either randomly created or with a correspondence to a network in real-life. You will need to add input routines to create the network layout for the latter.

Instead of having the `walker` agents searching for a single node in the network, try adding resources to each node so that the simulation is similar to the resource discovery problem where a resource (such as an MP3 file) might appear at multiple nodes throughout the network, and each node will have many resources (up to tens of 1000s). You will need to add a `resources` variable to the `nodes-own` field where the `resources` is a table that contains the names of the resources that the node holds locally.

Try turning off the visualisation of the network by using the `hide-turtle` command to hide all agents and links. This will allow you to run much larger simulations – up to 100,000s of nodes.

NETLOGO FEATURES

Note the use of the `hatch` and `die` commands to create new child walkers at each node and terminate all existing `walkers`. Note also how the `next-locations-list` parameter for the `expand-walkers` command determines the knowledge sharing strategy being used that defines the different searches.

RELATED MODELS

See the Searching for Kevin Bacon model.

7.6 Using communicating agents for searching networks

Exercise 7.6.1:

In the Being Kevin Bacon model, there are many different slider, chooser and switch values to try out when running the search simulation. (This is done by first pressing the `setup-network` button followed by the `go-walkers` button.) For example, there are five different knowledge sharing methods – "`None`", "`Word of mouth`", "`Blackboard`", "`Combined 1`" and "`Combined 2`" – and there are also seven different types of networks – "`P2P-no-supernodes`", "`P2P-has-super-nodes`", "`P2P-random-single-link`", "`P2P-incremental`", "`P2P-incremental-1`", "`Star-central-hub`", and "`Hierarchical`". Indeed, if we wished to explore all the different configurations for the search simulation, then moving through all the switch/chooser/slider values from right to left and from top to bottom (i.e. those for all the green `Interface` elements but ignoring the two possible settings for the `animate-dijkstra` switch as that switches between Dijsktra's algorithm and the search simulation), then the total number of different settings would be as follows:

$$2\times 2\times 5\times 7\times 3\times 9991\times 100\times 20\times 50\times 20000\times 101\times 11\times 11\times 20\times 100\times 10\times 101 = 2.071793\times 10^{26}.$$

To run the simulation for all of these settings would not be feasible as the number exceeds the total number of seconds that have elapsed since the beginning of the universe which is estimated at approximately 1 followed by 17 zeros. Clearly, we have to be more selective of which experimental settings to run in order to investigate the behaviour of the model for the different configurations.

Thankfully, NetLogo provides a tool called BehaviorSpace that makes it easier to explore the "space" of possible behaviours for this model. The tool systematically runs the model multiple times, and records the results of each run to a file. (This process is sometimes called "parameter sweeping"). The tool can be run directly from the `Tools` menu in NetLogo. The way to use it is described in NetLogo's user manual. The following excerpt is taken from the manual and explains why such a tool is so useful:

"*The need for this type of experiment is revealed by the following observations. Models often have many settings, each of which can take a range of values. Together they form what in mathematics is called a parameter space for the model, whose dimensions are the number of settings, and in which every point is a particular combination of values. Running a model with different settings (and sometimes even the same ones) can lead to drastically different behavior in the system being modeled. So, how are you to know which particular configuration of values, or types of configurations, will yield the kind of behavior you are interested in? This amounts to the question of where in its huge, multi-dimension parameter space does your model perform best?*"

In this exercise, try out the BehaviorSpace tool for the Being Kevin Bacon model in order to gauge which knowledge sharing behaviours perform best at searching the simulated networks. First select the `BehaviorSpace` option form the `Tools` menu, then to create a new experiment setup click on `New` in the `BehaviorSpace` dialog box that appears. This will create a further `Experiment` dialog box where you specify the settings for your experiment. Enter a name for the experiment at the top of the dialog, and use the values listed below for the remaining fields.

Vary variables as follows (note brackets and quotation marks):

```
["links-per-node" 5]
["layout-type" "circle"]
["animate-dijkstra" true]
["nodes-to-delete" 0]
["centre-of-the-universe" "Kevin Bacon"]
["new-walkers-per-node" 1]
["simulation-ticks" 500]
["new-walkers-per-tick" 5]
["percent-in-both" 10]
["network-update" 0]
["network-type" "P2P-no-supernodes"]
["time-to-live" 5 10 15 20]
["nodes-to-add" 0]
["no-of-nodes" 1000]
["revisits-allowed" false]
["links-per-super-node" 20]
["no-of-super-nodes" 5]
["knowledge-sharing" "None" "Word of mouth" "Blackboard" "Combined
1" "Combined 2"]
```

Repetitions: 5
Measure runs using these reporters: report-success-rate
Measure runs at every step: √
Setup-commands: setup-network
Go commands: go-walkers
Stop condition:
Final commands:
Time limit: 0

The field that begins 'Vary variables as follows...' sets the range of values for the possible settings for the model. The Repetitions field specifies how many times the experiment is repeated. The 'Measure runs using these reporters' field specifies which reporters in the model are executed in order to collect data about the run. These are recorded in the output file that is produced. The 'Measure runs at every step' ensures that these reporters are executed every step. The setup-commands field specifies which commands in the model are executed at the start of each run, and the 'Go commands' field specifies the commands that are executed each step. There are also fields for specifying the stop condition for the run, the final commands to be executed at the end of the run, and a fixed time limit (if set to 0, no time limit is specified).

Once you have filled in the Experiment dialog box, press OK at the bottom, then press Run in the BehaviorSpace dialog box. A Run options dialog box will then appear. If you wish to have the results of the experiment saved to a file with a format suitable for loading into a spreadsheet application, then tick the Spreadsheet output option followed by the OK button. An Exporting as spreadsheet dialog box will appear where you can specify the name of the results output file. Pressing the Save button will then start the experiment running. Your Interface should end up similar to that shown in Figure 7.6.1). Once the experiment is finished, you can run it again by pressing the Run button in the Experiment dialog box.

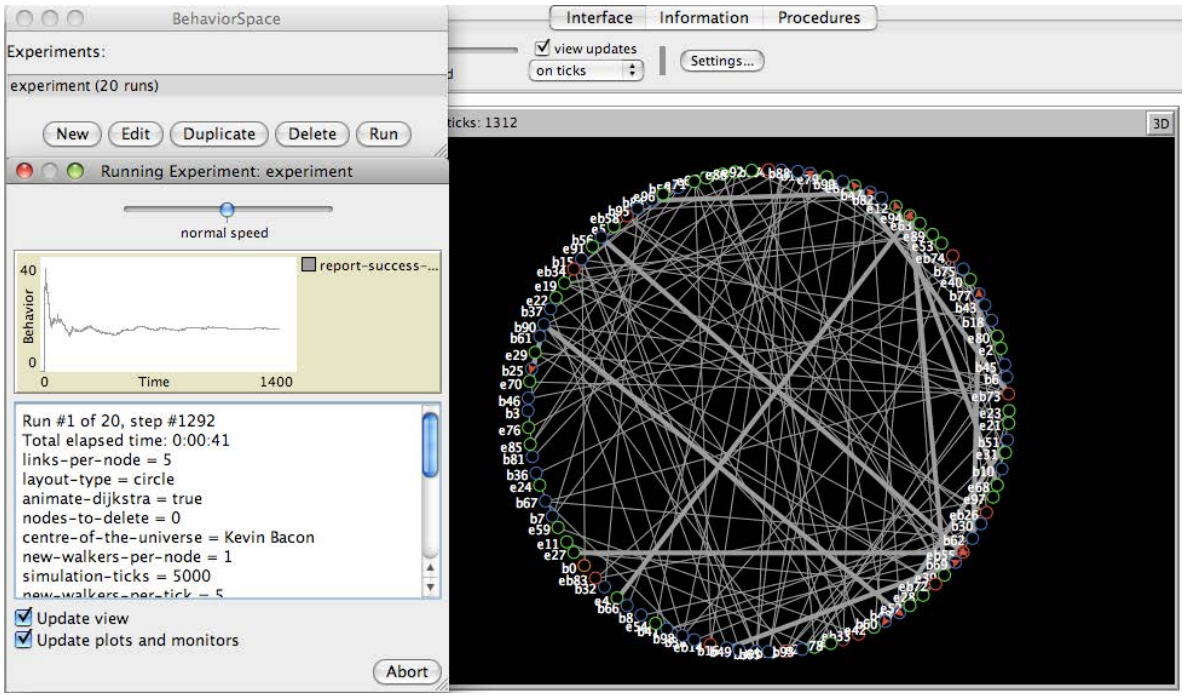


Figure 7.6.1. Screenshot of the Interface for the Being Kevin Bacon model with the BehaviorSpace tool in action using the settings above (but with the "no-of-nodes" set at 100 rather than 1000).

Have a look at the results of your experiment by loading the output file that was generated into a spreadsheet. Analyse the results to find out which of the knowledge sharing methods performed best on the various network types. Also try to spot any interesting results or trends in the data produced.

Create further experiments using different settings when the `Experiment` dialog box appears. Did these experiments produce different results to the first experiment?

7.7 Entropy and Information

Exercise 7.7.1:

Work out the entropy for messages to be encoded using symbols from the following probability distributions:

- for an alphabet containing two symbols with equal probabilities;
- for an alphabet containing four symbols with equal probabilities;
- for an alphabet containing six symbols, *Red*, *Orange*, *Brown*, *Yellow, Green* and *Lime*, with probabilities as follows:

$$p(Red)=\frac{1}{12}, p(Orange)=\frac{5}{12}, p(Brown)=\frac{3}{12}, p(Yellow)=\frac{1}{12}, p(Green)=\frac{1}{12}, p(Lime)=\frac{1}{12}.$$

Exercise 7.7.2:

Under what conditions is the entropy minimum for an alphabet of a specific size?

7.8 Calculating Entropy in NetLogo

Exercise 7.8.1: Cars Guessing Game NetLogo Model

Try out the Cars Guessing Game model in NetLogo:

Cars Guessing Game	http://files.bookboon.com/ai/Cars-Guessing-Game.nlogo

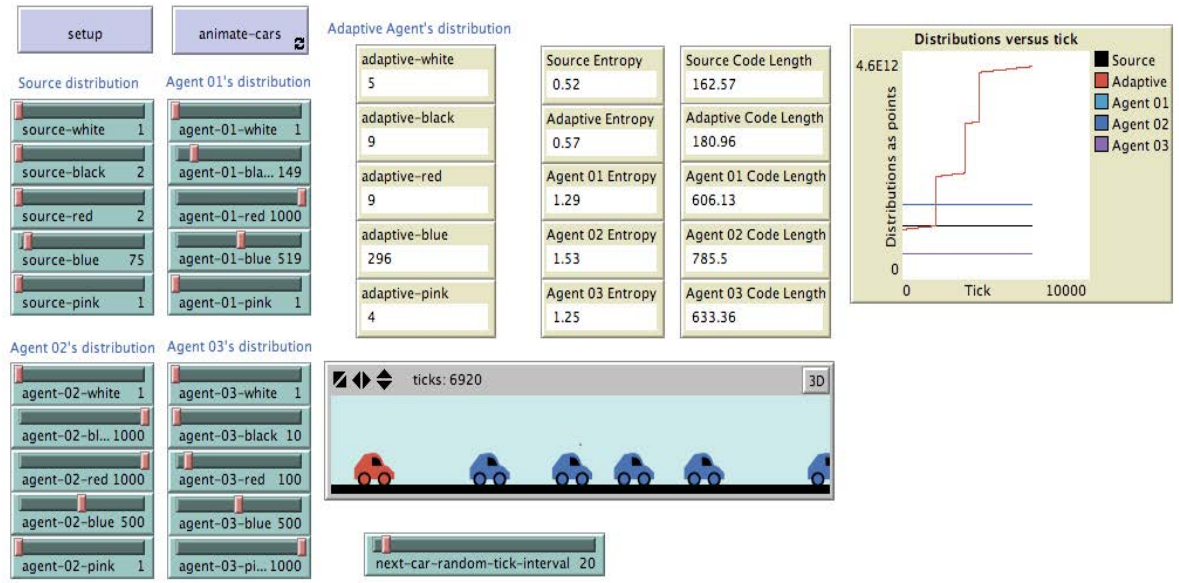


Figure 7.8.1. Screenshot of the Interface for the Cars Guessing Game model after the setup button has been pressed followed by the animate-cars button, with the slider values as shown in the image.

WHAT IS IT?

This model plays a simple game trying to guess the colour of cars as they drive past. Various agents maintain probability distributions that they use to predict the arrival of the cars. The source distribution is a fixed distribution that is used to generate the cars, so is the most accurate, and therefore its entropy is the source entropy and corresponding code length is the lowest (the code length is the optimal cost of encoding the sequence of cars given its probability distribution).

Agents 01 to 03 maintain fixed distributions which the user can adjust as they see fit by changing the slider values to the left of the Interface. An adaptive agent also maintains a dynamic distribution by updating counts of cars that have previously been seen. These counts are shown in the Interface by monitors under the heading "Adaptive Agent's distribution". The entropy and code length calculations are shown by monitors at the middle top of the Interface. The adaptive entropy and code length is the one that usually gets close to the source entropy and code length, whereas the other agents' entropy and code lengths reflect how different the slider settings are from the source distribution.

WHAT IS ITS PURPOSE?

The purpose of this model is show how entropy and code length calculations are made given a probability distribution.

HOW IT WORKS

A `car` turtle agent is used to represent the cars in the environment. These are created with colours distributed according to the source distribution. The arrival of the next car to the left of the environment is determined by a random number generated according to the slider `next-car-random-tick-interval`.

Turtle agents called "`agents`" are also present, but are not shown in the animation. These can be considered to be observers watching the cars go past. Each owns a distribution which is a list of counts that are used to calculate the probabilities in order to guess the upcoming cars.

HOW TO USE IT

Setting the values on the sliders for the `Source Distribution` will determine the distribution for the cars that appear in the animation. Setting the `next-car-random-tick-interval` will control how often the cars appear. The user can then set the slider values for the three `Agent distributions` on the left to see how this affects the entropy and code lengths for these distributions.

THE INTERFACE

The model's `Interface` buttons are defined as follows:

- `setup`: This resets the animation and initialises the counts, entropy and code length values.
- `animate-cars`: This starts the animation in the environment (the light blue rectangle shown middle bottom of the `Interface`).

The model's `Interface` sliders have the following naming convention:
`<agent-name>-<colour>`.
This sets the count for the specified colour in the list of counts maintained for the agent whose name is `<agent-name>`. These counts are used to determine the probability for a specific colour using the following formula:

$$P(colour) = C(colour) / C_{total}$$

where $C(colour)$ is the count for the symbol and C_{total} is the sum of all counts for all colours.

The model's `Interface` monitors are defined as follows:

- `adaptive-<colour>`: This is the count of the number of cars observed in the animation of the respective colour that is adaptively updated as the animation proceeds.
- `<agent-name> Entropy`: This is the entropy of the probability distribution maintained by the agent whose name is `<agent-name>`.
- `<agent-name> Code Length`: This is the cost of optimally encoding the sequence of observed cars given the agent's probability distribution.

The model's plot is defined as follows:

- The distributions are converted to a number that uniquely represents the values of the counts in the distribution. (For example, the number 1234 can be thought of as representing four separate counts – 1, 2, 3 and 4 – that combine to produce a unique number. In this case, the separate counts do not range from 0 to 9; they can range from 0 to 1000). The number that uniquely represents the distribution is then plotted versus ticks to show how the distributions evolve over the simulation.

THINGS TO NOTICE

Notice how well the adaptive distribution does compared to the source distribution. The source distribution will have the lowest code length total compared to the others, but usually the nearest distribution will be the adaptive one (unless the counts for one or more of the `Agent's distribution` exactly match the source distribution counts).

If the source distribution has equal counts, then the adaptive counts will rise at a similar rate, and as a consequence, the red line in the plot will rise diagonally from bottom-left to top-right. (Why?)

A horizontal line in the plot indicates a fixed distribution. Modifying the distribution counts in the middle of the animation will result in the lines in the plot changing to reflect this.

THINGS TO TRY

Try changing the distribution counts in the sliders to see what affect this has on the entropy and code length calculations, and on what happens in the plot. Note that the lower colours (blues and pinks) will cause the greatest shifts in the line plots. (Why?)

Can you achieve a situation where the code length of one of the three non-adaptive `Agents` is better than the adaptive Agent? (i.e. the code length is smaller and closer to the source agent's code length total).

7.9 Language Modelling

Exercise 7.9.1: Language Modelling NetLogo Model

Try out the Language Modelling model in NetLogo:

Language Modelling	http://files.bookboon.com/ai/Language-Modelling.nlogo

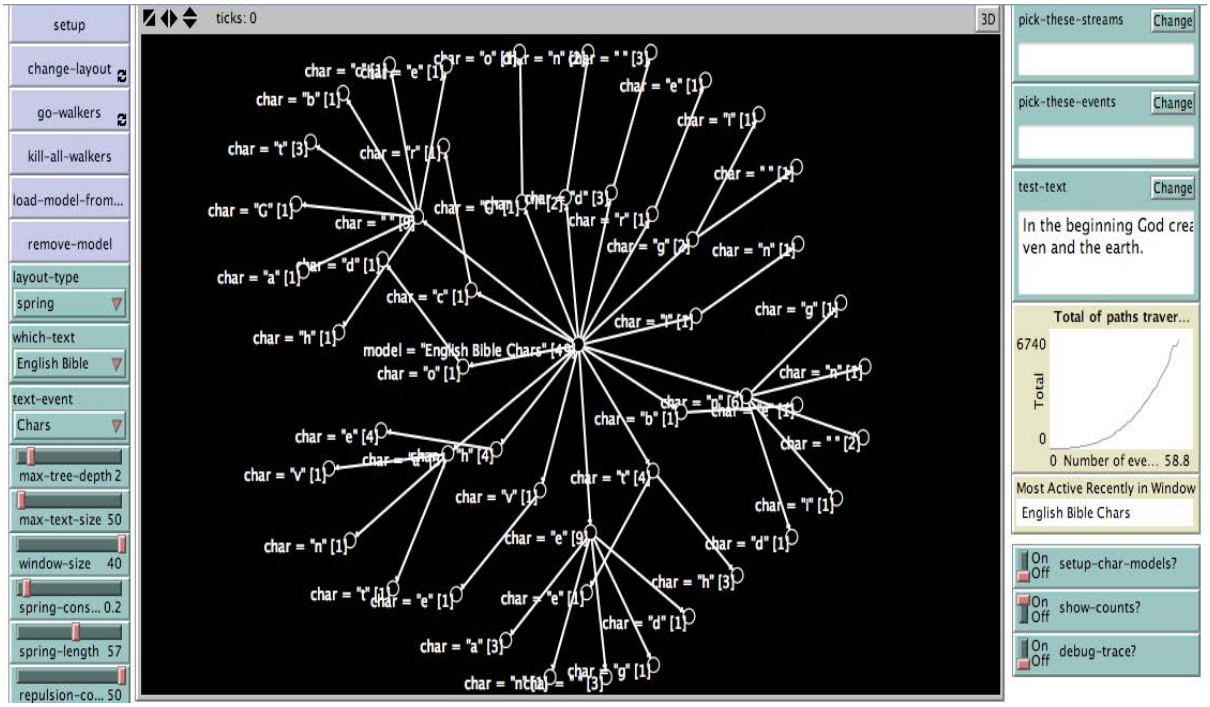


Figure 7.9.1. Screenshot of the Interface for the Language Modelling model after the setup button has been pressed followed by the load-model-from-text button and then the go-walkers button, with the slider values as shown in the image.

WHAT IS IT?

This model shows how a language model can be constructed from some training text. The type of language model is a fixed order Markov model – that is, according to the Markov property, the probability distribution for the next step and future steps depends only on the current state of the system and does not take into consideration the system's previous state. The language model is also fixed order – that is, the probability is conditioned on a fixed length prior context.

Note: There is some possible confusion of the term "model" for this NetLogo "model". NetLogo uses the term "model" to refer to a program/simulation. This can be confused with the use of the term "model" in the phrase "language model" which is a mechanism for assigning a probability to a sequence of symbols by means of a probability distribution. In the information listed below, the two types of models will be explicitly distinguished by using the phrases "NetLogo model" and "language model".

WHAT IS ITS PURPOSE?

The purpose of this NetLogo model is to show various important features of language models and to visualise them using NetLogo link and turtle agents.

HOW IT WORKS

The data being modelled is conceptualised as a stream of events (i.e. analogously to Event Stream Processing or ESP) where events occur with a specific order for a specific stream, but at the same time can occur simultaneously on separate streams.

A training text sequence is used to train the language model. This language model is then used to process a test text sequence. Each stream event from the training data is represented by a state turtle agent, and paths between states are represented by path link agents. When the language model is used for an application, such as processing a test sequence of text, then `walker` turtle agents are used to walk the network of states and links to show at what point the current contexts are active. Text turtle agents are used for maintaining information about a particular text such as sums and a variable for storing the language model.

HOW TO USE IT

Press the `setup` button in the `Interface` to reset all agents. To load a language model, first select which text you wish to load from the `which-text` chooser, then press the `load-model-from-text` button. This will load the language model into the environment so that it can be visualised using the layout specified by the `layout-type` chooser. If you have selected a `spring layout-type`, then pressing the `change-layout` button can sometimes remove some of the clutter once it has been loaded. Other language models from different texts can be subsequently loaded if needed. The `pick-these-streams` and `pick-these-events` input boxes can be used to filter out the language model so that only the part of the language model that matches the specific streams and events are visualised. The `text-event` chooser specifies what type of event should be loaded. The NetLogo model at the moment only processes character and word events.

To see how the language model(s) process a testing text sequence, first edit the `test-text` input box and insert the test text sequence that you would like to be processed, then press the `go-walkers` button. The paths taken through the language model(s) will then be highlighted as the testing text sequence is processed sequentially.

THE INTERFACE

The NetLogo model's `Interface` buttons are defined as follows:

- `setup`: This (re-) initialises all the variables.
- `change-layout`: This changes the layout to move the agents apart if the `layout-type` slider is set to "`spring`".

- `go-walkers`: If a language model is visualised in the environment, then this will run an animation that shows how the testing sequence (specified within the `test-text` input box) is processed using the language model.
- `kill-all-walkers`: This kills all current walkers.
- `load-model-from-text`: This will load a language model using the training text specified by the `which-text` chooser. The type of events (characters or words) is specified by the `text-event` chooser.

The NetLogo model's `Interface` sliders, and choosers are defined as follows:

- `layout-type`: This specifies the type of layout to use for the visualisation – either `spring` or `radial`.
- `which-text`: This specifies the training text to use to train the language models.
- `text-event`: This specifies the type of event in the training text being processed – either `characters` or `words`.
- `max-tree-depth`: This specifies the maximum tree depth of the language model (the order of the language model is equal to the `max-tree-depth` + 1).
- `max-text-size`: This specifies the maximum number of characters to be processed from the training text. A relatively low number is recommended, otherwise there will be too much to visualise.
- `window-size`: This sets the window size over which the NetLogo model tries to classify the text using a crude form of text categorisation. In this case, the activity count of the language models is maintained (where an activity is the highlighting of a crossed link during the animation), and the most active language model is shown in the `Most Active Recently in Window` monitor. The category of the testing sequence can then be set to the language label of the most active language model. i.e. If there were French, English and Spanish language models loaded from various training texts, and the most active language model shown by the monitor was the French language model, then the text can be categorized as being French.

The NetLogo model's `Interface` switches are defined as follows:

- `setup-char-models?`: If set to `On`, this sets up the models to recognize words and punctuation from characters.
- `show-counts?`: If set to `On`, this will show the counts in the visualisation of the language models.
- `debug-trace?`: If set to `On`, this will print out copious amounts of debug information to show the code being executed.

The NetLogo model's `Interface` input boxes, plot and monitor are defined as follows:

- `pick-these-streams`: This can be used to prune out what is visualised of each language model. Only the specified streams will be visualised.
- `pick-these-events`: This can be used to prune out what is visualised of each language model. Only the specified events will be visualised.
- `test-text`: This is the testing text that is being processed sequentially.
- `Total of paths traversed plot`: This plots the number of paths traversed in the language model versus the number of events processed.
- `Most Active Recently in Window`: This is the most active model in the past window (of length `window-size`) in the testing text.

THINGS TO NOTICE

Notice that word-based language models are significantly smaller in complexity when visualised than the complexity of the character language models. This is to be expected since there are a lot less word events in a text than character events (for example, there are on average for the English language 5 characters per word).

Try loading several language models all together. Then notice how when a testing text is chosen with language similar to one of the training texts used to train the language models, then the language model that becomes most active is usually the one with the similar language. This is because language can readily be identified uniquely, and explains why it is possible to do authorship ascription using language models with a high degree of accuracy.

THINGS TO TRY

Try loading all the language models from the different texts (specified by the `which-text` chooser). While doing this, set different values for the order of the language models by setting the `max-tree-depth` slider from 1 to 8. (Be careful when this value is set high – the language models can take some time to load, and the visualisation becomes extremely cluttered).

Try extending the maximum size of the training text (using the `max-text-size` slider) and see what affect this has on the complexity of the language models.

Try changing the `window-size` slider to see what effect this has on the `Most Active Recently in Window` monitor.

EXTENDING THE NETLOGO MODEL

Try a variation of the NetLogo model where most of the nodes are hidden, and only the ones chosen to be visualised (by the `pick-these-streams` and the `pick-these-events` choosers) are shown in the environment. This will mean much larger language models can be loaded (since the `reset-layout` procedure for the spring layout type is what slows down the program when a language model is loaded). An alternative animation could also be tried where only the active part of language model is animated. i.e. Everything in the language model remains hidden until it is activated.

If entropy and code length calculations are added (like in the Cars Guessing Game NetLogo model), then the model can be extended to perform with a high degree of accuracy the task of language identification, authorship identification or text categorisation simply by choosing the class (i.e. language, author or category) of the testing text to be the class of the text used to train the language model which produces the smallest code length on the testing sequence.

RELATED MODELS

See the Central Park Events and Wall Following Events NetLogo models. For prediction using these language models, see the Shannon Guessing Game NetLogo model.

7.10 Entropy of a Language

Exercise 7.10.1: Shannon Guessing Game NetLogo Model

Try out the Shannon Guessing Game model in NetLogo:

Shannon Guessing Game	http://files.bookboon.com/ai/Shannon-Guessing-Game.nlogo

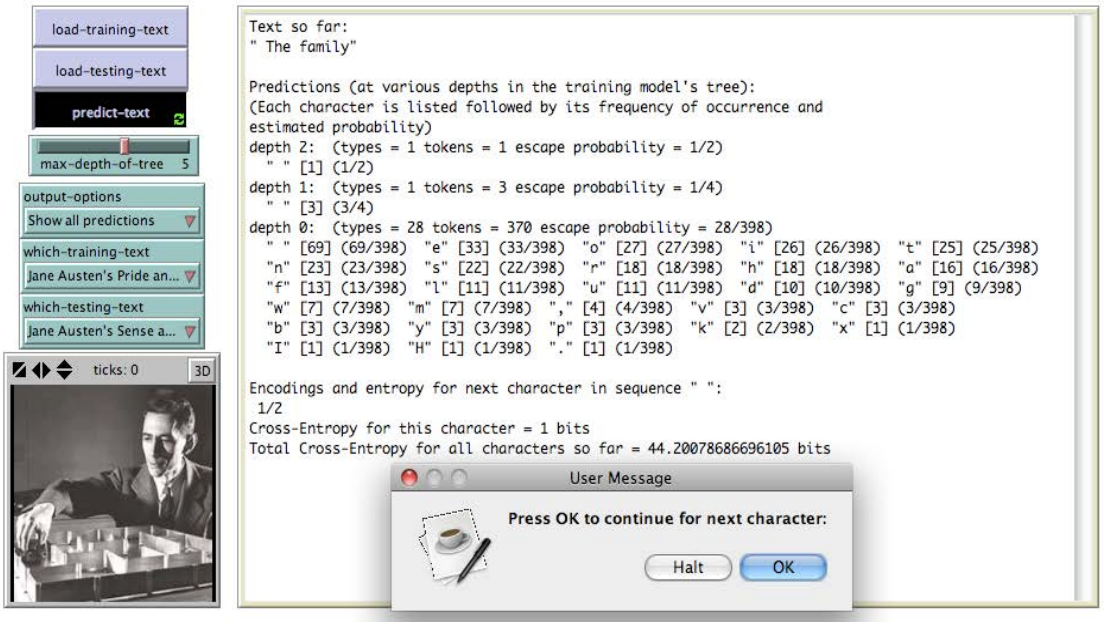


Figure 7.10.1. Screenshot of the Interface for the Shannon Guessing Game model after the load-training-text and load-testing-text buttons have been pressed followed by the predict-text button and the first 11 characters have been predicted, with the slider values as shown in the image.

WHAT IS IT?

This model shows how a language model can be constructed from some training text and then used to predict text – i.e. play the "Shannon Guessing Game", a game where the agent (human or computer) tries to predict upcoming text, one letter at a time, based on the prior text it has already seen.

Note: There is some possible confusion of the term "model" for this NetLogo "model". NetLogo uses the term "model" to refer to a program/simulation. This can be confused with the use of the term "model" in the phrase "language model" which is a mechanism for assigning a probability to a sequence of symbols by means of a probability distribution. In the information listed below, the two types of models will be explicitly distinguished by using the phrases "NetLogo model" and "language model".

The type of language model used in this NetLogo model is a fixed order Markov model – that is, according to the Markov property, the probability distribution for the next step and future steps depends only on the current state of the system and does not take into consideration the system's previous state. The language model is also fixed order – that is, the probability is conditioned on a fixed length prior context.

HOW IT WORKS

The language model is based on the PPMC compression algorithm devised by John Cleary and Ian Witten. It uses a back-off mechanism called escaping, to smooth the probability estimations with lower order models in order to overcome the zero frequency problem. This is the problem that occurs when an upcoming character has not been seen before in the context, and therefore has zero frequency. Assigning a zero probability estimate will result in an infinite code length (since $\log 0 = \infty$) which means that it is impossible to encode the occurrence using this estimate. To overcome this, the estimates are calculated by smoothing with lower order models using the escape mechanism.

The escape probability is estimated using Method C devised by Alistair Moffat (hence why it is called PPMC). The smoothing method is often erroneously called Witten-Bell smoothing although it was invented by Alistair Moffat. The PPMC language model implemented by this NetLogo model does not implement full exclusions or update exclusions.

WHAT IS ITS PURPOSE?

The purpose of this NetLogo model is show how to implement a language model based on the PPMC compression scheme, and then use it to predict text one character at a time.

HOW IT WORKS

The data being modelled is conceptualised as a stream of events (i.e. analogously to Event Stream Processing or ESP) where events occur with a specific order for a specific stream, but at the same time can occur simultaneously on separate streams.

A training text sequence is used to train the language model. This language model is then used to process a test text sequence. Each stream event from the training data is represented by a `state` turtle agent, and paths between states are represented by `path` link agents. When the language model is used for prediction, such as when playing the Shannon Guessing Game as in this NetLogo model, then `walker` turtle agents are used to walk the network of states to determine which of the current contexts are active. These are then used to make the probability estimates. `Text` turtle agents are used for maintaining information about a particular text such as sums and a variable for storing the language model.

HOW TO USE IT

To use this NetLogo model, perform the following steps:

1. Select a training text to load using the `which-training-text` slider.
2. Load and build a language model for it using the `load-training-text` button.
3. Select a testing text to perform the prediction on using the `which-testing-text` slider.
4. Then predict the testing text one character at a time by pressing the `predict-text` button.

THE INTERFACE

The model's `Interface` buttons are defined as follows:

- `load-training-text`: This loads the training text specified by the `which-training-text` slider. At the same time, it constructs the language model from it.
- `load-testing-text`: This loads the testing text specified by the `which-testing-text` slider.
- `predict-text`: This predicts the text in the testing text one character at a time.

The model's `Interface` slider and choosers are defined as follows:

- `max-depth-of-tree`: This is the maximum depth of the context tree. The order of the model is (`max-depth-of-tree` – 1).
- `output-options`: This determines how much output is written to the output box during the prediction phase.
- `which-training-text`: This slider allows the user to select from a small but eclectic list of natural language texts to use as the data for training the language model.

- `which-testing-text`: This slider allows the user to select from a small but eclectic list of natural language texts to use as the data on which the language model is tested by predicting each character one after the other.

THINGS TO NOTICE

Notice how the prediction improves as the `max-depth-of-tree` variable is increased from 1 to 5 or 6, then drops off slightly. However, this depends to a great extent on the training text used to build the language model and how closely it is related to the testing text.

Notice how poor the prediction is when the training and testing texts are from different languages.

Notice also that for the first few characters in the testing text, the prediction is relatively poor. Why is this?

In contrast, notice how well the language model does at predicting the testing text if it is trained on exactly the same text (i.e. the training and texts are exactly the same). Note: This situation seldomly appears in real life prediction because usually the testing sequence is completely unpredictable. Note also that the best the language model for higher order models can do for predicting the next character is often only 1/2 or 1 bit to encode even in the situation where it has been trained on the testing text. Why is this? Hint: What would happen if the length of the training text was much longer?

THINGS TO TRY

Try altering the `max-depth-of-tree` to see how this affects the prediction.

Try different combinations of training and testing texts.

EXTENDING THE MODEL

Extend the PPMC language model so that it implements full exclusions and update exclusions.

Can you devise further methods to improve the prediction of the language model?

RELATED MODELS

See the Language Modelling NetLogo model and the Cars Guessing Game NetLogo model.

7.11 Communicating Meaning

Exercise 7.11.1:

Fill in the conceptual metaphor for the following two verses of the ABBA song 'The Winner Takes It All':

The gods may throw a dice
Their minds as cold as ice
And someone way down here
Loses someone dear
The winner takes it all
The loser has to fall
It's simple and it's plain
Why should I complain?

The judges will decide
The likes of me abide
Spectators of the show
Always staying low
The game is on again
A lover or a friend
A big thing or a small
The winner takes it all

Exercise 7.11.2:

How might it be possible to communicate with an alien race? Is there some common frame of reference with which we might communicate? Could mathematics, for example, provide a common ground to facilitate communication? But is human mathematics itself based too much on our own human embodied existence? Is there an alternative approach where we could encode 'meaning' directly in the message we wish to convey to the alien race?

Exercise 7.11.3:

Analyse the following excerpt from Bill Gates' autobiography *My Life*:

"*Al Gore gave a marvelous concession speech. It was genuine, gracious, and patriotic. When I called to congratulate him, he told me that a friend who was a professional comedian had joked to him that he had gotten the best of both worlds: he had won the popular vote and didn't have to do the job.*

The next morning, after Tony Blair and I talked a bit, I walked outside, complimented Al, and pledged to work with President-elect Bush. Then Tony and Cherie accompanied Hillary, Chelsea, and me to the University of Warwick, where I gave another of my farewell speeches, this one on the approach to globalization our Third Way group had embraced: trade plus a global contract for economic empowerment, education, health care, and democratic governance. The speech also gave me a chance to publicly thank Tony Blair for his friendship and our partnership. I had treasured our times together and would miss them.

"Before we left England, we went to Buckingham Palace, accepting Queen Elizabeth's kind invitation to tea. We had a pleasant visit, discussing the election and world affairs. Then Her Majesty took the unusual step of accompanying us down to the ground floor of the palace and walking us out to our car to say good-bye. She, too, had been gracious and kind to me over the past eight years."

Describe how this excerpt reveals some of the thoughts of Bill Gates – i.e. what he knows and doesn't know; his beliefs; his likes or dislikes; his opinions; his hopes and fears; what he respects and has high regard for; or what he disrespects and has contempt for; what he supports or is opposed to; his desires/wants/needs; and his intentions.

8 Search

8.1 Search Behaviour

Exercise 8.1.1:

Research in the literature how the terms 'search' and 'research' are being defined. Observe the methods you use to perform the research. Are there distinctly different meanings of these terms or do they all spring from the same generic meaning?

8.2 Search Problems

Exercise 8.2.1: Searching Mazes NetLogo Model

Try out the Searching Mazes model in NetLogo:

Searching Mazes	http://files.bookboon.com/ai/Searching-Mazes.html

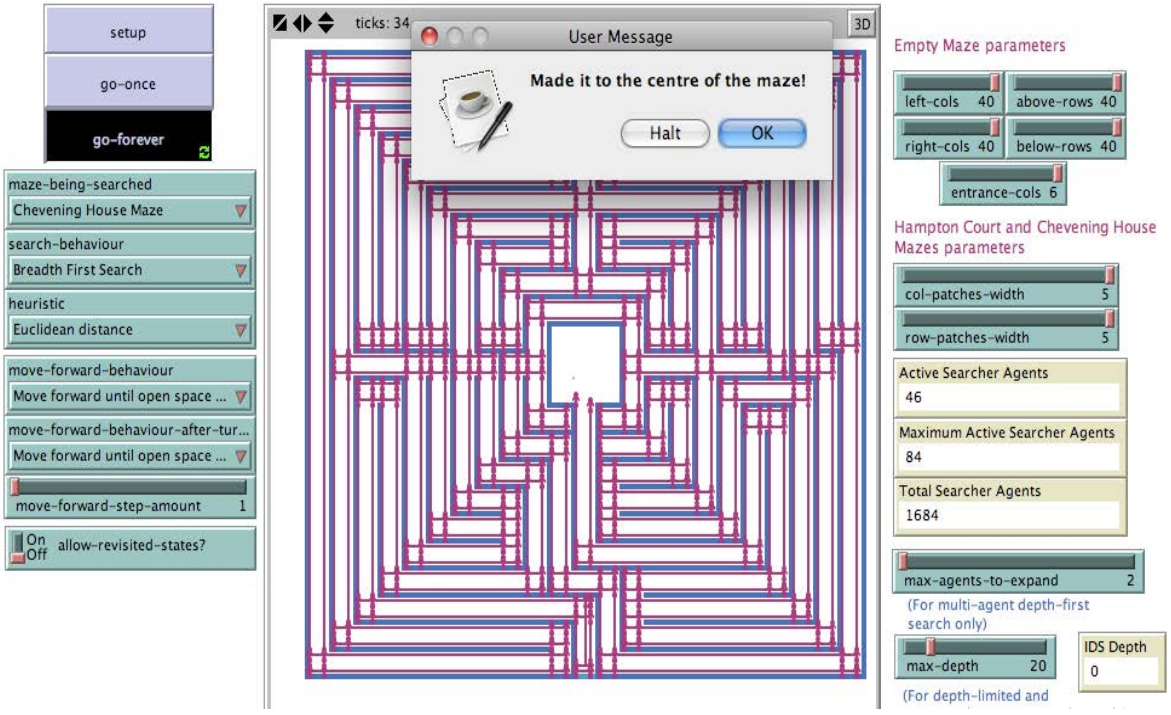


Figure 8.2.1. Screenshot of the Interface for the Searching Mazes model for the Chevening House Maze after the setup button has been pressed followed by the go-forever button, with the slider values as shown in the image.

WHAT IS IT?

This model applies standard search algorithms to the problem of searching mazes. Three mazes are provided in the model: the empty maze with no internal walls; the Hampton Court maze, which is a schematic representation of the real-life garden maze at Hampton Court Palace in the U.K.; and the Chevening House maze, which is a representation of the real-life maze at Chevening House in the U.K. designed to thwart the hand-on-the-wall method of getting to the centre of the maze. The entrance of the empty maze is at the bottom middle of the maze, and the goal is to reach the exit at the top of the maze. The entrance of the Hampton Court maze is in the middle bottom, and the goal is to reach the centre. Similarly, the entrance for the Chevening House maze is in the middle bottom, and the goal is to reach the centre.

The problem is the following: How do you reach the goal from the entrance? To solve this problem, search needs to be employed to try out the different paths that are possible. This model implements some of the classic search algorithms and these can be applied to this toy problem to see how the different search strategies perform against each other.

WHAT IS ITS PURPOSE?

The purpose of this model is to show how some of the classic search algorithms such as breadth-first search and depth-first search can be implemented in NetLogo, and also how they can then be applied to solving a classic toy problem in order to compare how they perform.

HOW IT WORKS

The model implements the search algorithms in a novel way by making use of an agent-oriented approach which is relatively easy to implement in NetLogo. Rather than use a queue data structure to implement the classic search algorithms (this is a standard solution; see, for example, Russell & Norvig's AI textbook), the model instead adopts a purely agent-oriented solution where the information is distributed amongst NetLogo agents. These agents conduct the search by transferring information to other agents that continue the search. Hence, an explicit queue data structure separate from what is happening in the search is not needed (although technically the information is still being stored in a queue behind the scenes, but the implementation of this is hidden behind the top-level commands that NetLogo provides).

The hope is that this "queueless" agent-oriented approach provides a more intuitive solution that makes it easier to grasp how the search strategies work. It uses a situated, embodied first person design perspective, therefore arguably making it easier to understand the essential differences between the search strategies.

The breed of searcher turtle agents is used for the searchers that move throughout the maze trying to get to the goal. These searcher agents maintain information about the current state of the search such as the time taken and the path and estimated costs. Each searcher agent expands the search in three directions: forward; a right 90 degrees turn then forward; and a left 90 degrees turn then forward. The forward movement behaviour of the searcher agents in the maze is controlled by two choosers – `move-forward-behaviour` and the `move-forward-behaviour-after-turning`. The options for the type of move forward behaviour used by both these choosers are as follows:

- "`Move forward n steps unless hit wall`": The agent moves forward a number of steps determined by the `move-forward-step-amount` slider.
- "`Move forward until hit wall`": The agent will move forward indefinitely until it hits a wall.
- "`Move forward until any open space`": The agent will move forward until it has found open space (i.e. until there is a wall in the way or there is open space on both sides).
- "`Move forward until open space after side wall`": The agent will move forward until it has found open space (i.e. until there is a wall in the way or there is open space on both sides but will only stop if it has previously been following a wall on either side).

HOW TO USE IT

To initialise the search and draw the selected maze (specified by the `maze-being-searched` chooser), press the `setup` button in the `Interface`. A turtle agent with a shape of a person will slowly walk towards the entrance of the maze. Then the entrance will be blocked (to prevent the agent from exiting in that direction).

To start the search, either press the `go-once` button to make the search proceed one step at a time, or press the `go-forever` button to make the search proceed continuously until it reaches the goal state or it gets stuck.

THE INTERFACE

The model's `Interface` buttons are defined as follows:

- `setup`: This will clear the environment and all variables, and redraw the selected maze (chosen by using the `maze-being-searched` slider).
- `go-once`: This will make the search proceed one step at a time.
- `go-forever`: This will make the proceed continuously until it has reached the goal state or it gets stuck.

The model's `Interface` choosers, sliders and switch are defined as follows:

- `maze-being-searched`: This selects the maze to be searched. There are three available: the empty maze, the Hampton Court maze, and the Chevening House maze.
- `search-behaviour`: This specifies what search strategy the agents should employ when performing the search. A full description of the different search strategies can be found in Chapter 8 of the book "*Artificial Intelligence: Agent Behaviour I*".
- `heuristic`: This specifies the heuristic to be used for the Informed searches – Greedy Best First Search and A* Search.
- `move-forward-behaviour`: This specifies the type of move forward behaviour the searcher agent executes. (For an explanation of the options, see the `How It Works` section above).
- `move-forward-behaviour-after-turning`: This specifies the type of move forward behaviour the searcher agent executes after it has turned either right or left. (For an explanation of the options, see the `How It Works` section above).
- `move-forward-step-amount`: This is the amount of steps forward the agent makes when it moves forward (either directly ahead of itself on its current heading, or after it has made a right or left turn).

- `allow-revisited-states?`: This is a flag that if set to `On` will allow the search to proceed to already visited states.
- `left-cols`, `right-cols`, `above-rows`, `above-cols`, `entrance-cols`: These sliders set the height and width of the empty maze, and the size of the entrance.
- `col-patches-width`, `row-patches-width`: These sliders set the height and width of the Hampton Court and Chevening House mazes.
- `max-agents-to-expand`: This sets the maximum number of agents to expand the search each step. It is only used by the Multi-Agent Depth First Search.
- `max-depth`: This sets the maximum depth of the search, for the depth-limited search and the Iterative Deepening Search only.

The model's `Interface` monitors are defined as follows:

- `Active Searcher Agents`: This shows the current number of active searcher agents still participating in the search.
- `Maximum Active Searcher Agents`: This shows the maximum value that the `Active Searcher Agents` monitor has achieved throughout the search.
- `Total Searcher Agents`: This shows the total number of searcher agents that the search has used.
- `IDS Depth`: This is the current depth during the execution of the Iterative Deepening Search.

THINGS TO NOTICE

Notice the effect of turning the flag `allow-revisited-states?` `On` and `Off`. Why is turning it `Off` so effective at dramatically reducing the search for the different search behaviours?

Notice that some of the searches if repeated with the same settings end up in different places. Why is this?

Notice that many of the behaviours result in the searching turtle agents getting stuck. When does this happen, and why?

THINGS TO TRY

Try slowing down the animation by changing the speed slider at the top of the `Information` window. You will be able to see how the search proceeds.

Which search behaviour seems to be the best at this problem? Which seems to be the worst? Try out each of the different search behaviours to see which one is the most effective. Find out how well each search algorithm performs against the four evaluation criteria – time complexity, space complexity, optimality and complexity. Relate the theory to what happens in practice.

Try switching the searching behaviour mid-stream. The difference in behaviours is most noticeable in the empty maze. For example, try switching from a breadth-first search to a depth-first search then back again (after setting the `move-forward-behaviour` and `move-forward-after-turning-behaviour` chooser settings to "`Move forward n steps unless hit wall`", and set the `move-forward-step-amount` slider to 1).

Try sliding the `move-forward-step-amount` slider back and forth as the search proceeds if you have set either of the `move-forward-behaviour` and `move-forward-after-turning-behaviour` choosers to be "`Move forward n steps unless hit wall`".

Try out the different heuristics for the informed searches (Greedy Best First Search and A* Search). Do they have any effect on how effective the search is?

Try out the all the different combinations of the `move-forward-behaviour` and `move-forward-after-turning-behaviour` chooser settings, and the `move-forward-step-amount` slider setting. This results in a huge variety of behaviours for the searching turtle agents. For example, have a go at reproducing the behaviour shown by the screenshots in Chapter 8 (e.g. Figure 8.10) of the book *Artificial Intelligence – Agent Behaviour I*.

Try changing the parameters of each maze by altering the settings of the sliders on the right.

EXTENDING THE MODEL

Try adding other search algorithms, or adding your own variations to the ones implemented in the model.

Try adding your own maze, perhaps of a garden maze or other maze that exists in real life. If your maze does have a real-life equivalent, note the discrepancies that you have to introduce into the virtual representation where it does not exactly correspond to what is happening in the real-life maze.

NETLOGO FEATURES

Note the use of the `hatch` and `die` commands to clone child searchers and terminate parent searchers thus avoiding the need for a separate queue data structure to maintain the information during the search. Note also the variation in the `ask` command that determines the strategy being used that defines the different searches.

RELATED MODELS

For other search problems, see the Searching For Kevin Bacon model, and the Searching Mazes model. For an insight into the Manhattan distance and Euclidean distance metrics used as heuristics for the Informed searches, see the Manhattan Distance model.

Exercise 8.2.2: Missionaries and Cannibals NetLogo Model

Try out the Missionaries and Cannibals model in NetLogo:

Missionaries and Cannibals	http://files.bookboon.com/ai/Missionaries-and-Cannibals.html

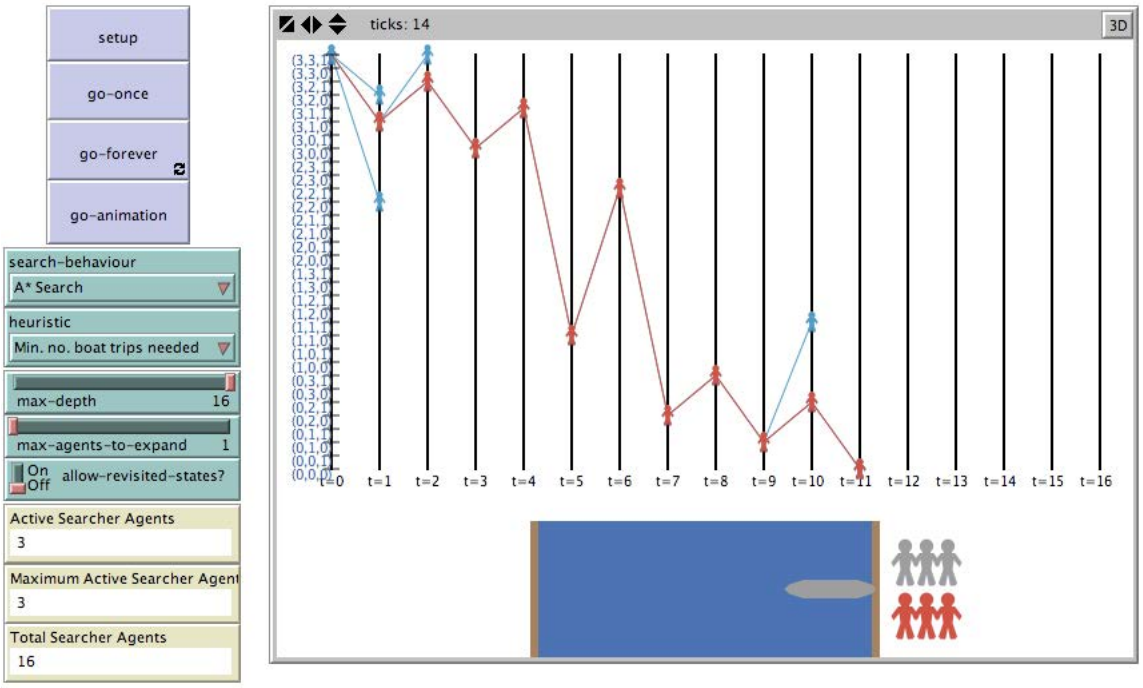


Figure 8.2.2. Screenshot of the Interface for the Missionaries and Cannibals model for after the setup button has been pressed followed by the go-forever button and then the go-animation button, with the slider values as shown in the image.

WHAT IS IT?

This model applies standard search algorithms to the classic search problem called Missionaries and Cannibals. In this toy problem, there are 3 missionaries and 3 cannibals. They have arrived together on one side of a river and they all wish to cross to the other side. Luckily, there is available a boat for crossing the river. However, there are two catches to the problem. The first catch is that only two people can fit in the boat at any one time. The second catch is that the cannibals are not allowed to outnumber the missionaries at any stage. If they do, they will overpower the missionaries and then eat them.

The problem is the following: Is it possible to get all the missionaries and cannibals safely across to the other side of the river? To solve this problem, search needs to be employed to try out the different scenarios. This model implements some of the classic search algorithms and these can be applied to this toy problem to see how the different search strategies perform against each other.

WHAT IS ITS PURPOSE?

The purpose of this model is to show how some of the classic search algorithms such as breadth-first search and depth-first search can be implemented in NetLogo, and also how they can then be applied to solving a classic toy problem in order to compare how they perform.

HOW IT WORKS

The model implements the search algorithms in a novel way by making use of an agent-oriented approach which is relatively easy to implement in NetLogo. Rather than use a queue data structure to implement the classic search algorithms (this is a standard solution; see, for example, Russell & Norvig's AI textbook), the model instead adopts a purely agent-oriented solution where the information is distributed amongst NetLogo agents. These agents conduct the search by transferring information to other agents that continue the search. Hence, an explicit queue data structure separate from what is happening in the search is not needed (although technically the information is still being stored in a queue behind the scenes, but the implementation of this is hidden behind the top-level commands that NetLogo provides).

The hope is that this "queueless" agent-oriented approach provides a more intuitive solution that makes it easier to grasp how the search strategies work. It uses a situated, embodied first person design perspective, therefore arguably making it easier to understand the essential differences between the search strategies.

The breed of searcher turtle agents is used for the searchers that move throughout the maze trying to get to the goal. These searcher agents maintain information about the current state of the search such as the time taken and the path and estimated costs. Each searcher agent expands the search by seeing if the following actions are possible:

- two missionaries get in the row-boat and row to the other side;
- two cannibals get in the row-boat and row to the other side;
- one missionary and one cannibal get in the row-boat and row to the other side;
- just one missionary gets in the row-boat and rows to the other side;
- just one cannibal gets in the row-boat and rows to the other side.

Note that the states of the search for this problem can be represented by a 3-tuple:

(#missionaries on start side of river, #cannibals on start side of river, #boats on start side of river)

Hence, the start state is represented by the tuple (3, 3, 1) and the goal state by (0, 0, 0).

In the `Interface`, a graph using parallel co-ordinates is used to visualise the search as it proceeds. An animation of one possible solution to the problem is also provided below the graph. The graph is updated as the animated solution proceeds.

HOW TO USE IT

To initialise the search, press the `setup` button in the `Interface`. A person shape will appear at the start state (3, 3, 1) at the top left of the graph. The goal state is at the bottom of the graph, so a rough estimate of whether a search is making progress is how far it has progressed downwards.

To start the search, either press the `go-once` button to make the search proceed one step at a time, or press the `go-forever` button to make the search proceed continuously until it reaches the goal state or other termination criteria are satisfied (i.e. it reaches the right side of the graph without finding a solution).

Pressing the `go-animation` button will start the animation and show the path taken at the same time in the graph. This is shown using the red colour.

THE INTERFACE

The model's `Interface` buttons are defined as follows:

- `setup`: This will clear the environment and all variables, reset the animation and redraw the graph.
- `go-once`: This will make the search proceed one step at a time.
- `go-forever`: This will make the search proceed continuously until it has reached the goal state or it is deemed to be unsuccessful.
- `go-animation`: This will start the animation shown at the middle bottom of the `Interface`.

The model's `Interface` choosers, sliders and switch are defined as follows:

- `search-behaviour`: This specifies what search strategy the agents should employ when performing the search. A full description of the different search strategies can be found in Chapter 8 of the book "*Artificial Intelligence: Agent Behaviour I*".
- `heuristic`: This specifies the heuristic to be used for the Informed searches – Greedy Best First Search and A* Search.
- `max-depth`: This sets the maximum depth of the search.
- `max-agents-to-expand`: This sets the maximum number of agents to expand the search each step. It is only used by the Multi-Agent Depth First Search.
- `allow-revisited-states?`: This is a flag that if set to `On` will allow the search to proceed to already visited states.

The model's `Interface` monitors are defined as follows:

- `Active Searcher Agents`: This shows the current number of active searcher agents still participating in the search.
- `Maximum Active Searcher Agents`: This shows the maximum value that the `Active Searcher Agents` monitor has achieved throughout the search.
- `Total Searcher Agents`: This shows the total number of searcher agents that the search has used.
- `IDS Depth`: This is the current depth during the execution of the Iterative Deepening Search.

THINGS TO NOTICE

Notice the effect of turning the flag `allow-revisited-states?` `On` and `Off`. Why is turning it `Off` so effective at dramatically reducing the search for the different search behaviours?

Notice that some of the searches if repeated with the same settings end up in different places (for example, try Greedy Best First Search several times using the heuristic "`Min. no. boat trips needed`" and with the flag `allow-revisited-states?` set to `Off`.). Why is this?

THINGS TO TRY

Which search behaviour seems to be the best at this problem? Which seems to be the worst? Try out each of the different search behaviours to see which one is the most effective. Find out how well each search algorithm performs against the four evaluation criteria – time complexity, space complexity, optimality and complexity. Relate the theory to what happens in practice.

Try figuring out how many solutions there are to the problem (the one shown in the animation is not the only one).

Try out the different heuristics for the informed searches (Greedy Best First Search and A* Search). Do they have any effect on how effective the search is?

EXTENDING THE MODEL

Try adding other search algorithms, or adding your own variations to the ones implemented in the model.

NETLOGO FEATURES

Note the use of the `hatch` and `die` commands to clone child searchers and terminate parent searchers thus avoiding the need for a separate queue data structure to maintain the information during the search. Note also the variation in the `ask` command that determines the strategy being used that defines the different searches.

RELATED MODELS

For other search problems, see the Searching For Kevin Bacon model, and the Searching Mazes model. For an insight into the Manhattan distance and Euclidean distance metrics used as heuristics for the Informed searches, see the Manhattan Distance model.

Exercise 8.2.3: Searching for Kevin Bacon NetLogo Model

Try out the Searching for Kevin Bacon model in NetLogo:

Searching for Kevin Bacon	http://files.bookboon.com/ai/Searching-for-Kevin-Bacon.html

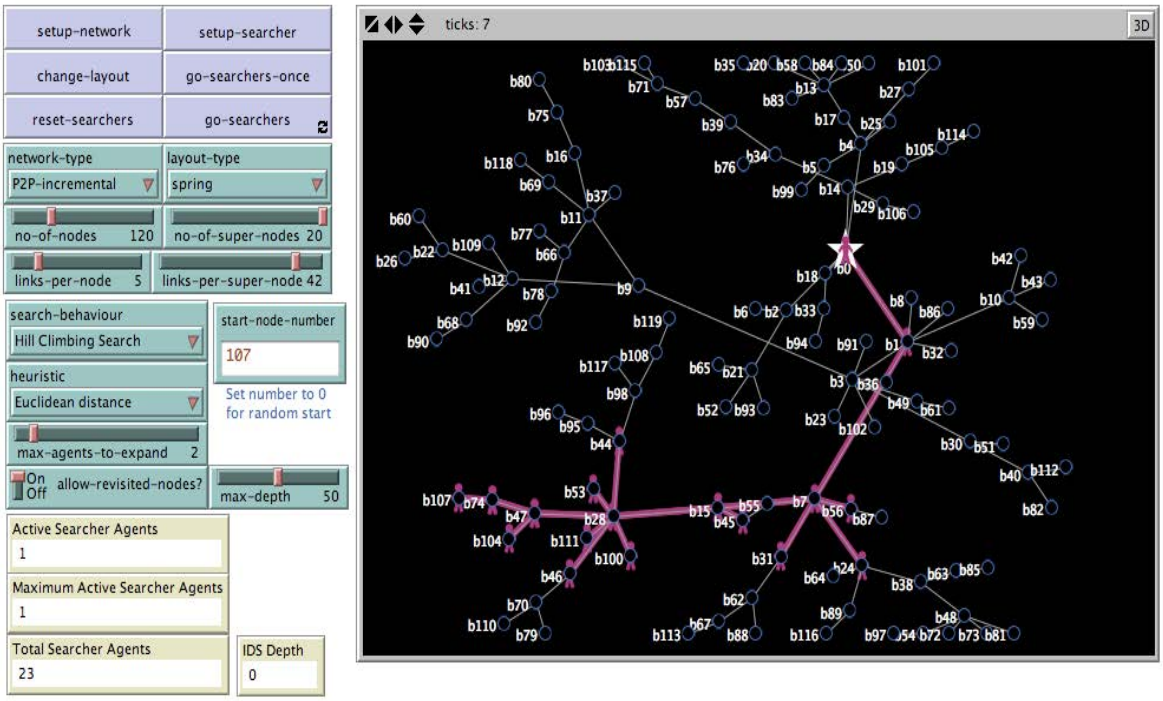


Figure 8.2.3. Screenshot of the Interface for the Searching for Kevin Bacon model for after the setup-network button has been pressed followed by the setup-searcher button and then the go-searchers-once button repeatedly, with the slider values as shown in the image.

WHAT IS IT?

This model applies standard search algorithms to the problem of searching for a specific goal node in a network. The problem is that you do not know where the node is, or how the network is configured. To solve this problem, search needs to be employed to try out the different paths that might lead to the goal node. This model implements some of the classic search algorithms and these can be applied to this toy problem to see how the different search strategies perform against each other.

WHAT IS ITS PURPOSE?

The purpose of this model is to show how some of the classic search algorithms such as breadth-first search and depth-first search can be implemented in NetLogo, and also how they can then be applied to solving a problem in order to compare how they perform.

HOW IT WORKS

The model implements the search algorithms in a novel way by making use of an agent-oriented approach which is relatively easy to implement in NetLogo. Rather than use a queue data structure to implement the classic search algorithms (this is a standard solution; see, for example, Russell & Norvig's AI textbook), the model instead adopts a purely agent-oriented solution where the information is distributed amongst NetLogo agents. These agents conduct the search by transferring information to other agents that continue the search. Hence, an explicit queue data structure separate from what is happening in the search is not needed (although technically the information is still being stored in a queue behind the scenes, but the implementation of this is hidden behind the top-level commands that NetLogo provides).

The hope is that this "queueless" agent-oriented approach provides a more intuitive solution that makes it easier to grasp how the search strategies work. It uses a situated, embodied first person design perspective, therefore arguably making it easier to understand the essential differences between the search strategies.

The breed of searcher turtle agents is used for the searchers that move throughout the network trying to get to the goal. These searcher agents maintain information about the current state of the search such as the time taken and the path and estimated costs. Each searcher agent expands the search by following all outgoing paths that lead out of the current node.

HOW TO USE IT

To initialise the search and create a random network (whose type is specified by the `network-type` chooser), press the `setup-network` button in the `Interface`. To setup a searcher agent to search the network, press the `reset-searchers` button first, then press the `setup-searcher` button. A turtle agent with a shape of a person will be drawn at the node specified in the `start-node` input box.

To start the search, either press the `go-searchers` once button to make the search proceed one step at a time, or press the `go-searchers` button to make the search proceed continuously until it reaches the goal node or it gets stuck.

THE INTERFACE

The model's `Interface` buttons are defined as follows:

- `setup-network`: This will clear the environment and all variables, and create a random network, whose type is specified by the `network-type` chooser. Its layout is specified by the `layout-type` chooser. The four sliders `no-of-nodes`, `no-of-super-nodes`, `links-per-node` and `links-per-super-node` control the number of nodes and super-nodes, and the number of links between them.
- `change-layout`: If the layout is of type `spring`, then this may help to remove some of the clutter.
- `reset-searchers`: This kills off all existing searchers and creates a new one as specified by the `start-node-number` Input box to restart the search.
- `setup-searcher`: This starts up a new searcher as specified by the `start-node-number` Input box (this can be used to create multiple searchers at the start of the search).
- `go-searchers-once`: This will make the search proceed one step at a time.
- `go-searchers-forever`: This will make the search proceed continuously until it has reached the goal node or it gets stuck.

The model's `Interface` choosers, sliders, switch and Input box are defined as follows:

- `network-type`: This selects the type of network that is created when the `setup-network` button is pressed. The types of networks are as follows:
  `"P2P-no-super-nodes"`: This simulates a peer-to-peer network with no super-nodes.
  `"P2P-has-super-nodes"`: This simulates a peer-to-peer network with super-nodes.
  `"P2P-random-single-link"`: This simulates a peer-to-peer network where each node has only one link with another random node.
  `"P2P-incremental"`: This creates the simulated peer-to-peer network by incrementally building the links to other random nodes one at a time.
  `"P2P-incremental-1"`: This creates the simulated peer-to-peer network by incrementally building links to other random nodes.
  `"Star-central-hub"`: This creates a network with one node (the Kevin Bacon node; i.e. the goal node) as the central hub and all other nodes linked to it and not to any other node.
  `"Hierarchical"`: This creates a tree network with the Kevin Bacon goal node at the root.
- `layout-type`: This specifies how the network should be laid out when it is visualised.
- `no-of-nodes`: This is the number of nodes to place in the network.
- `no-of-super-nodes`: This is the number of super-nodes to place in the network. (These are nodes that usually will have significantly more links than standard nodes as specified by `links-per-super-node` slider).
- `links-per-node`: This specifies the maximum number of links a standard node will have. The actual number chosen for a particular node will be a random number between 1 and this number.
- `links-per-super-node`: This specifies the maximum number of links a super-node will have. The actual number chosen for a particular super-node will be a random number between 1 and this number.
- `search-behaviour`: This specifies what search strategy the agents should employ when performing the search. A full description of the different search strategies can be found in Chapter 8 of the book "*Artificial Intelligence: Agent Behaviour I*".
- `start-node-number`: This is the node where the searcher agent is placed when it starts the search. Specifying 0 will mean that the searcher agent will be placed randomly in the network.
- `heuristic`: This specifies the heuristic to be used for the Informed searches – Greedy Best First Search and A* Search.
- `max-agents-to-expand`: This sets the maximum number of agents to expand the search each step. It is only used by the Multi-Agent Depth First Search.
- `allow-revisited-states?`: This is a flag that if set to `On` will allow the search to proceed to already visited states.
- `max-depth`: This sets the maximum depth of the search, for the depth-limited search and the Iterative Deepening Search only.

The model's `Interface` monitors are defined as follows:

- `Active Searcher Agents`: This shows the current number of active searcher agents still participating in the search.
- `Maximum Active Searcher Agents`: This shows the maximum value that the `Active Searcher Agents` monitor has achieved throughout the search.
- `Total Searcher Agents`: This shows the total number of searcher agents that the search has used.
- `IDS Depth`: This is the current depth during the execution of the Iterative Deepening Search.

THINGS TO NOTICE

Notice the effect of turning the flag `allow-revisited-states?` `On` and `Off`. Why is turning it `Off` so effective at dramatically reducing the search for the different search behaviours?

Notice that some of the searches if repeated with the same settings end up in different places. Why is this?

Notice that many of the behaviours result in the searching turtle agents getting stuck. When does this happen, and why?

THINGS TO TRY

Which search behaviour seems to be the best at this problem? Which seems to be the worst? Try out each of the different search behaviours to see which one is the most effective. Find out how well each search algorithm performs against the four evaluation criteria – time complexity, space complexity, optimality and complexity. Relate the theory to what happens in practice.

Try out all the different types of networks with different slider settings. Which types of network cause problems for the various searches, and which do not? i.e. Which seem to be more difficult to search?

Try switching the searching behaviour mid-stream. For example, try switching from a breadth-first search to a depth-first search then back again.

Try out the different heuristics for the informed searches (Greedy Best First Search and A* Search). Do they have any effect on how effective the search is?

EXTENDING THE MODEL

Try adding other search algorithms, or adding your own variations to the ones implemented in the model.

Try adding your own type of network, either randomly created or with a correspondence to a network in real-life. You will need to add input routines to create the network layout for the latter.

NETLOGO FEATURES

Note the use of the `hatch` and `die` commands to clone child searchers and terminate parent searchers thus avoiding the need for a separate queue data structure to maintain the information during the search. Note also the variation in the `ask` command that determines the strategy being used that defines the different searches.

RELATED MODELS

For other search problems, see the Being Kevin Bacon model, the Missionaries and Cannibals model, and the Searching Mazes model. For an insight into the Manhattan distance and Euclidean distance metrics used as heuristics for the Informed searches, see the Manhattan Distance model.

Exercise 8.2.4: Searching for Kevin Bacon 2 NetLogo Model

Try out the Searching for Kevin Bacon 2 model in NetLogo:

Searching for Kevin Bacon 2	http://files.bookboon.com/ai/Searching-for-Kevin-Bacon-2.html

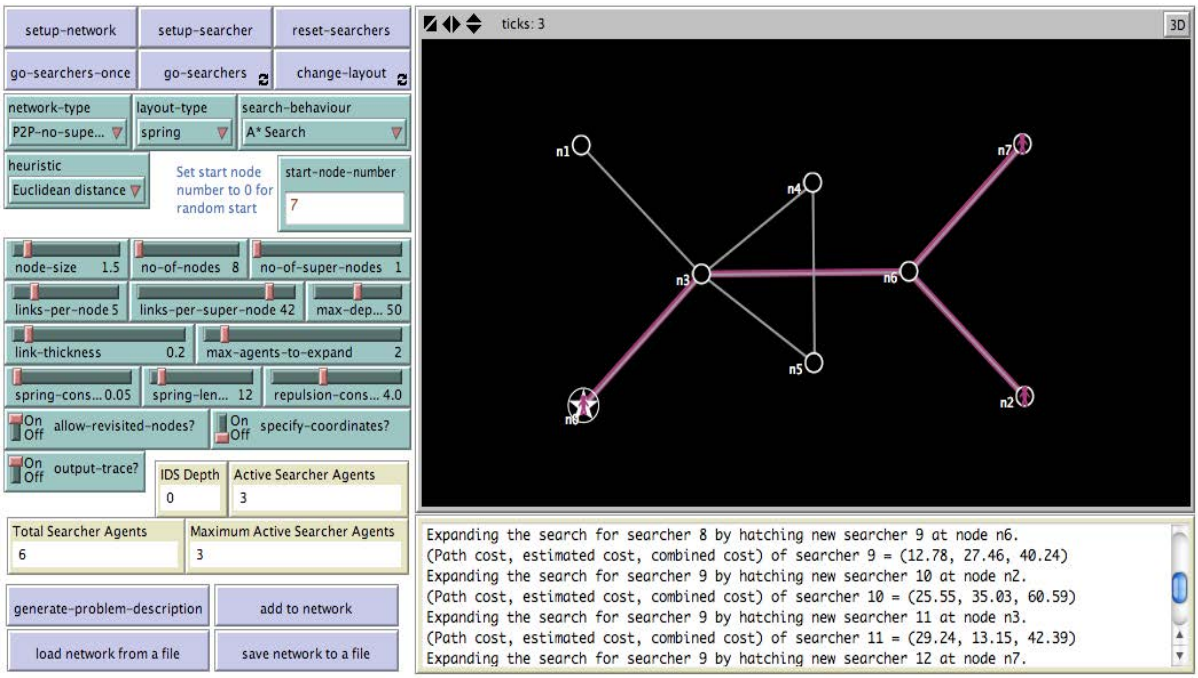


Figure 8.2.4. Screenshot of the Interface for the Searching for Kevin Bacon 2 model for after the setup-network button has been pressed followed by the setup-searcher and go-searchers buttons, with the slider values as shown in the image. The screenshot shows a small network with 8 nodes, with the Output box showing a trace of the search.

The `Information` for this model is the same as the `Information` for the Searching for Kevin Bacon model detailed in Exercise 8.2.3, except for the following:

WHAT IS IT?

This NetLogo model is an extension to the Searching for Kevin Bacon model that has an `Output` that provides an English description of the search problem, and provides a trace of how the search proceeds. Specific networks can also be saved to a file, and loaded latter into the application.

As with the Searching for Kevin Bacon model, this model applies standard search algorithms to the problem of searching for a specific goal node in a network.

HOW TO USE IT

To initialise the search and create a random network (whose type is specified by the `network-type` chooser), press the `setup-network` button in the `Interface`. To create your own network, set the `network-type` chooser to "`User-specified`". If the `specify-coordinates?` slider value is set to `Off`, then the user can then use the mouse to click on the points in the environment where they want the network nodes located. (The first point clicked will be designated as the goal node i.e. the Kevin Bacon node.) If the slider is set to `On`, then the user will be prompted for the *x* and *y* co-ordinates of the nodes. For each of the nodes in the network, the user will then be asked how many links there are from the node and which nodes link to it.

To save the network to a file, press the `Save Network to a File` button. To reload it at a letter date, press the `Load Network from a File` button. To generate a problem description (that could be used for an exam or test question, for example), then press the `Generate Problem Description` button. This will write the description into the `Output` box shown on the middle-right bottom of the `Interface`. To add further nodes to the network, press the `Add to Network` button.

THE INTERFACE

The model's `Interface` buttons are defined as follows:

- `Generate Problem Description`: This will generate a description in English of the search problem and write it into the `Output` box. The problem description is based on the values of the choosers and sliders in the `Interface`. It can be used for setting an exam or test question, or set an exercise for an assignment as course work.
- `Add to Network`: This will allow the user to add nodes to the network, including networks that have been generated randomly.
- `Load Network from a File`: This will load a network layout that has been previously saved to a file on disk.
- `Save Network to a File`: This will save the network layout to a file on disk. This can be loaded at a latter date using the `Load Network from a File` button.

The model's Interface sliders are defined as follows:

- `link-thickness`: This sets the thickness of the links that are drawn between the nodes in the network.
- `spring-constant`, `spring-length`, `repulsion-constant`: These sliders are used by the `change-layout` algorithm when the `layout-type` slider is set to `spring` and can be used to control how the network looks. For example, setting the `spring-length` to 0, then increasing it, will often unclutter the layout of the network.

The model's Interface switches are defined as follows:

- `specify-coordinates?`: If this switch is set to `On` when the `network-type` slider is set to `User-specified`, then the user has to specify the precise co-ordinates of each node rather than using the mouse to point and click on a location.
- `output-trace?`: If this switch is set to `On`, then a trace of the search is written to the `Output` box (this is located to the middle-right at the bottom of the `Interface`).

THINGS TO TRY

Try to follow how the informed searches, Greedy Best First and A*, search the various networks. Trace the searches as they proceed by turning the `output-trace?` switch to `On`. Work out how the path cost and estimated cost are calculated at each stage of the search. Try out the different heuristics. Do they have any effect on how effective the search is?

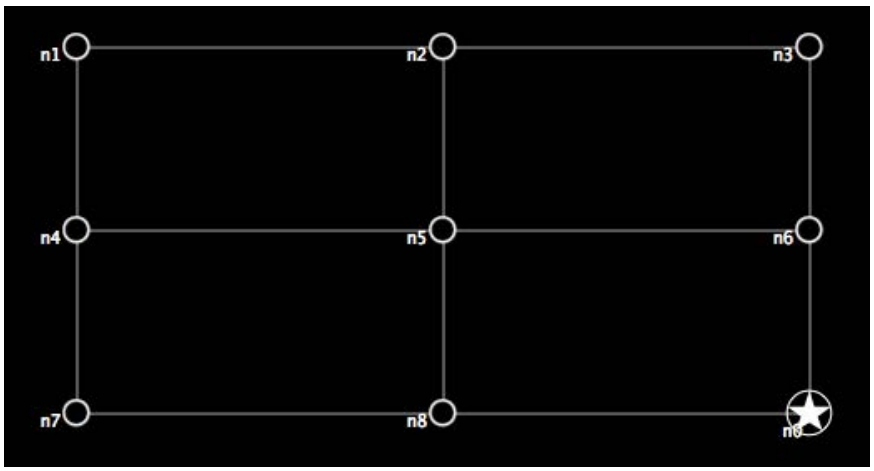
RELATED MODELS

For other search problems, see the Being Kevin Bacon model, the Missionaries and Cannibals model, Searching for Kevin Bacon and the Searching Mazes model. For an insight into the Manhattan distance and Euclidean distance metrics used as heuristics for the Informed searches, see the Manhattan Distance model.

8.3 Uninformed (blind) search

Exercise 8.3.1:

Show how the Breadth First Search and Depth First Search algorithms search the network shown below. Show the list of active searcher agents along with the nodes they are visiting. The start node is node `n1` and the star node is the goal node (i.e. node `n0`). Expand the nodes in the order of their node number. For example: expand node `n1` before every other node; expand node `n2` before every other node except node `n1`; and so on. Also, allow nodes to be revisited.



Repeat this exercise for the following uninformed search algorithms:

- Uniform Cost Search;
- Multi-agent Depth First Search;
- Depth Limited Search;
- Iterative Deepening Search.

8.4 Implementing uninformed search in NetLogo

Exercise 8.4.1:

Compare how the `expand-path` and `expand-paths` procedures are defined for the three different models – the Searching Mazes model, the Missionaries and Cannibals model, and the Searching for Kevin Bacon model. How do the differences in the way they are defined reflect the nature of the search problem? How do they reflect the way that multiple paths occur leading to different outcomes? Is there some way that these procedures can be made more generic and less specific to the problem?

8.5 Search as behaviour selection

Exercise 8.5.1:

Observe situations where search is being employed by a human or an animal, including yourself. Do you think these searches are best characterized as a choice between alternative sequences of actions or as a choice between alternative sequences of behaviours? Or is another characterisation more appropriate? Observe the process you employ when you make this decision.

Exercise 8.5.2:

As pointed out in Section 8.5 of the companion *Artificial Intelligence – Agent Behaviour I* book, in the Searching Mazes model, in order to illustrate the behavioural approach to characterising the search process, the agents have been given the ability to execute three different reactive behaviours – (i) moving forward, (ii) turning left then moving forward, and (iii) turning right then moving forward. The agents can also execute several different types of forward movement behaviour. These are controllable by two choosers in the Interface: [1] the `move-forward-behaviour`; and [2] the `move-forward-behaviour-after-turning`. The first defines the forward movement for reactive behaviour (i), and the second defines the forward movement for the reactive behaviours (ii) and (iii). The types of forward movement are as follows: (a) "`Move forward n steps unless hit wall`", where the agent moves forward *n* steps, and *n* is a number defined by the Interface variable `move-forward-step-amount`; (b) "`Move forward until hit wall`", where the agent will keep on moving forward until it hits a wall; (c) "`Move forward until any open space`", where the agent will move forward until there is a wall in the way or there is open space on both sides of it; and (d) "`Move forward until open space after side wall`", where the agent will move forward until there is a wall in the way or there is open space after a side wall.

Despite only being able to execute three basic reactive behaviours, the maze-searching agents can exhibit a surprising variety of movements as a result. Figure 8.10 captures a few of these in screenshots taken from the Searching Mazes model using breadth first search on the empty maze. Referring to the numbers and letters in the previous paragraph to describe the four configurations: in the top left image, the agents apply behaviours [1](a) and [2](a), both using a step of 10; in the top right image, the behaviours are [1](a), using a step of 1, and [2](b); in the bottom left image, the behaviours are [1](a), using a step of 10, and [2](c); and in the bottom right image, the behaviours are [1](c) and [2](a), using a step of 10. For example, the grid effect in the top left image is caused by the agents repeatedly moving 10 steps in three directions – forward, left and right. How the grid is built up is best seen by continually pressing the `go-once` button in the Interface at the start of each search.

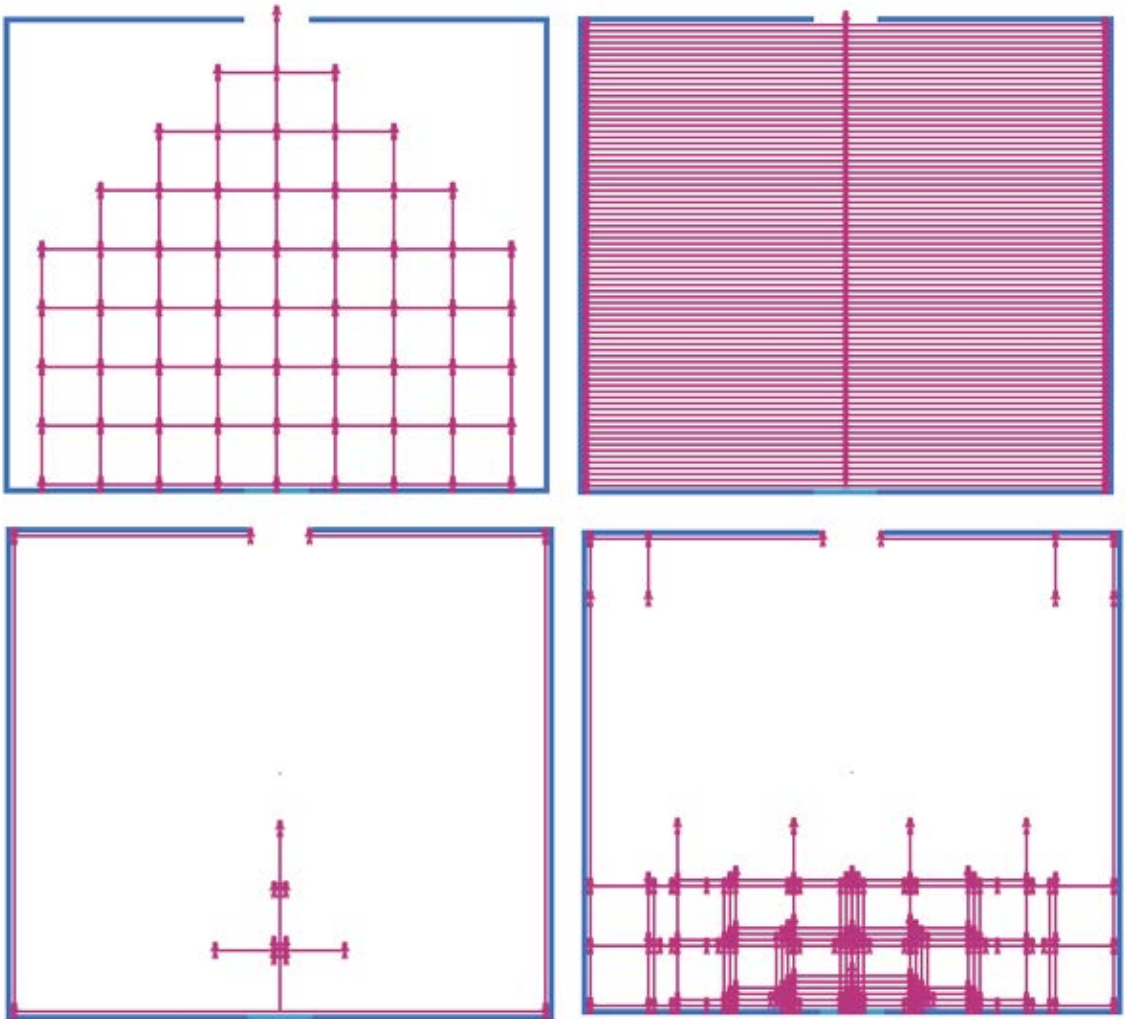


Figure 8.5.2 Screenshots of different movement behaviours for the Searching Mazes model.

For this exercise, try reproducing the turtle movement behaviour for the Empty Maze as illustrated in Figure 8.5.2. Then try creating your own patterns by changing the values of the choosers and sliders.

8.6 Informed search

Exercise 8.6.1: Manhattan Distance Model

Try out the Manhattan Distance Model in NetLogo:

Manhattan Distance	http://files.bookboon.com/ai/Manhattan-Distance.html

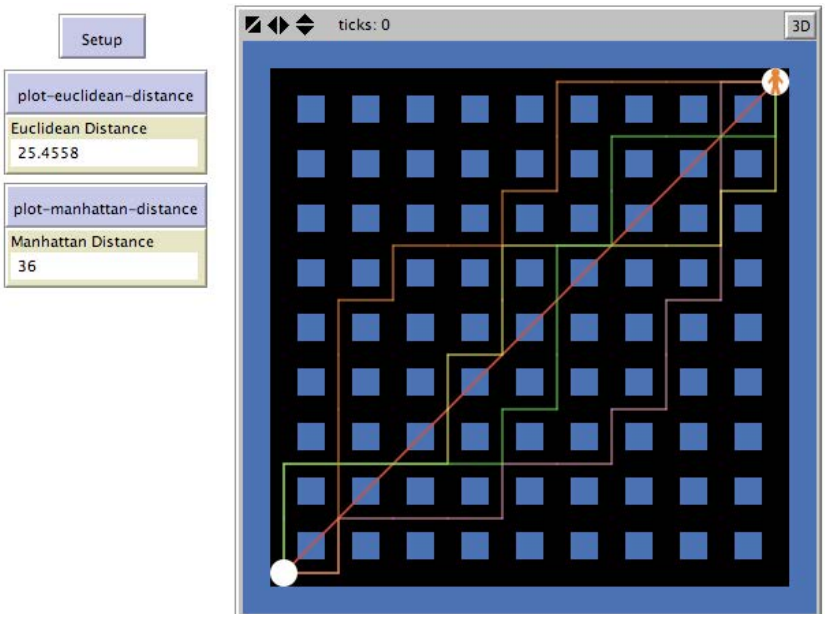


Figure 8.6.1. Screenshot of the Interface for the Manhattan Distance model for after the Setup button has been pressed followed by the plot-euclidean-distance button, and then the plot-manhattan-distance button four times in succession.

WHAT IS IT?

This model shows how Manhattan distance is calculated. This is compared with Euclidean distance in the model.

WHAT IS ITS PURPOSE?

The purpose of this model is to highlight via basic animation techniques the fundamental differences in the two different distance calculations.

HOW IT WORKS

Two `point` agents are used to designate the start and end points. A `walker` agent is used to go between the two points. This is done either in a direct line to illustrate what the Euclidean distance metric is based upon. Or for the Manhattan distance metric, it is done through any of the paths that avoid crossing into the Manhattan "skyscrapers" represented by the blue boxes in the environment. In other words, the paths can only go down either an avenue (roads that run north/south) or street (roads that run west/east).

HOW TO USE IT

First press `Setup` to reset the agents and environment. Then press the `plot-euclidean-distance` button if you wish to show the path the Euclidean distance metric (shown in the Euclidean Distance monitor) is based upon. Or press the `plot-manhattan-distance` button if you wish to plot one of the paths the Manhattan distance metric (shown in the Manhattan Distance monitor) is based upon. Pressing this button again will plot a different path, but with the same value shown in the monitor.

THE INTERFACE

The model's `Interface` buttons are defined as follows:

- `Setup`: This resets the model.
- `plot-euclidean-distance`: This draws the straight line path for which the Euclidean distance calculation is based upon.
- `plot-manhattan-distance`: This draws one of the paths for which the Manhattan distance calculation is based upon.

The model's `Interface` monitors are defined as follows:

- `Euclidean Distance`: This is the Euclidean distance between the two points.
- `Manhattan Distance`: This is the Manhattan distance between the two points.

THINGS TO NOTICE

Notice that Euclidean distance is always less than Manhattan distance. Why?

Both Euclidean distance and Manhattan distance are admissible heuristics if used for an informed search algorithm such as greedy search or A* search. Why?

THINGS TO TRY

Keep plotting the different Manhattan paths by repeatedly pressing the `plot-manhattan-distance` button. A visually appealing pattern will emerge. Some of the paths will take longer to appear, however. Why?

EXTENDING THE MODEL

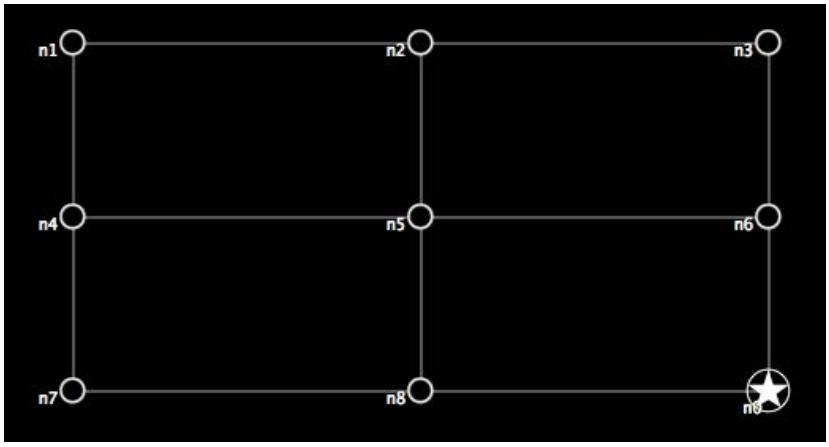
Consider how to visualise the distance calculations in 3 or more dimensions.

RELATED MODELS

See the Searching Mazes, Missionaries and Cannibals and Searching for Kevin Bacon models to see where these distance metrics become useful as heuristic functions in various informed search algorithms.

Exercise 8.6.2:

Show how the Greedy Best First Search and A* Search algorithms search the network shown below. Show the list of active searcher agents along with the nodes they are visiting. The start node is node `n1` and the star node is the goal node (i.e. node `n0`). Expand the nodes in the order of their node number. For example: expand node `n1` before every other node; expand node `n2` before every other node except node `n1`; and so on. Also, allow nodes to be revisited.

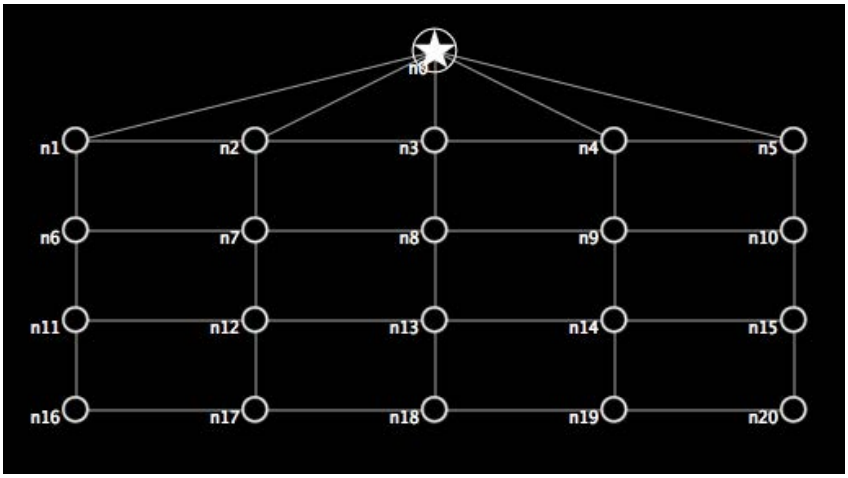


How do these informed searches compare to the way uninformed search algorithms such as Breadth First Search and Depth First Search search the network? (See Exercise 8.3.1).

8.7 Local search and optimisation

Exercise 8.7.1:

Show how the Hill Climbing Search searches the network shown below. Show the list of active searcher agents along with the nodes they are visiting. The start node is node `n16` on the lower left corner and the star node is the goal node (i.e. node `n0`). Expand the nodes in the order of their node number. For example: expand node `n1` before every other node; expand node `n2` before every other node except node `n1`; and so on. Also, allow nodes to be revisited. Use Manhattan Distance as the heuristic for the problem.



Compare how the informed search algorithms, Greedy Best First Search and A* Search, perform on the same problem.

8.8 Comparing the search behaviours

Exercise 8.8.1:

Have a play with the Searching Mazes, Missionaries and Cannibals and Searching for Kevin Bacon NetLogo models to try out the different search algorithms that have been implemented in each model:

- Breadth First Search
- Uniform Cost Search
- Depth First Search
- Multi-agent Depth First Search
- Depth Limited Search
- Iterative Deepening Search
- Greedy Best First Search
- A* Search
- Hill Climbing Search

Try different settings to see what happens. Select the different types of search by selecting the appropriate option using the `search-behaviour` chooser. For the informed searches – Greedy Best First Search, and A* – you will also be able to select from one of three heuristics ('`Zero`', '`Euclidean Distance`' and '`Manhattan Distance`') with two further heuristics ('`People on the left side`' and '`Min. no. boat trips needed`') available for the Missionaries and Cannibals model.

For the Searching Mazes model, first set the `maze-being-searched` chooser in the `Interface` to `Empty Maze`. Also find out what happens when you select the Hampton Court maze or the Chevening House maze. Explain your results.

Exercise 8.8.2:

Use the BehaviorSpace tool to find out which of the different search algorithms work best for the following NetLogo models: Searching Mazes; Missionaries and Cannibals; and Searching for Kevin Bacon. (Do this in a more systematic manner than for Exercise 8.8.1. See Exercise 7.6.1 for more information on how to use the BehaviorSpace tool).

When filling in the `Experiment` dialog in the BehaviorSpace tool, you can use the `total-searchers` reporter for measuring the runs for all three models (i.e. enter the text '`total-searchers`' into the '`Measure runs using these reporters`' field). You will also have to remove the various user messages in the three models, such as "`Search is completed!`", "`No more searchers! Abort!`", "`Made it to the centre of the maze`" and "`Found Kevin Bacon!`", replace them with reporters and use these to specify the `Stop condition` field for the experiment. For the Searching Mazes model and the Missionaries and Cannibals model, use the `setup` and `go` procedures to fill in the `Setup commands` and `Go commands` fields. For the Searching for Kevin Bacon model, use the `setup-network` and `go-searchers` procedures as the setup and go commands instead.

9 Knowledge

9.1 Knowledge and Knowledge-based Systems

Exercise 9.1.1:

As pointed out in Exercise 1.1.1 of the book Artificial Intelligence – *Agents and Environments*, one of the issues for Artificial Intelligence research concerns the problem of categorization for intelligent systems – the problem of how to select suitable categories to cover a set of examples, and the resultant classification errors that arise once a particular set of categories has been chosen.

As a further example, use your favourite search engine to find out how the terms 'Knowledge' and 'Knowledge-based System' are defined by different people. Classify the various definitions you find into a set of different categories. Which definitions fit well into your set of categories? Which definitions pose problems that require a re-evaluation of the suitability of your taxonomical classification?

Exercise 9.1.2:

What must a knowledge-based system have that is different to a database system in order for it to become a repository of "knowledge" rather than just a repository of data?

Your answer must include a discussion of the following claim: "To create a knowledge based system, all you need to do is add some rules to a database system to infer new data."

9.2 Knowledge as justified true belief

Exercise 9.2.1:

Discuss the following statement: "Knowledge cannot exist by itself, independent of us. It only exists in our heads".

Exercise 9.2.2:

Discuss the standard definition of knowledge that states that knowledge is "justified true belief". What types of knowledge are well suited to being characterized in this manner? What types of knowledge pose problems for this definition?

9.3 Different types of knowledge

Exercise 9.3.1:

Label each of the following as being declarative knowledge, procedural knowledge, task knowledge, behavioural knowledge, episodic knowledge, explanatory knowledge or inferred knowledge:

- the knowledge that Margaret Thatcher is still the current prime minister of Great Britain;
- the knowledge that the Titanic sank on April 15th, 1912;
- the knowledge of how to cook a roast meal;
- the knowledge of how to drive from Bangor to Betwys-y-Coed in North Wales;
- the knowledge of when you last went to a wedding or a funeral;
- the knowledge for a robot of how to lift its robotic arm;
- the knowledge that there are choices to be made when you are at a junction in a maze;
- the knowledge that if a Takehe lays an egg, then it must be a bird;
- the knowledge that we cannot be in two places at the same time;
- the knowledge required to solve a mathematical equation;
- the knowledge of which wild mushrooms are poisonous;
- the knowledge that the mother of Jesus is Mary;
- the knowledge of how to open a tin of spaghetti;
- the knowledge of what a Pohutukawa tree looks like;

- the knowledge of what bagpipes sound like;
- the knowledge of what haggis tastes like;
- the knowledge of how it feels to stroke a cat;
- the knowledge of what your face looks like;
- the knowledge that Elvis is still alive;
- the knowledge of what strong coffee smells like when hung-over;
- the knowledge of how to pronounce the word "Llangollen" correctly;
- the knowledge of when someone is being polite or being rude;
- the knowledge of what might happen when you do something wrong;
- the knowledge of how to bear oneself when in the presence of Her Majesty, the Queen;
- the knowledge required to explain the meaning of life, the universe and everything;
- the knowledge that the sun will explode at the end of this century (in the year 2099);
- the knowledge that global warming is causing irreversible damage to the Earth's environment;
- the knowledge that you think, are self-aware and are conscious.

Pohutukawa Tree

Which of these do not fit well in any of the categories? What problems arise when trying to represent the knowledge encompassed by each of these in a computer system?

9.4 Some approaches to Knowledge Representation and AI

Exercise 9.4.1:

Compare and contrast the following approaches to knowledge representation:

- symbolic;
- connectionism (non-symbolic);
- conceptual spaces.

Exercise 9.4.2:

Discuss for each of the three approaches listed in Exercise 9.4.1 above how you might design the knowledge to describe the following:

- the sound of bagpipes (e.g. to a deaf person);
- the taste of milk;
- what sandpaper feels like;
- what coffee smells like;
- what your face looks like (e.g. to a blind person).

For each approach, indicate the shortcomings in the solution you chose.

Exercise 9.4.3: Colour Cylinder NetLogo Model

Try out the Colour Cylinder model in NetLogo:

Colour Cylinder	http://files.bookboon.com/ai/Colour-Cylinder.html

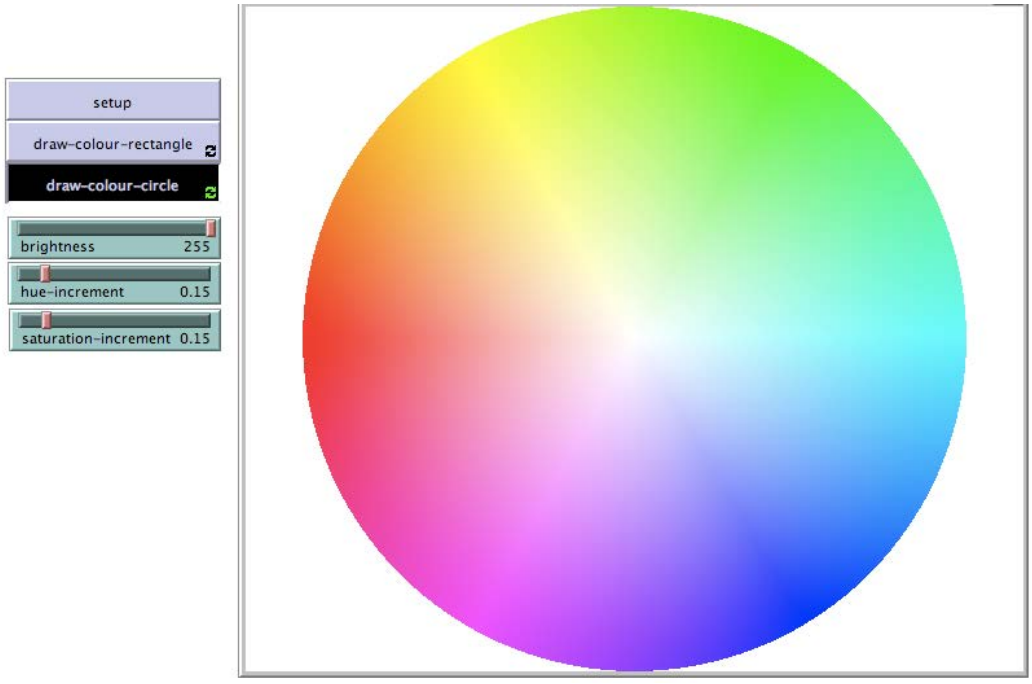


Figure 9.4.3. Screenshot of the Interface for the Colour Cylinder model after the setup button has been pressed followed by the draw-colour-circle button.

WHAT IS IT?

This model demonstrates how colour can be represented in 3 dimensions: hue, saturation and brightness. These correspond to the dimensions that experiments indicate humans use to perceive colour.

WHAT IS ITS PURPOSE?

The purpose of this model is to show that if humans rely on three dimensions to perceive colour (hue, saturation and brightness), then it is not possible to exactly define precisely the concept of a particular colour (such as red or yellow) as it is not possible to define the boundaries between the different colours shown in the colour spindle or cylinder.

HOW IT WORKS

A `drawer` agent is used to draw the colour circle or rectangle. Two of the colour dimensions – hue, and saturation – determine what the colour circle looks like for a specific value of brightness. Peter Gärdenfors' work with conceptual spaces theory talks about how humans perceive colour using a colour spindle – where the radius of the circle gets progressively smaller as the brightness is increased or decreased, eventually ending up at points for when the brightness is maximum or minimum. In the model, this is when the brightness is set to 255 and 0, and corresponds to the colours white and black, respectively.

As the model's visualisation is limited to 2D, the model depicts a colour cylinder instead of a colour spindle, with each colour circle representing a slice of the cylinder for a specific brightness value.

HOW TO USE IT

Press the `setup` button first. Then press either the `draw-colour-rectangle` or `draw-colour-circle` button next. The brightness slider determines the brightness of the colours that are drawn. The `hue-increment` and `saturation-increment` sliders can be used to control how fine the colour is drawn.

THE INTERFACE

The model's `Interface` buttons are defined as follows:

- `setup`: This clears everything, makes the background white, and creates a `drawer` agent ready to draw the colour circle or rectangle.
- `draw-colour-rectangle`: This draws a colour rectangle for a specific brightness value. The hue is the *x* dimension and the saturation is the *y* dimension.
- `draw-colour-circle`: This draws a colour circle for a specific brightness value. The hue values are spread out around the 360 degrees of the circle, and the saturation values are spread out along various radii lengths from 0 to 255 from the centre of the circle.

The model's `Interface` sliders are defined as follows:

- `brightness`: This controls the brightness of the colour drawn.
- `hue-increment` and `saturation-increment`: This controls the increments used when drawing the colours. Smaller increments will mean that the drawing has finer detail.

THINGS TO NOTICE

Notice how difficult it is to define the region for a specific colour. For example, try drawing an imaginary line around everything that is yellow.

THINGS TO TRY

Try using the brightness slider to change the brightness value to see what effect it has on the colour circle and colour rectangle. Change the values of the hue-increment and saturation-increment sliders to change how fine an increment is used to draw the colour circle. Increasing the size of the increments will result in symmetrical patterns being drawn. Why is this? Try doing this while the circle is being drawn.

You can draw a border to the circle by modifying the following code:

```
[ set pcolor hsb-as-rgb ]
; set this to a specific color if you want to draw a border
; to the circle
```

Change "`hsb-as-rgb`" in the above code to the colour you want for your border. For example, if you want it to be black, set it to 0, if you want it to be blue, set it to 105.

EXTENDING THE MODEL

Have a go at adding labels to the colour circle such as red, yellow and green. Alternatively, set up a HubNet activity where each student chooses where to place colour labels. See how well they match across the classroom.

NETLOGO FEATURES

It uses the NetLogo command `hsb` to generate the patch colours based on the hue, saturation and brightness values. The `patch-at-heading-and-distance` command is used to draw the colour circle.

9.5 Knowledge engineering problems

Exercise 9.5.1:

List two or more real-life examples for the following types of problems for intelligent systems:

- classification;
- clustering;
- control;
- diagnosis;
- optimisation;
- prediction;
- selection.

How can knowledge be leveraged by an intelligent system for these types of problems? Is the nature of these problems inherently different requiring that knowledge is processed in a fundamentally different way, or is knowledge processed in a similar manner?

9.6 Knowledge without representation

Exercise 9.6.1:

What do we mean by 'representation' when we talk about 'knowledge representation'? Is it fair to say that the ant and termite agents in the Ants and Termites models do not 'represent' knowledge in some manner? (After all, the knowledge is represented globally in the entire multi-agent system, but not individually within each agent.)

(See also Exercise 10.2.1.)

9.7 Representing knowledge using maps

Exercise 9.7.1: Map Drawing NetLogo Model

Try out the Map Drawing model in NetLogo:

Map Drawing	http://files.bookboon.com/ai/Map-Drawing.html

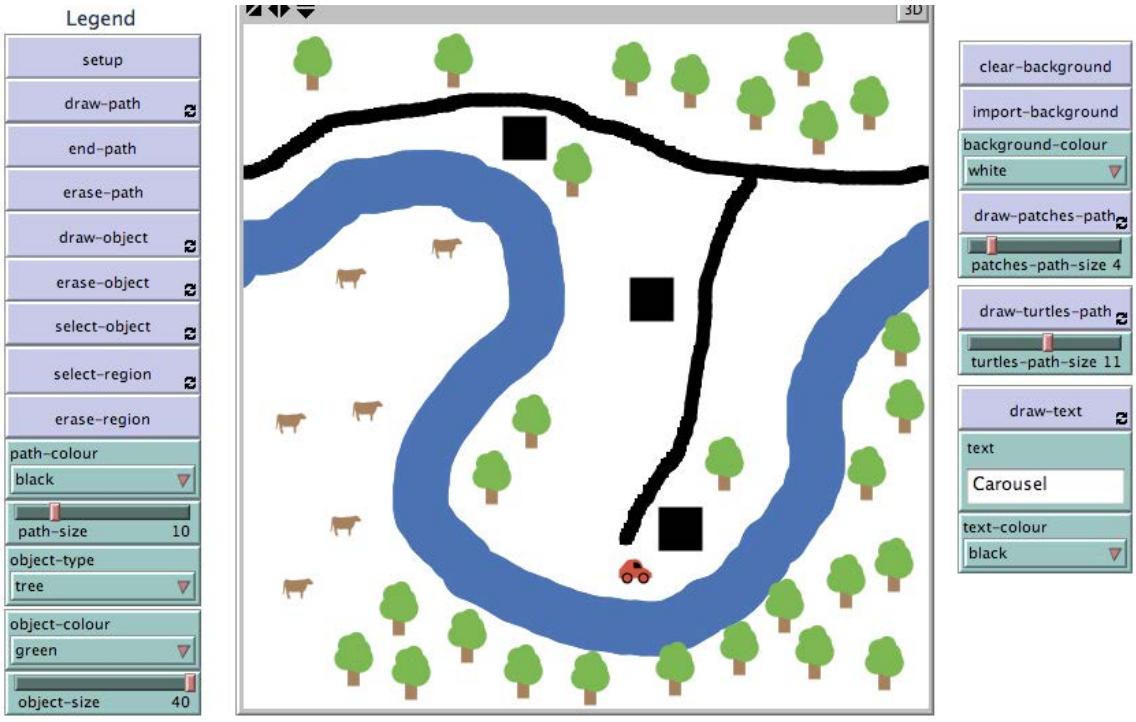


Figure 9.7.1.1 Screenshot of the Interface for the Map Drawing model with a sample map drawn in the environment.

WHAT IS IT?

This model allows the user to draw a map. e.g. A map of Central Park in New York.

This model was created using NetLogo version 4.0.4. Although it works perfectly fine when converted to NetLogo version 4.1, a run-time error occurs when you try to import the Map Drawing Central Park World into the latter version using the CSV file that was exported using the earlier version. A simple work-around to this problem is to keep running the model using version 4.0.4 rather than 4.1.

If you would like to load the Central Park Map shown in Chapter 9 of the Artificial Intelligence Book, first ensure you are running the model in version 4.0.4, then select the `Import` option from the `File` tab, select the `Import World` option, then import the Map Drawing Central Park World file: `Map-Drawing-Central-Park-World.csv`.

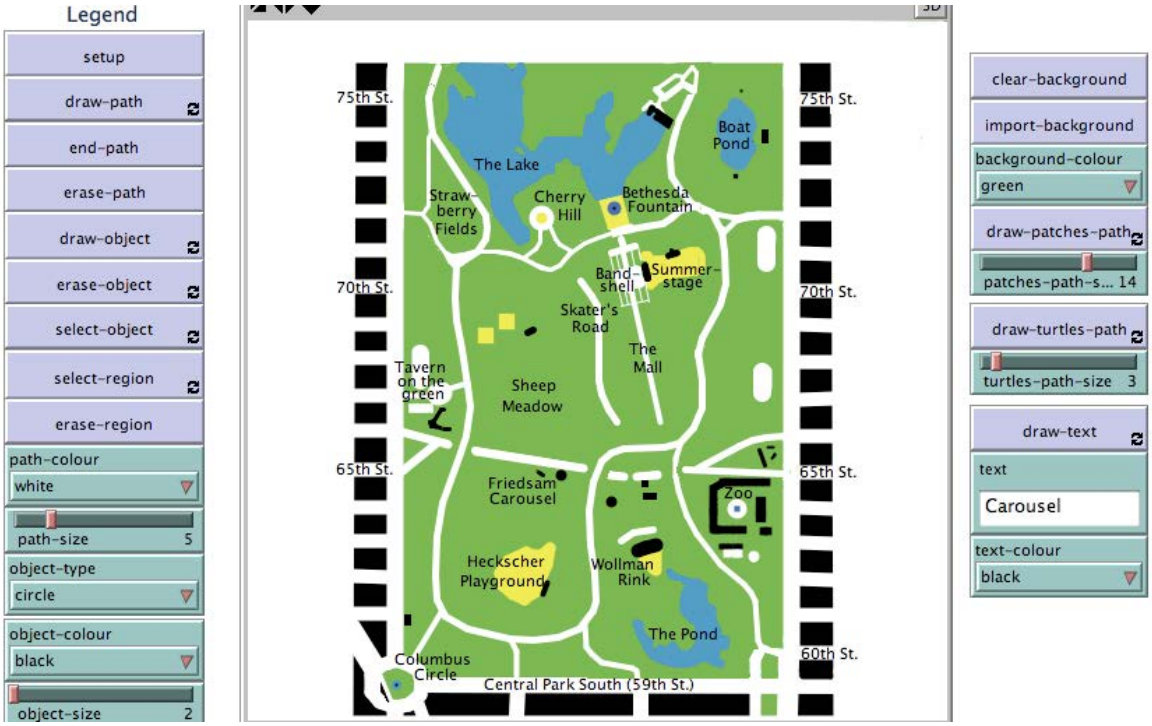


Figure 9.7.1.2. Screenshot of the Interface for the Map Drawing model with the Central Park map loaded into the environment.

WHAT IS ITS PURPOSE?

The purpose of this model is to provide a facility that allows users to draw their own maps, thereby allowing them to represent knowledge about a real environment in the virtual 2D NetLogo environment. Maps can be considered a form of knowledge representation which provides a different approach with different capabilities for reasoning in comparison to the more traditional forms of knowledge representation such as first order logic and decision trees usually described in AI textbooks.

A secondary purpose is to show the different ways that objects and paths can be drawn and edited using NetLogo's features.

HOW IT WORKS

The model allows the user to place various objects such as circles, squares, triangles, trees, and persons in the environment. These objects are represented by different breeds of turtle agents.

The paths a user can draw on the map have different types. The `draw-path` command draws a path using the turtle draw commands (i.e. using the drawing turtle's pen via the `pen-down` command). As a consequence, this type of path cannot be "seen" by other agents in the environment, so the `select-region` command does not work on this type of path, and it cannot be erased using the `erase-path` if the path has been ended (for example, by the `end-path` command). In this case if you do not wish to erase everything by pressing the `setup` command, then the best strategy is to draw over the top of the path with another path of a different colour to the desired background colour.

There are two other types of paths – these are paths drawn using patch agents (using the `draw-patches-path` command) and using turtle agents (using the `draw-turtles-path` command). Unfortunately, the quality of these types of paths can be very patchy (excusing the pun). Only the turtle agents type of path can be selected, then moved or erased. The patch type of path cannot be selected unlike the path drawn using the turtle's pen described above.

The model uses `drawer` turtle agents for drawing the paths, `tracer` turtle agents for tracing them, `text` turtle agents for drawing the text and `square-drawers` for drawing a path using square turtles.

HOW TO USE IT

Press the `setup` button first. To place an object in the map, first select the desired `object-type` and `object-colour`, press the `draw-object` button, then place the object at the desired location by moving the mouse and then clicking. You can erase the object by pressing the `erase-object` button then clicking on it with the mouse. To move it, press the `select-object` button, then when it is highlighted using a halo, the object can be moved around using the mouse. The `select-region` button allows the user to select a region of objects to be moved around using the mouse or erased using the `erase-region` button.

If you wish to draw a path, there are three options: paths drawn using a turtle agent's pen, paths drawn using patch agents and paths drawn using turtle agents. The first option comes out looking smoother; the other two options can end up being patchier. When drawing the path, move the mouse to the desired location and then click and drag to draw the path.

To draw a path using a turtle agent's pen, first choose the desired `path-colour` and `path-size`, then press the `draw-path` button. The path cannot be erased once it has been ended either when `end-path` has been pressed or another object or path has been drawn elsewhere. If this is not the case, then `erase-path` will work by erasing each point in the path one by one. If `erase-path` is not successful at deleting the just-drawn path, then the only way to "remove" it is to draw over the top of it.

To draw a path using patch agents, select the desired `patches-path-size`, then press the `draw-patches-path` button.

To draw a path using turtle agents, select the desired `turtles-path-size`, then press the `draw-turtles-path` button.

Text can be added by the `draw-text` button. Before doing so, make sure that you type in the text you want to draw into the `text` input box first, then select the desired `text-colour`. The font size for all text is specified globally using the `Settings` button in the `Interface`.

The colour of the background for the map is specified using the `background-colour` chooser. The background is changed/cleared using the `clear-background` button. An image can be used for the background instead of a single colour by pressing the `import-background` button.

To save the entire map and all the settings to disk, select the `File` menu from the NetLogo menu bar, then select `Export World` option. To load it back in again at a latter date, use `Import World` instead.

THE INTERFACE

The model's `Interface` buttons are defined as follows:

- `setup`: This clears the map and resets the background.
- `draw-path`: This draws a path by having a turtle agent draw with the pen down between successive points as the mouse is dragged around. The path colour is specified by the `path-colour` chooser, and the path size is specified using the `path-size` slider.
- `end-path`: This designates that the path is ended. A subsequent press of the `erase-path` button will not be successful in deleting previously drawn paths. The only way to get rid of them is to draw over the top of them with another colour.

- `erase-path`: This will erase the points one by one in a path that has just been drawn using the `draw-path` button, unless that path has been designated as complete using the `end-path` button, or an intervening object has been drawn (which ends the path by default).
- `draw-object`: This will draw an object at the location specified by the next mouse click. The object type and colour is specified by the `object-type` and `object-colour` choosers, and its size by the `object-size` slider.
- `erase-object`: This will erase the nearest object if there is one. Make sure to click on the centre of the object, otherwise it will not be erased.
- `select-object`: This will allow the user to drag a selected object using the mouse in order to change its position. The selected object is shown with a halo. Make sure to click on the centre of the object, otherwise a halo showing it has been selected will not appear.
- `select-region`: This allows the user to select a rectangular region of objects for moving around. The user clicks and drags the mouse to define the region being selected (this is shown with a red box dynamically as the mouse is dragged; all objects that can be moved in the region are also highlighted). To move the selected objects around, lift the mouse momentarily when the rectangular region is as you want it, click back inside the red rectangle then drag it to the desired location.
- `erase-region`: This will erase all the objects that have been selected using the `select-region` button.
- `clear-background`: This clears the existing background, and redraws it using the colour specified by the `background-colour` chooser.
- `import-background`: This allows the user to use an image file as the background.
- `draw-patches-path`: This draws a path using patch agents. The size of the patches is specified by the `patches-path-size` slider.
- `draw-turtles-path`: This will draw a path using turtle agents. The size of the turtles is specified by the `turtles-path-size` slider.
- `draw-text`: This draws the text specified in the text `Input` box on the map. The right side of the text ends up at the point the mouse is clicked. The colour of the text is specified by the `text-colour` chooser. The font size for all text can be changed globally by pressing the `Settings` button in the `Interface`.

The model's `Interface` choosers, sliders and `Input` box are defined as follows:

- `path-colour`: This sets the colour that is used to draw the path when the `draw-path` button is pressed.
- `path-size`: This sets the pen size that is used to draw the path when the `draw-path` button is pressed.
- `object-type`: This sets the type of object that is drawn when the `draw-object` button is pressed.

- `object-colour`: This sets the colour of the object that is drawn when the `draw-object` button is pressed.
- `object-size`: This sets the size of the object that is drawn when the `draw-object` button is pressed.
- `background-colour`: This sets the colour of the background that is drawn when the `clear-background` button is pressed.
- `patches-path-size`: This sets the size of the path that is drawn when the `draw-patches-path` button is pressed.
- `turtles-path-size`: This sets the size of the path that is drawn when the `draw-turtles-path` button is pressed.
- `text`: This sets the text that is drawn when the `draw-text` button is pressed.
- `text-colour`: This sets the colour of the text that is drawn when the `draw-text` button is pressed.

THINGS TO TRY

Try drawing your own map – perhaps of your house, apartment or flat, or of the local school. Note while you are making the map the problems that arise with representing the real world. Once finished, check how many errors there are, and how much detail has been missed off the map. A month or so latter, check again, and see how much has changed.

Try loading the Central Park map. This is done by using the NetLogo menu option for importing a world. Select the `File` menu from the NetLogo menu bar, then select `Import World` option.

EXTENDING THE MODEL

This model has a long way to go before it could be used to draw more high quality maps, such as that used for orienteering. One of the main drawbacks is the poor quality of the path drawing. Can this be improved in some manner?

NETLOGO FEATURES

The `Import World` and `Export World` menu options are used to save the map to disk and re-load it.

Exercise 9.7.2:

Adapt the code from the Map Drawing model so that you can annotate any imported map or image with the following:

- circles;
- arrows;
- text;
- marks (an ‘x’ for marking a spot).

9.8 Representing knowledge using event maps

Exercise 9.8.1: Central Park NetLogo Model

Try out the Central Park Events model in NetLogo:

Central Park Events	http://files.bookboon.com/ai/Central-Park-Events.html

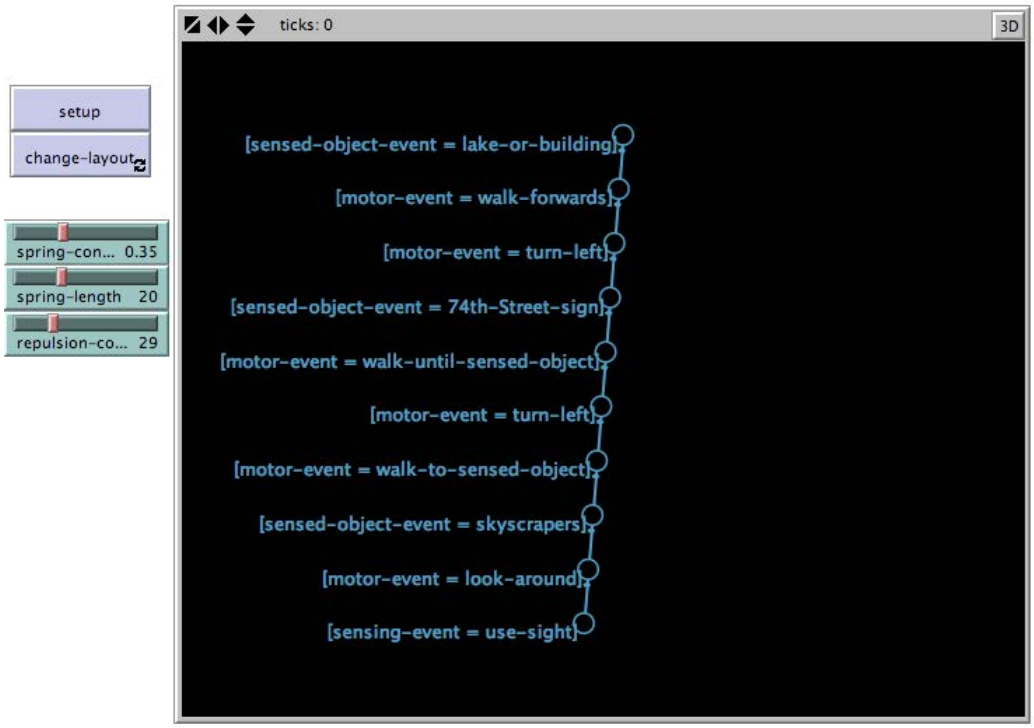


Figure 9.8.1. Screenshot of the Interface for the Central Park Events model after the setup button has been pressed followed by the change-layout button combined with moving the spring-length slider down to 0 and back up to 20.

WHAT IS IT?

This model visualises a small set of events that an agent needs to perform if they wish to go from the Zoo to the Boat Pond in New York's Central Park. The events are shown using an event map representation, where events are linked to other events on separate streams in an ordered sequence. The idea is that an agent processes events simultaneously on separate streams. The approach is similar to the approach adopted for Event Stream Processing (ESP).

WHAT IS ITS PURPOSE?

The purpose of this model is to show how to visualise a series of events using an event map.

HOW IT WORKS

The model uses turtle agents to represents the states in the event map, and uses links agents to represent the paths between states. States own three variables:

- `depth`: The depth in the event map tree.
- `stream`: The stream name (where a stream consists of a sequence of sensory or motor events).
- `event`: The event – either sensory or motor. (Abstract events are not needed for this particular event map).

A spring layout is used to visualise the event map.

HOW TO USE IT

Press the `setup` button first. This will usually produce a cluttered layout. To unfold the clutter, press the `change-layout` button, and then dynamically change the values in the sliders that control the layout. One effective technique is to reduce the value of the `spring-length` slider to 0, then slowly increase it back up again until the desired length and layout is achieved.

THE INTERFACE

The model's `Interface` buttons are defined as follows:

- `setup`: This will clear the environment and variables and (re)-load the event map. Normally, this will appear in a cluttered form and the `change-layout` button needs to be pressed subsequently.
- `change-layout`: This can be used to clear some of the clutter by changing the values in the three sliders.

The model's `Interface` sliders are defined as follows:

- `spring-constant`: This is a value used by the `layout-spring` command. Changing it will usually not affect the visualisation of the event map much.
- `sprint-length`: This modifies the length of the paths between the states of the event map network.
- `repulsion-constant`: This controls how much each of the states repel each other.

THINGS TO NOTICE

Notice how the `repulsion-constant` slider can be used to "repel" the states away from each other (for larger values) and "attract" the states towards each other (for smaller values).

Notice that the clutter in the network layout can often be removed by setting the value of the `spring-length` slider to zero and then increasing it afterwards.

THINGS TO TRY

Try altering the values of the sliders to see what effect this has on the layout.

EXTENDING THE MODEL

Combine this model with the Map Drawing model loaded using the Map Drawing Central Park World. Then show several agents wandering between the Zoo and Boat Pond by animating the map and event map at the same time. This can be done so that each agent ends up with slight variations in the paths they choose since there is the possibility for minor variations to occur between the states of the event map. For example, when turning left, the agent may choose to turn left by slightly more or slightly less than 90 degrees. Similarly, when walking forwards, the agent can choose to walk forwards a distance that varies slightly to other agents applying the same event map.

NETLOGO FEATURES

The model uses the `layout-spring` command for modifying the layout of the network of states (turtle agents) and paths (link agents).

RELATED MODELS

See the Wall Following Events model and the Knowledge Representation model.

Exercise 9.8.2: Knowledge Representation NetLogo Model

Try out the Knowledge Representation model in NetLogo:

Knowledge Representation	http://files.bookboon.com/ai/Knowledge-Representation.html

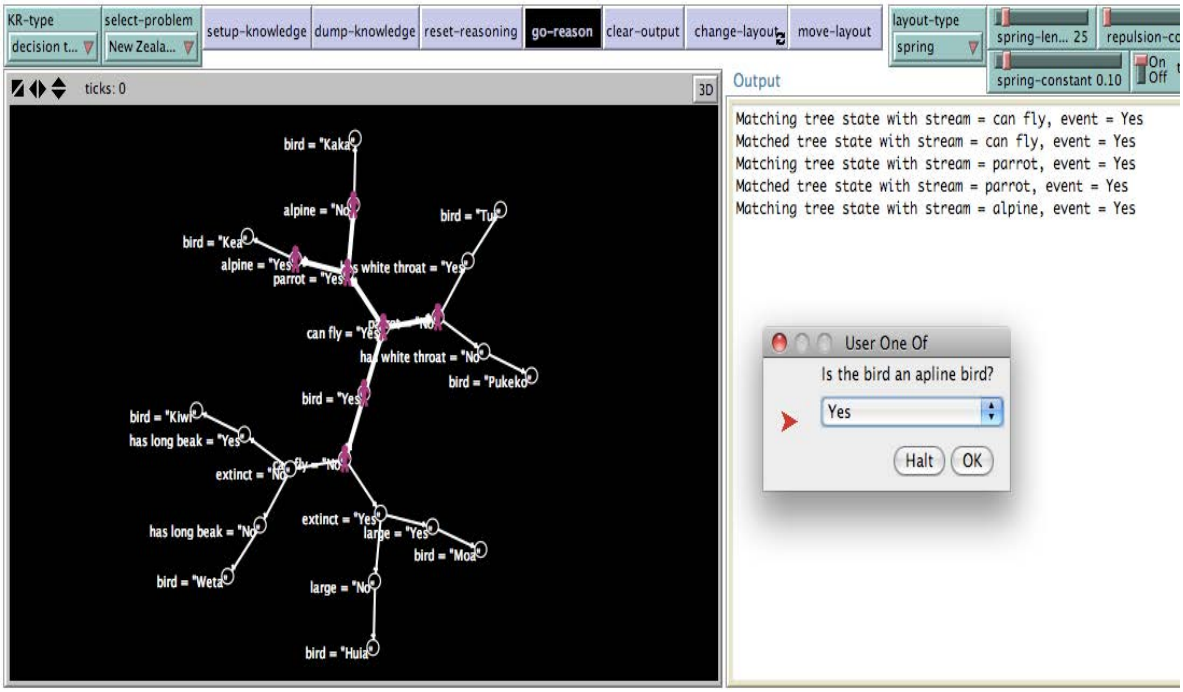


Figure 9.8.2. Screenshot of the Interface for the Knowledge Representation model after the setup-knowledge button has been pressed followed by the go-reason button. The type of knowledge representation has been set to decision tree using the KR-type chooser, and the problem has been set to the New Zealand birds problem. The Output box shows a trace of the initial stages of the reasoning process.

WHAT IS IT?

This model shows how the standard types of knowledge representation such as logic, frames, rules and decision trees can be implemented in NetLogo. The model also shows how to perform reasoning using the different types of knowledge representation. A new type of knowledge representation, called event maps, is also implemented. Event maps are related to the concept of Event Stream Processing (ESP) where events occur concurrently on separate streams.

The knowledge for three toy problems is implemented in the model. These three problems are:

- *Zoo animals*: Identify the name of a zoo animal from observations.
- *New Zealand birds*: Guess the name of a New Zealand bird.
- *Sailing boats*: Identify the type of boat that is sailing past in a tall ships parade.

WHAT IS ITS PURPOSE?

The purpose of this model is to show how to represent knowledge using the traditional forms of knowledge representation such as first order logic, frames, rules and decision trees. A secondary purpose is to show how these different forms of knowledge representation can be re-cast and visualised within a new type knowledge of representation called event maps. The event maps representation also facilitates the visualisation of the reasoning process as well.

HOW IT WORKS

The different forms of knowledge are represented using `state` turtle agents and `path` link agents that are used to describe a transition path between the states. State agents own three variables:

- `depth`: the depth in the event tree (where the root of the tree is at depth 0, a child of the root state is at depth 1, and so on);
- `stream`: This is the name of the stream with which the event is associated;
- `event`: This is the event itself. Events can be sensory events, motor events, or abstract events (the latter is sensory, motor and abstract events that when they occur together cause this event to occur simultaneously as well).

The states and paths form a finite state network. `Walker` turtle agents are used to "walk" the event map network during the reasoning process. The paths (i.e. transitions) are not labelled; they just indicate the order with which events occur. The network of states and paths can be thought analogously of as a map of events (hence why it is called an event map). The event map may or may not be a faithful representation of what is really happening in the environment i.e. in this model, the event map may or may not be a faithful representation of the knowledge that it is trying to represent.

Event maps can be used to implement all the standard types of knowledge representation as shown by this model and also the associated reasoning processes such as forward and backward reasoning for rules, and frame-based reasoning using frame-matching methods. They also provide a useful way of visualising the different types of knowledge representation, and of animating the reasoning processes.

HOW TO USE IT

First pick a type of knowledge representation using the `KR-type` chooser in the `Interface`. Then pick a problem using the `select-problem` chooser. To load the knowledge related to the problem, press the `setup-knowledge` button. This will display the knowledge in the large black environment box on the left. If the visualisation is a bit cluttered at first and the `layout-type` has been set to `spring`, press the `change-layout` button while playing around with the `spring-length`, `spring-constant` and `repulsion-constant` sliders. Then make sure the reasoning process is reset to the initial state by pressing the `rest-reasoning` button. Finally, to perform the reasoning process, press the `go-reason` button.

THE INTERFACE

The model's `Interface` buttons are defined as follows:

- `setup-knowledge`: This loads the knowledge for the problem as specified by the `select-problem` chooser using the knowledge representation specified by the `KR-type` chooser. The knowledge is visualised in the environment using an event map finite state automata that depicts the relationships between knowledge states.
- `dump-knowledge`: This dumps into the `Output` box the knowledge that was loaded by the `setup-knowledge` button.
- `reset-reasoning`: This resets the reasoning to start over again from the start state. This state is determined by the type of reasoning being performed i.e. for backward reasoning using rules, the start state is the goal state; for forward reasoning, the start state is the state associated with the first rule in the rules database.
- `go-reasoning`: This starts the reasoning process. If rules-based reasoning is being used, the user will be asked whether they wish to perform forward or backward reasoning.
- `clear-output`: This clears the `Output` box.

- `change-layout`: This changes the layout if the `layout-type` chooser is set to `spring`. This can be useful to remove some of the clutter. In order to change the layout configuration, the user can alter the `spring-length`, `spring-constant` and `repulsion-constant` sliders while the `change-layout` button is pressed.
- `move-layout`: Sometimes the labels of states in the layout appear off the edge of the environment. This button allows the layout to be moved left, right, up or down by a user-specified distance.

The model's `Interface` sliders, switches, choosers and output box are defined as follows:

- `KR-type`: This selects the type of knowledge representation e.g. rules, frames, decision trees and semantic networks.
- `select-problem`: This selects the type of problem:
  `"Zoo Animals"`: Identify the name of a zoo animal from observations.
  `"New Zealand birds"`: Guess the name of a New Zealand bird.
  `"Sailing boats"`: Identify the type of boat that is sailing past in a tall ships parade.
- `layout-type`: This selects the type of layout to be used for visualising the states and paths in the event map.
- `spring-length`, `spring-constant`, `repulsion-constant`: This can be used to modify the layout if the `layout-type` is `spring`.
- `trace-on?`: If set to `On`, this will write output into the `Output` box that traces the reasoning process.
- `Output box`: This is used for writing the output of the reasoning process and other information into.

THINGS TO NOTICE

Notice which of the types of knowledge representation make the reasoning process more difficult to understand when they are visualised using the event map format in the environment (for example, the `walker` agents for semantic networks seem to jump around a lot).

Notice which of the types of knowledge representation produce more complicated event maps for the different problems.

THINGS TO TRY

Try the different types of knowledge representation for the three types of problems. Try to see how they relate to each other and how they differ. Find out what happens during the reasoning process. Which are easy to follow, and which are difficult?

Try finding out how the knowledge for the different types of knowledge representation is initialised. (Hint: look at the `setup-knowledge` command). Find out how rules are defined, and then how they are visualised as event maps. Note that in the model, rules and semantic networks are defined explicitly, whereas frames and decision trees are setup by converting from rules. (How?) Have a go at explicitly defining the knowledge for frames and decision trees instead rather than use a conversion process.

EXTENDING THE MODEL

Try adding other types of knowledge representation, perhaps some that are hybrids of the ones implemented in the model. For example, the semantic event network type that comes with the model is a hybrid of the semantic networks and event map types.

Investigate alternative methods for visualising the reasoning process for each type of knowledge representations.

NETLOGO FEATURES

Note the use of the `hatch` command to create `walker` agents to continue the walking of the event map network during the reasoning process. How the `walker` agents walk the network determines the type of reasoning.

RELATED MODELS

For other examples of the use of event maps for event stream processing, see the Language Modelling, Central Park Events and Wall Following Events NetLogo models.

Exercise 9.8.3:

Have a play with the Knowledge Representation NetLogo model to try out the different methods of representation. To select the problem, use the `select-problem` chooser in the Interface. Select the different types of knowledge representation by selecting the appropriate option using the `KR-type` chooser. Try different settings to see what happens – for example, you can use the `change-layout` button combined with the sliders on the right to make the displayed network more visually appealing. Also try out the `go-reason` button to see what happens. Explain what you observe.

Exercise 9.8.4:

Try adding the knowledge for your own problem into the Knowledge Representation model described in Exercise 9.8.2. Observe while you are doing this the problems you have with formulating the knowledge for each of the different types of knowledge representation. What are the advantages or disadvantages of using the different types? How could they possibly be improved given the insights you have gained by the knowledge representation process? Is there yet another new type of knowledge representation that would be better suited for the task?

Exercise 9.8.4:

The three example problems tackled by the Knowledge Representation NetLogo model are 'toy' problems rather than real-world problems. Enumerating the full knowledge involved and scaling up to a non-trivial real-world problem presents many difficulties and is still an active area of research in Artificial Intelligence. However, the merit of these examples is their simplicity and their usefulness in illustrating how the models differ.

What would need to be done to scale up the knowledge for these problems to their real-world equivalents – for example, identifying the name of an animal while walking around a zoo, or the name of a New Zealand bird while out tramping the Milford Track, or identifying the different types of yachts seen in a harbour?

Exercise 9.8.5:

Explain the difference between forward reasoning and backward reasoning when rules are chosen as the `KR-type` for the Knowledge Representation model. (Hint: One way of answering this question is by looking to see how they are implemented in the NetLogo code for the model).

9.9 Representing knowledge using rules and logic

Exercise 9.9.1:

Consider the following sentences:

1. *Peter is an old man.*
2. *Mary is the sister of Peter.*
3. *Some cats have large ears.*
4. *If X is the sister of Y, and Y is a boy, then Y is the brother of X.*
5. *Simba is a white elephant.*
6. *All elephants are mammals.*
7. *Elephants have a trunk and large ears.*
8. *Ruby is a black and white cat.*
9. *Ruby is the mother of Pearl and Pearl is the daughter of Ruby.*
10. *Some cats are small.*
11. *All cats eat meat.*
12. *Cats have whiskers and a tail.*

Represent the knowledge contained in each of the sentences above using the two knowledge representation paradigms – production rules and first order logic. Which sentences do you have problems with when trying to represent the knowledge and why?

Exercise 9.9.2:

Convert the following predicate logic sentences into English sentences:

(i) $\exists x(Car(x) \wedge Red(x))$

(ii) $\forall x(Cake(x) \Rightarrow Sweet(x))$

(iii) $\exists x \forall y(Person(x) \wedge Cake(y) \wedge \neg Likes(x,y))$

Exercise 9.9.3:

Convert the following sentences into predicate logic sentences:

(i) All cats like fish.

(ii) Bananas are either green or yellow or brown.

(iii) Some people eat everything.

9.10 Reasoning using rules and logic

Exercise 9.10.1:

Consider the following rule based expert system:

```
Database:
      A B C D
Rules base:
      Rule 1: IF X is true AND Y is true AND A is true
              THEN Z is true
      Rule 2: IF V is true AND B is true
              THEN Y is true
      Rule 3: IF V is true AND C is true
              THEN X is true
      Rule 4: IF D is true
              THEN V is true
```

i. Use forward chaining to determine if the fact Z can be inferred from this knowledge base. Show the state of the database and the rule that was fired after each step in the reasoning process.

ii. Use backward chaining on the same knowledge base to determine if the goal Z succeeds. Show the state of the database and the rule that was matched after each step in the reasoning process.

9.11 Knowledge and reasoning using frames

Exercise 9.11.1:

Represent the knowledge contained in each of the sentences for Exercise 9.9.1 using frames, and compare with the solutions you devised for production rules and first order logic. Which sentences do you have problems with when trying to represent the knowledge and why?

Exercise 9.11.2:

Modify the Knowledge Representation model so that the frames knowledge is explicitly defined for the three problems implemented in the model instead of being converted from rules (as is done in the `convert-rules-to-frames-base` procedure). You will need to modify the `setup-frames-base` procedure so that it is similar to the `setup-semantic-network` procedure and also define a `setup-frame` procedure for defining a specific frame.

9.12 Knowledge and reasoning using decision trees

Exercise 9.12.1:

Modify the Knowledge Representation model so that the decision-tree knowledge is explicitly defined for the three problems implemented in the model instead of being converted from rules (as is done in the `convert-rules-to-decision-tree` procedure). You will need to define a `setup-decision-forest` procedure similar to the `setup-semantic-network` procedure and also define a `setup-decision-tree` procedure for defining a specific decision tree.

9.13 Knowledge and reasoning using semantic networks

Exercise 9.13.1:

Redo Exercise 9.8.3 by running the Knowledge Representation model again for the three different problems (as specified by the `select-problem` chooser) and the different types of knowledge representation (as specified by the `KR-type` chooser). This time play close attention to how the knowledge is visualised in the environment for the different types, and observe what happens during the reasoning process when the `go-reason` button is pressed. Which of the types seem easier to follow what is happening? In particular, note what happens with the semantic networks type of reasoning. Is there a way of improving the visualization of the reasoning process of semantic networks, or is this an inherent problem with this type of knowledge representation?

10 Intelligence

10.1 The nature of intelligence

Exercise 10.1.1:

Conversational ability and problem solving ability are postulated as two important ingredients for human-level intelligence in this chapter. Suggest some further ingredients and justify why you think they are necessary for your recipe for intelligence.

10.2 Intelligence without representation and reason

Exercise 10.2.1:

What do we mean by 'representation' and 'reason' when we talk about 'intelligence without representation and reason'? Is it fair to say that Brooks' insect-like robots and the ant and termite agents in the Ants and Termites models do not 'represent' knowledge and 'reason' in some manner? (After all, the 'intelligence' is represented globally in the entire multi-agent system, but not individually within each agent.)

(See also Exercise 9.6.1.)

10.3 What AI can and can't do

Exercise 10.3.1:

Write a list of 10 tasks that humans can do that require some degree of intelligence. Find out whether there are computer systems that can do each of the tasks on your list. How well do they perform at each of the tasks? Are they capable of successfully completing the task? What human level of performance are they capable of – novice, intermediate or expert?

10.4 The Need for Design Objectives for Artificial Intelligence

Exercise 10.4.1:

Research in the A.I. literature how the term 'intelligence' is being defined. Are there distinctly different meanings of the term or do they all spring from the same generic meaning? Do the publications provide any justification for why the systems described can be called 'intelligent'?

10.5 What are Good Objectives?

Exercise 10.5.1:

Devise your own design objectives for Artificial Intelligence. Become your own worst critic – analyse your objectives to see if they are SMARTER. Which attributes did you fail on? – i.e. are they specific enough, are they measurable, are they achievable, are they realistic, are they time-bound and so on?

10.6 Some Design Objectives for Artificial Intelligence

Exercise 10.6.1: Do you think the design objectives listed in Section 10.6 of the companion book *Artificial Intelligence – Agent Behaviour I* are 'good' objectives? Explain your reasoning.

10.7 Towards believable agents

Exercise 10.7.1: Chatbot Model

Try out the Chatbot model in NetLogo:

Chatbot	http://files.bookboon.com/ai/Chatbot.html

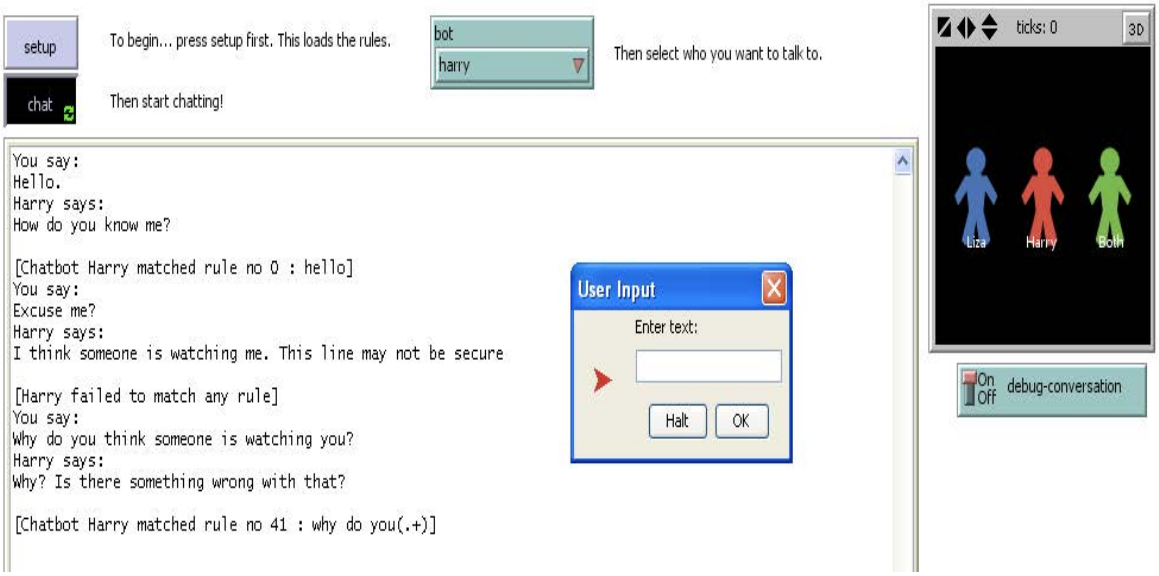


Figure 10.7.1. Screenshot of the Interface for the Chatbot model after the setup button has been pressed followed by the chat button. The chatbot selected by the bot chooser is Harry.

WHAT IS IT?

This model implements two basic chatbots – Liza and Harry. These are named after their more illustrious counterparts, Eliza and Parry. Unlike Eliza, Liza does not try to be a Rogerian psychotherapist, but Harry does try to be a bit paranoid like Parry.

HOW IT WORKS

The model makes use of an extension to NetLogo called "`re`" for regular expressions. The extension allows developers in NetLogo to specify and match regular expressions. In the NetLogo code, a `setup-rule` command is used to specify the regular expression and a list of possible chatbot responses. If the chatbot matches the user's reply against the regular expression in the rule, then it will choose one of the replies from the response list.

The extension `re` adds the following commands to NetLogo:

- `compile-pattern`: This compiles the regular expression pattern passed as the first argument.
- `set-target`: This sets the target string that you want to search for to that specified by the first argument.
- `run-matcher`: This tries to match the regular expression in the target string.
- `get-group-length`: This returns the number of groups.
- `get-group`: This returns the item in the group at the position specified by the first argument.
- `get-starts-length`: This returns the length of `starts`.
- `get-starts`: This returns the item in `starts` at the position specified by the first argument.
- `get-ends-length`: This returns the length of `ends`.
- `get-ends`: This returns the item in `ends` at the position specified by the first argument.
- `clear-all`: This clears all the variables.
- `setup-regex`: This sets up the complete regular expression system and searches for the target specified as the first argument.

HOW TO USE IT

First press the `setup` button in the `Interface` – this will load the rules for each chatbot. Then choose the chatbot you would like to chat with using the bot chooser – either "`liza`", "`harry`" or "`both`". Then start chatting by pressing the `chat` button.

THE INTERFACE

The `Interface` buttons are defined as follows:

- `setup`: This loads the rules for each chatbots.
- `chat`: This starts or continues the conversation with the chatbot that was selected using the bot chooser.

The Interface chooser and switch is defined as follows:

- `bot`: This sets the chatbot to the `Liza` chatbot, the `Harry` chatbot or `Both`.
- `debug-conversation`: If this is set to `On`, debug information is also printed showing which rules matched.

THINGS TO NOTICE

Harry seems to do a bit better at being paranoid than Liza does at being a Rogerian psychotherapist. The latter also tends to repeat things a lot. However, it doesn't take long for the conversation with both chatbots to break down.

THINGS TO TRY

Try out the different chatbots by changing the `bot` chooser. Does Liza seem to have a different personality to `Harry`, or is `Both` any different to the other two? Try changing the chatbot mid-conversation.

Clearly, the conversational abilities of all three chatbots are poor. But can you devise a conversation that seems believable. Now what made it believable? Can you change the chatbots to do this automatically?

EXTENDING THE MODEL

Try adding your own rules to the chatbots. You may need to turn the switch `debug-conversation` to `On` to find out why some of your new rules do not work. Remember, the order the rules are processed is very important – an earlier rule that matches will supercede a latter one, resulting in the latter one never coming into effect. So be very careful about where you place your rule. Can you think of a way to avoid this problem?

NETLOGO FEATURES

The model demonstrates the use of an extension where the code is written in Java.

Exercise 10.7.2:

Describe how you might go about building a conversational agent to pass the Turing Test. What are the major hurdles that need to be overcome? Do you think that an Eliza-like approach based on using regular expressions adequate for tackling this problem?

10.8 Towards computers with problem solving ability

Exercise 10.8.1: Water Flowing Uphill Model

Try out the Water Flowing Uphill model in NetLogo:

Water Flowing Uphill	http://files.bookboon.com/ai/Water-Flowing-Uphill.html

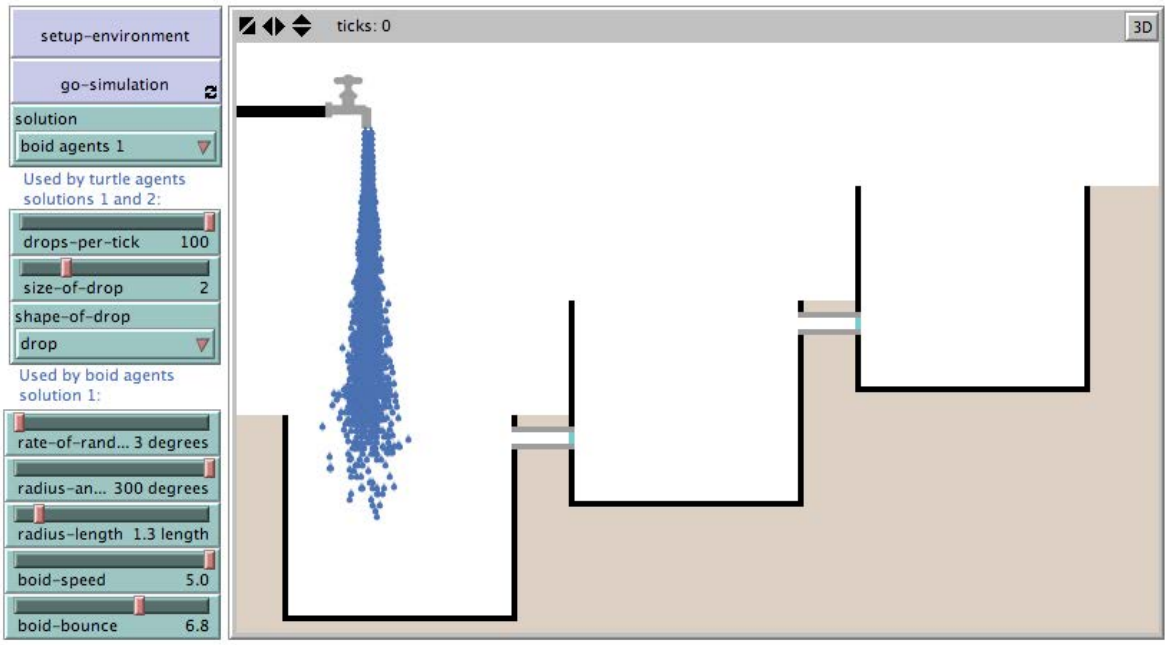


Figure 10.8.1. Screenshot of the Interface for the Water Flowing Uphill model after the setup-environment button has been pressed followed by the go-simulation button. The solution selected by the solution chooser is "boid agents 1".

WHAT IS IT?

This model tries to visually simulate one possible solution to the problem of trying to get water to flow uphill. The solution is to use tanks or reservoirs at increasing heights, and have these fill up one after the other. A stop-valve (shown in blue at the end of the connecting pipes between reservoirs in the model) is used to prevent back filling of the reservoirs – once the water moves through the valve, it cannot flow backwards.

Various methods are tried to simulate how the "water" flows. All of these methods are failures, some of them producing strange effects that are impossible in the real world. The reader is encouraged to add his or her own methods which better approximate how water flows in real life.

WHAT IS ITS PURPOSE?

The purpose of the model is to draw an analogy between a seemingly impossible task (trying to get water to flow uphill) and what some people claim to also be impossible (creating Artificial Intelligence). For both of these tasks, we can try to find solutions that change the problem in some way, or we can try to change the physical properties of the universe (i.e. have it work in a virtual environment, but it would not work when transferred to a real environment), or we could create an illusion (i.e. it would look like the problem was solved, but like magic, it would just be trickery.)

A secondary purpose is to highlight the process of problem solving that humans are very good at. No computer program is capable of creating the variety of solutions that are shown in this model. For that matter, no computer programs are yet capable of understanding the problem itself.

HOW IT WORKS

The model implements various types of agents to try to simulate water flow – turtle agents, patch agents, agents that are boids (from Craig Reynolds work) with a cone of vision, and particle agents that simulate the velocity of each particle and takes into consideration viscosity, wind and gravity.

None of these solutions work i.e. they behave nothing like water does.

HOW TO USE IT

Press the `setup-environment` button in the `Interface` first to reset the simulation. Then choose one of the solutions using the `solution` chooser. See what happens when you press the `go-simulation` button.

THE INTERFACE

The model's `Interface` buttons are defined as follows:

- `setup-simulation`: This will reset the simulation back to the start state which is where all the reservoirs have no water in them, and the tap is turned off.
- `go-simulation`: This turns on the tap, and agents that simulate water droplets start flowing downwards out of the tap into the first reservoir on the left.

The model's `Interface` sliders, switches and chooser are defined as follows:

1. For when the `solution` variable is set to either "`turtle agents 1`" or "`turtle agents 2`":

- `drops-per-tick`: This is the number of drops that will emerge out of the tap every tick.
- `size-of-drop`: This is the size of each drop.
- `shape-of-drop`: This is the shape of each drop. Possible shapes are "`drop`", "`circle`", "`square`" and "`hex`".

2. For when the `solution` variable is set to "`boid agents 1`":

- `rate-of-random-turn`: This controls how much the boid randomly turns each tick.
- `radius-angle` and `radius-length`: This defines the boids' cone of vision.
- `boid-speed`: This specifies how far the boids should move each tick. `boid-bounce`: This specifies how far the boids should bounce back when they hit the tank walls or the earth.

3. For when the `solution` variable is set to "`particle agents 1`":

- `max-particles`: This specifies the maximum number of particles and can be used to control how many particles come out of the tap.
- `initial-velocity-x` and `initial-velocity-y`: This specifies the initial velocity of the particle in the *x* and *y* directions.
- `initial-range-x`: This provides some random variability in the velocity in the *x* direction.
- `restitution-constant`: This can be used to control how much bounce with floor damping occurs if the particle touches the tank floor.
- `gravity-constant`: This is used to scale the force of the gravity according to the particle's mass thereby simulating the effect of air resistance.
- `viscosity-constant`: This can be used to simulate the effect of viscosity.
- `rate`: This controls the rate of particle emission.
- `step-size`: The new location and speed of each particle is calculated by multiplying the forces by this number. The ticks also advance by this number.
- `wind-velocity-x` and `wind-velocity-y`: This specifies the velocity of the wind in the *x* and *y* directions.

THINGS TO NOTICE

When the chooser `solution` in the `Interface` is set to "`turtle agents 1`", the agents behave a bit like water, but do not spread out at the bottom. Why? How can their behaviour be changed to fix the problem?

When the chooser `solution` in the `Interface` is set to "`turtle agents 1`", the agents behave a bit like snow. Why? How can their behaviour be changed so that they behave more like water?

When the chooser `solution` in the `Interface` is set to "`patch agents 1`", the agents do not flow correctly on the right. Why? How can their behaviour be changed so that the right side of the reservoir fills up correctly and then flows via the connecting pipe to the next reservoir and so on?

When the chooser `solution` in the `Interface` is set to "`patch agents 2`", "`patch agents 3`", or "`patch agents 4`", the simulations are complete failures. Can any of these be salvaged in order to find a possible solution?

When the chooser `solution` in the `Interface` is set to "`boid agents 1`", the agents behave a bit like spray-painting in some configurations. Why? Also, like the "`turtle agents 1`" solution, they do not spread out at the bottom. Why? How can their behaviour be changed to fix the problem? How can their behaviour be changed so that they behave more like water?

When the chooser `solution` in the `Interface` is set to "`particle agents 1`", the agents tunnel through the surrounding earth. Why? How can their behaviour be changed so that they behave more like water?

THINGS TO TRY

Try out each of the solutions in turn to see what happens. Which one seems to be closest to what happens in real life?

Try changing the values of the sliders to see what effect they have on each solution (for `turtle agents solutions 1` and 2, `patch agents solution 1`, `boids agents solution 1` and `particle agents solution 1`).

When the solution variable is set to "`particle agents 1`", try sliding the "`max-particles`" slider back and forth. The effect of this is that the tap will seem to be temporarily turned on and off.

EXTENDING THE MODEL

Try devising your own solution that simulates water flow more accurately. Observe the process of problem solving that you apply in order to come up with your solution. Would it be possible to devise a computer program to replicate your behaviour?

Exercise 10.8.2:

Run the Water Flowing Uphill NetLogo model and set the `solution` chooser to "`particle agents 1`". Run the simulation by then pressing the `setup-environment` button followed by the `go-simulation` button. Observe how the particle agents start 'flowing' uphill by somehow managing to tunnel through the surrounding soil, nothing like what happens in real life. Look at the code that gets executed when this happens. See if you can determine what causes the agents to tunnel, and try to figure out a way of fixing this in order to make the agents behave the way they might do in real life.

11 Solutions to Selected Exercises

Solution to Exercise 6.1.2:

The Firebreak model illustrates how the Fire model can be extended to allow the user to add firebreaks, extra forest and ignition points as well as rudimentary image processing to simulate a real environment.

Firebreak	http://files.bookboon.com/ai/Firebreak.html

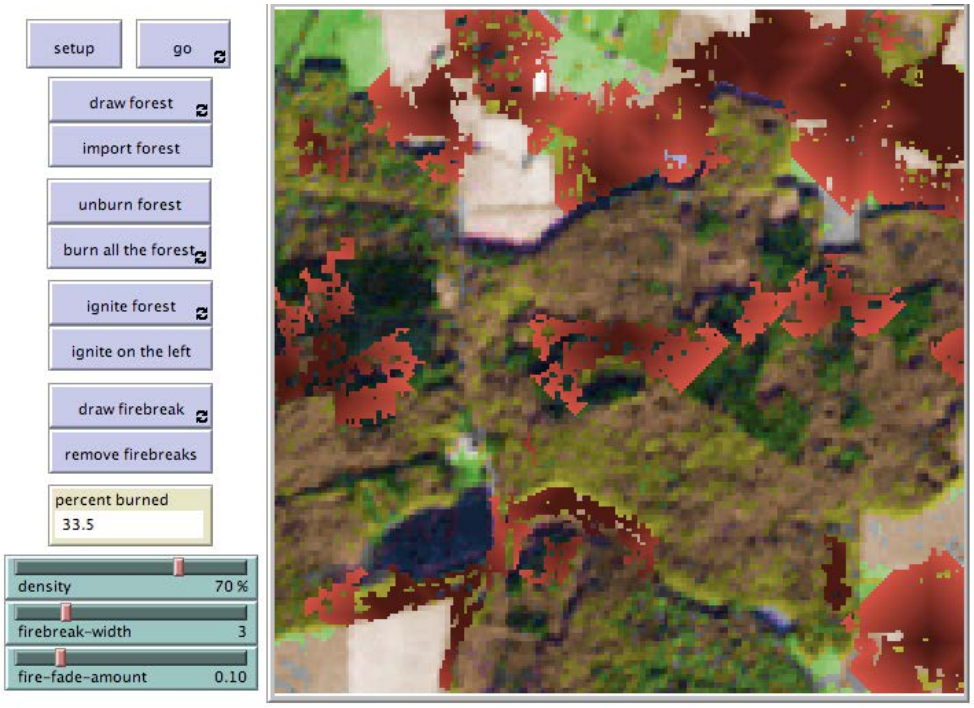


Figure S-6.1.2. Screenshot of the Interface for the Firebreak model after the setup button has been pressed followed by the import-forest and burn all the forest buttons, with the slider values as shown in the image.

WHAT IS IT?

This model is an extension of the Fire model developed by Uri Wilensky that comes with NetLogo's models library. It simulates the spread of a fire through a forest. This model allows the user to add firebreaks, extra forest and ignition points. It also allows the user to import a small area taken from a simulated natural colour satellite image of Châteaubriant, France in order to test out what might happen in a real-life environment.

HOW IT WORKS

The `setup` button will set up a random forest with the specified density. If the user presses the `import forest` button, then the satellite image of Châteaubriant, France is imported instead.

The user can start fires at various points by pressing the `ignite forest` button and then clicking the mouse button at the required location. Alternatively, the user can choose to ignite the fire on a continuous vertical line at the left of the environment using the `ignite on the left` button. Extra forest can be drawn using the `draw forest` button, and firebreaks can be drawn using the `draw firebreak` button.

Pressing the `go` button will then start the simulation, and the fire will then spread to neighbouring trees in four directions (N, S, E and W) from the initial ignition points. The spread of the fire depends on the density of the surrounding forest. There is no wind being simulated in the model, so the fire can only spread to adjacent trees, and has no ability to jump over an unwooded area (patch).

The fire is implemented using `fire` turtle agents. The fire spreads in a breadth-first manner similar to the breadth-first search implemented in the Searching Mazes, Missionaries and Cannibals and Searching for Kevin Bacon models. Once a `fire` agent has ignited its nearest neighbours (if there are any), then it will change its breed to an `ember` agent, whose colour slowly fades from red to black according to the value of the `fire-fade-amount` slider, and then it will die.

HOW TO USE IT

First press the `setup` button. This will create a forest of randomly spaced trees with density as specified by the `density` slider. Alternatively, to import the satellite image of Châteaubriant, France, press the `import forest` button.

To start a fire at a specific point, press the `ignite forest` button and click the mouse button at the required location. To have the fire start at the left of the environment as with the Fire model, then press the `ignite on the left` button.

To create more forest, press the `draw forest` button. To create a firebreak, press the `draw firebreak` button. Firebreaks can be removed using the `remove firebreaks` button. Burnt forest can be 'unburnt' using the `unburn forest` button. This means that a simulation for the same forest can be run multiple times using different firebreaks and ignition points.

THE INTERFACE

The `Interface` buttons are defined as follows:

- `setup`: This will reset the simulation and create a random forest.
- `go`: This will start the simulation. If there have been any fire ignition points placed in the environment, then the fire will start spreading to adjacent trees.
- `draw forest`: This will draw more trees where the mouse is subsequently clicked.
- `import forest`: This will import an environment based on a small area in a satellite image taken of Châteaubriant, France.
- `unburn forest`: This will restore the trees to the way the forest was before any trees were burnt. Note that this will not work when the satellite image has been imported. To restore that forest, simply press the `import forest` button again.
- `burn all the forest`: This will eventually burn all the forest. It repeatedly chooses unburnt trees at random each tick to spontaneously combust, while at the same time spreading the fire from the previously selected trees. When the satellite image is imported, this will end up showing which parts of the image the `is-tree?` reporter considers to be trees.
- `ignite forest`: This will set an ignition point where the mouse is clicked from which the fire will start burning. Keeping the mouse down will draw multiple overlapping ignition points. The fire will then spread out once the `go` button is pressed.
- `ignite on the left`: The vertical line of fire will be ignited from the left side of the environment. The fire will then spread across to the right once the `go` button is pressed.

- `draw firebreak`: This will draw a firebreak coloured dark brown into the environment at the point where the mouse is clicked. To continue drawing the firebreak, keep on clicking the mouse, while at the same time dragging it in the desired direction.
- `remove firebreaks`: This will remove all firebreaks that have been drawn in the environment.

The `Interface` monitor and sliders are defined as follows:

- `percent burned`: This is the percentage of trees that have been burnt.
- `density`: This is the density of the randomly generated forest when the `setup` button is pressed.
- `firebreak-width`: This sets the width of the firebreaks.
- `fire-fade-amount`: This controls how long a tree remains burning once it has been ignited and has already spread the fire to its neighboring trees.

THINGS TO NOTICE

Notice how much of the forest burns for different density settings. What threshold does the density setting need to be reached so that almost all of the forest gets burned?

Notice what happens when the unburnt forest is used to reset the forest back to the way it was when the `setup` button was pressed, and then the `go` button is pressed again. Are the same parts of the forest burnt? If so, then why?

Notice that the turtles do not move. (How can you check whether this is true?) But yet the fire seems to move. How?

Notice which arrangements of firebreaks seem to be best at containing the fire.

Notice which parts of the imported satellite image are confused with trees. How could the `is-tree?` reporter be improved to make a better categorisation of the image?

THINGS TO TRY

Try drawing firebreaks as the fire is spreading. Often it is very difficult to contain the fire and prevent it from spreading further. Slowing down the simulation using the `Interface` speed slider can help, but this will also slow down the drawing of the firebreaks.

Try drawing a firebreak from one side of the environment to the other. Then try leaving in some gaps. Also try drawing firebreak circles to see what happens (with ignition points placed inside and outside of the circles).

Load the satellite image for Châteaubriant, France and start the simulation using different ignition points. How well does the simple code for recognizing a tree (i.e. the `is-tree?` reporter) do at recognizing where the trees in the image are?

Try different ignition points with different configurations of firebreaks. What firebreak drawing behaviour works best at containing the different fires?

Try changing the values of the three sliders to see how this affects the simulation.

Try pressing the `unburn forest` button while the fire is still spreading.

Try using the BehaviorSpace feature (see `Tools` menu) to see the affect on the percentage of trees burnt for different slider values. (See Exercise 7.6.1 for more information on how to use the BehaviorSpace tool).

Try changing the way the fire spreads using the `neighbours` reporter instead of the `neighbors4` reporter in the code. How does this affect the ability of the fire to spread?

EXTENDING THE MODEL

Add wind to the model so that the fire has a chance to 'jump' over firebreaks.

Add two types of people agents to the model – firstly, `fire-fighters` who autonomously fight the fire by creating firebreaks and using water to put out the fire; and secondly, `fire-lighters` who run around lighting the fire.

Add a further `helicopter` breed of agent that will fly over the forest with a large water bucket trying to put out the fire.

Add the ability to import any image file, not just the one used by the model.

RELATED MODELS

See the implementation of Breadth First Search in the Searching Mazes, Missionaries and Cannibals and Searching for Kevin Bacon models.

CREDITS AND REFERENCES

The particle agents code used in this model was obtained from the Particle System Waterfall produced by Uri Wilensky. See the model for a list of references and the `How to Cite` section below.

HOW TO CITE

If you mention this model in an academic publication, we ask that you include citations for the original Fire model and for the NetLogo software:

- Wilensky, U. (1997). NetLogo Fire model. http://ccl.northwestern.edu/netlogo/models/Fire. Center for Connected Learning and Computer-Based Modeling, Northwestern University, Evanston, IL.
- Wilensky, U. (1999). NetLogo. http://ccl.northwestern.edu/netlogo/. Center for Connected Learning and Computer-Based Modeling, Northwestern University, Evanston, IL.

In other publications, please use:

See http://ccl.northwestern.edu/netlogo/models/Fire for terms of use.

The extension to the model was written by Bill Teahan. To refer to this model in publications, please use:

Firebreak NetLogo model.
Teahan, W.J. (2010). Artificial Intelligence. Ventus Publishing Aps.

Solution to Exercise 6.2.1:

The reactive agent approach to this problem is for the agent to merely react to the situation it finds itself in. A reactive agent does not require a representation of the universe in which it is operating. It therefore cannot carry out the type of *a priori* reasoning required for the plan adopted by the cognitive agent as detailed in Exercise 6.2.1.

One possible way to construct a reactive agent that can resolve this problem is to define a set of rules that the agent executes:

R1: *If I am in front of the door and I have a key, then open it.*
R2: *If I am in front of the door and have no key, then try to open it.*
R3: *If the door does not open and I have no key, then go to look for the key.*
R4: *If I am looking for a key and there is a key in front of me, then take the key and go towards the door.*

Solution to Exercise 6.3.1:

All the four models exhibit reactive behaviour because they all react directly to sensory input. None of the models exhibit cognitive behaviour as none of them demonstrate higher level cognitive abilities such as the ability to acquire and process knowledge, the ability to reason, the ability to learn, the ability to plan, the ability to make rational decisions, and so on. All of the methods exhibit some form of self-organisation and therefore exhibit an emergent behaviour as a result. The turtles in the Flocking model organize themselves into flocks, the turtles in the Fireflies model synchronise their flashing, and the turtles in the Termites and State Machine Example models end up with woodchips gathered into piles. This is done through local interactions, rather than being imposed externally.

Solution to Exercise 6.3.2:

The ANZ-Continental-Drift model illustrates one solution of how to get Australia and New Zealand back together again.

ANZ Continental Drift	http://files.bookboon.com/ai/ANZ-Continental-Drift.html

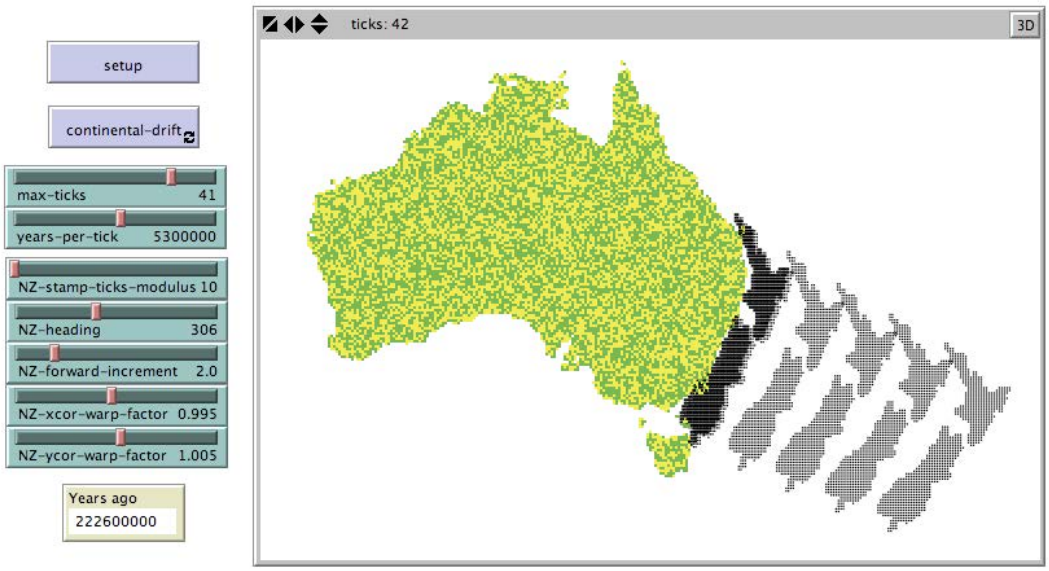


Figure S-6.3.2. Screenshot of the Interface for the ANZ-Continental-Drift model after the setup button has been pressed followed by the continental-drift button, with the slider values as shown in the image.

WHAT IS IT?

This model shifts New Zealand back towards Australia in order to illustrate the process of continental drift. In effect, the model is running time backwards in order to show where New Zealand was in relation to Australia millions of years ago.

WHAT IS ITS PURPOSE?

The main purpose of the model is *not* to accurately simulate what has occurred during the continental drift process between Australia and New Zealand over hundreds of millions of years (since that would require much more code well-grounded in scientific evidence than what is used in this model). Instead, the primary aims of the model are to provide a simple animation of the concept of continental drift, to allow the user to try out different possibilities to see where Australia and New Zealand might have one time been joined, and importantly to show how the process of continental drift has occurred over millions of years.

One salient insight that can be gained from the knowledge of how this process has occurred is that species will have to adapt to epoch-level changes if they are to survive for a very long time.

HOW IT WORKS

The model uses the `import-pcolors` command to load a map of Australia and New Zealand and convert the image into patch agents. In order to simplify the shifting of New Zealand back to Australia, these patch agents are replaced with turtle agents, yellow and green for Australia, and black for New Zealand, to signify the traditional colours of the respective countries. Then a simple translation on a specific heading with some warping is used to move all the black agents back closer to the yellow and green ones.

Clearly, these simulated transformations are nothing like the type of transformations that have occurred during the real process of continental drift between Australia and New Zealand. For example, in the model Australia remains static and does not drift in the simulation – only New Zealand does. However, as stated above, much more code would be required to get a decent simulation that more accurately represents what occurred during the real-life process. (And this was not the primary purpose of the model).

HOW TO USE IT

To reset the simulation and load the present day shapes and relative positions of Australia and New Zealand, press the `setup` button. To start the process of "winding the clock back" so that New Zealand gradually creeps it way back to Australia, press the `continental-drift` button.

If you want New Zealand to end up at a different location, then change the values of the sliders to suit.

THE INTERFACE

The buttons in the `Interface` are defined as follows:

- `setup`: This button clears the environment and resets the variables, then loads the map of present-day Australia and New Zealand.
- `continental-drift`: This button simulates the process of continental drift by drawing the relative positions of the two countries as the clock is wound backwards.

The sliders and monitor in the `Interface` are defined as follows:

- `max-ticks`: This is the maximum number of ticks that the model should be run for. If this number is too high, then New Zealand will eventually run through the middle of Australia (unless the values of the other sliders means that it ends up heading elsewhere). If this number is too low, then the animation will stop early.
- `years-per-tick`: This sets the number of years per tick of the simulation. This number multiplied by the number of ticks is the number that appears in the `Years ago` monitor.
- `NZ-stamp-ticks-modulus`: This determines how many times the shape of New Zealand is "stamped" during the simulation. When the remainder of the number of ticks divided by this number is 0, then all New Zealand agents (i.e. the agents coloured black) will be stamped, thereby leaving an impression in the environment of their current position at the time of the stamping. This can be used to give an impression of where New Zealand has been at earlier stages of the animation.
- `NZ-heading`: This is the heading of New Zealand as it moves towards Australia. Changing this to another value will result in New Zealand heading elsewhere.
- `NZ-forward-increment`: This controls how far New Zealand moves forward each tick on the heading specified by the `NZ-heading` slider.
- `NZ-xcor-warp-factor`, `NZ-ycor-warp-factor`: A straight translation of New Zealand towards Australia on a direct heading does not work – the two countries end up hitting each other where their shapes do not fit well together. These sliders can be used to warp the shape of New Zealand along the *x* and *y* axes respectively so that the shapes more closely fit together when the two countries eventually reach each other.
- `Years ago`: This monitor reports the model's estimate of the number of years ago when the two countries were in the relative positions depicted by the simulation. This number is equal to the following: `ticks` × `years-per-tick`.

THINGS TO NOTICE

Notice that it is not clear where New Zealand should end up joining with Australia. The presence of Tasmania adds an interesting factor in the equation. If you run the simulation several times with different slider values, you will find out that no simple transformation is possible that will lead to a satisfactory joining up of the two countries. Either New Zealand has to be stretched (as in the model), or an added rotation is required to rotate the country slightly around a central rotation point (such as the present day location of Picton) while New Zealand is being translated in a north-west direction towards Australia.

THINGS TO TRY

Play around with different values in the sliders. Try to get New Zealand to miss Australia all together, or alternatively, have New Zealand end up wafting through the middle of it.

Try changing the values of the sliders during the middle of the simulation to see if you can change New Zealand's direction dynamically. Especially try dynamically changing the value of the `NZ-xcor-warp-factor` and `NZ-ycor-warp-factor` sliders. Warping New Zealand will change its direction slightly. (Why?) This can also be changed back again at a latter stage of the simulation.

EXTENDING THE MODEL

Change the model so that Australia moves as well. This will make the animation look better, but will also slow it down. Why?

Have a go at changing the simulation in the model so that that it more accurately reflects projections that are based on scientific evidence. Once you have perfected it for the relatively straightforward situation of the drifting apart of Australia and New Zealand, then adapt your model so that it works for the entire world. Run your model backwards until you reach the time when there was just a single "super" continent – Pangaea.

NETLOGO FEATURES

The `import-pcolors` command is used to load the map of Australia and New Zealand. The `stamp` command is used to stamp the shape of New Zealand at various stages of the simulation. Note also the use of the `sprout` command to create a turtle agent for each patch agent, where the colour is changed to black for New Zealand and yellow and green for Australia. The change of colour makes the coding much easier for identifying which agents to shift around. Either changing part of the environment (for example, by adding a different coloured border around the outside) or changing a specific set of agents by altering a property (such as colour) is a common trick that can be used in NetLogo to greatly simplify the coding.

Solution to Exercise 6.3.3:

The State Machine Example 2 model provides one solution to this problem.

State Machine Example 2	http://files.bookboon.com/ai/State-Machine-Example-2.html

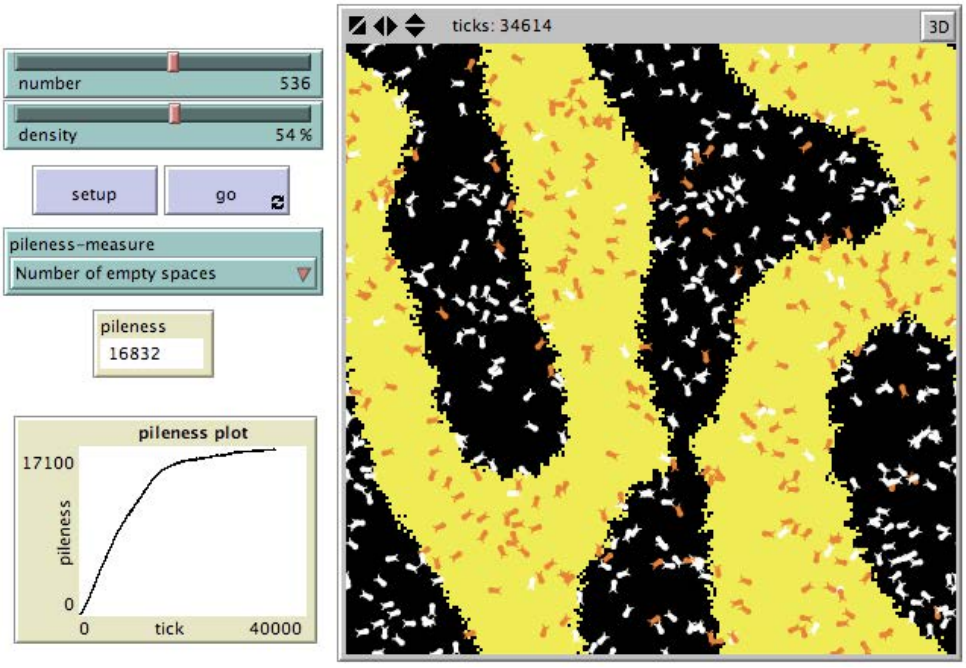


Figure S-6.3.3. Screenshot of the Interface for the State Machine Example 2 model after the setup button has been pressed followed by the go button, with the slider values as shown in the image.

The `Information` for this model is the same as that provided by the State Machine Example model, except for the following:

WHAT IS IT?

This model has been slightly modified by Bill Teahan from the State Machine Example model in NetLogo's Models Library. It has added a plot to graph the growth in the number of piles, or equivalently, the number of spaces, during the simulation. This is in order to estimate the progress of the self-organisation of the system as a whole.

WHAT IS ITS PURPOSE?

The purpose of the modified model is to ascertain whether a function based on global features can be used to measure the state of the simulation (i.e. how self-organised the system is). If the graph of the function shows a smooth upward curve, then it might be useful (for example) as a fitness function for an evolutionary algorithm in order to evolve the behaviour of the termites automatically rather than being explicitly programmed.

HOW TO USE IT?

First select a function to plot using the `pileness-measure` chooser. Then press the `setup` button to reset the simulation, and then press the `go` button.

THE INTERFACE

The buttons in the `Interface` are defined as follows:

- `setup`: This resets the simulation so that the wood chips and termites are placed randomly.
- `go`: This runs the simulation.

The sliders, chooser, monitor and plot in the `Interface` are defined as follows:

- `number`: This sets the number of termites that will be created at the start of the simulation.
- `density`: This sets the percentage of patches that will be created as wood chips at the start of the simulation.
- `pileness-measure`: These are the functions that can be used to measure the self-organisation of the system i.e. how much of the wood chips have been organised into piles. There are two possible functions:
  `"Number of crowded spaces"`: This counts the number of patches which have more than 7 yellow-coloured neighbours.
  `"Number of empty spaces"`: This counts the number of patches which have more than 7 black-coloured neighbours.
- `pileness`: This reports the value of the pileness measure as computed by the function chosen by the `pileness-measure` chooser.
- `pileness plot`: This plots the pileness value for each tick of the simulation.

THINGS TO TRY

Try devising your own pileness measures, or alternative functions to measure the self-organisation of the system.

RELATED MODELS

See the Termites model in the Models Library.

Solution to Exercise 6.3.4:

Self-organisation itself is an emergent property of the system, so your answer to Exercise 6.3.3 provides one way to measure when an emergent property of this type has occurred. But the measure is problem specific – i.e. the pileness measures devised for Exercise 6.3.2 will work for the Termites and State Machine Example 2 models but will be less useful for other models.

A more difficult problem is whether *any* emergent property or *any* form of self-organisation can be determined when it occurs. This might be useful when examining many different simulations to determine if some interesting or unforeseen property of the system arises. In some multi-agent systems, however, this might happen rarely. Computers are good at running many simulations and at collecting statistics about them, but when a property of the system cannot be predicted at startup due to its complex nature, then it can become problematical to signal that an emergent property has arisen or that self-organisation has occurred, especially if the observer of the simulations (usually us) has no knowledge in advance that they could occur.

Solution to Exercise 6.5.1:

The Moths and Virus models do not exhibit any self-organisation, so therefore are not examples of swarm intelligence. The other models all make use of the environment in some manner through stigmergic local knowledge to co-ordinate their activities to produce complex structures through self-organisation. For the Ants, and Termites models, physical objects (i.e. chemical scent and woodchips) are placed or moved about in the environment and it is the way that this is done that leads the agents to self-organise. For the other three models, the stigmergic local knowledge is present in the local positions or localised flashing of the other agents.

Solution to Exercise 7.7.1:

For working out solutions to Exercise 7.71, try out the Entropy Calculator model in NetLogo.

Entropy Calculator	http://files.bookboon.com/ai/Entropy-Calculator.nlogo

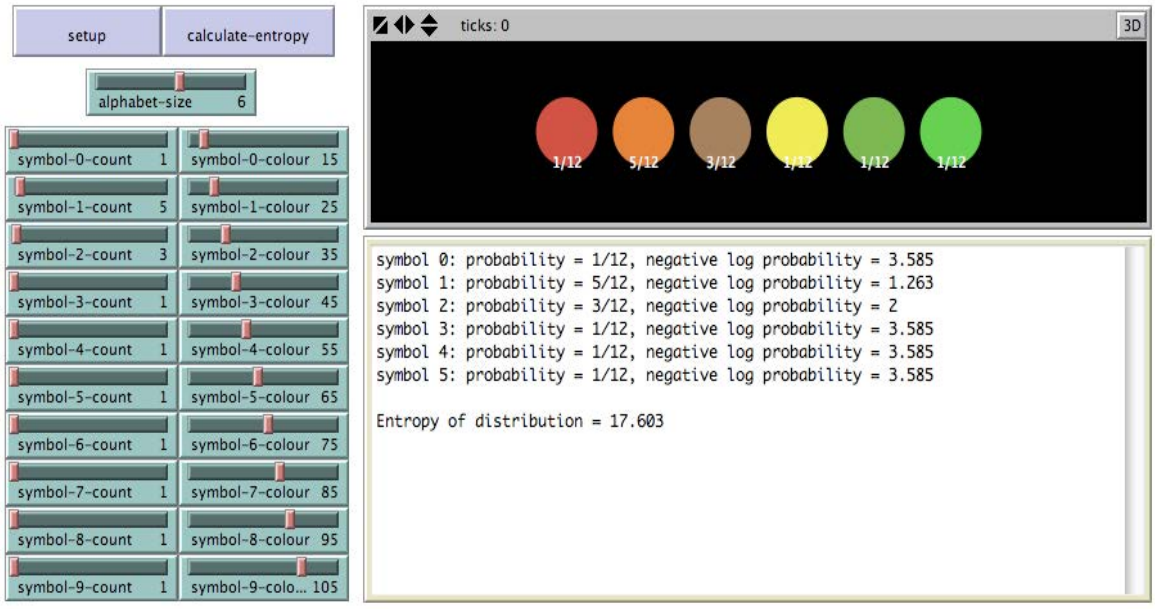


Figure S-7.7.1. Screenshot of the Interface for the Entropy Calculator model after the setup button has been pressed followed by the calculate-entropy button, with the slider values as shown in the image.

WHAT IS IT?

This model allows the user to calculate the entropy for a specific probability distribution.

The analogy is that there is an imaginary bag that a person pulls coloured balls from. The range of colours that the ball can have is its "alphabet". Each colour is a "symbol". Each symbol has a frequency count which can be used to determine its probability by dividing the count by the sum of the counts from all the symbols. The probabilities therefore sum to 1 and form the probability distribution for the set of balls.

Entropy provides a measure of how much variation there is in the probability distribution. It will be minimum when the symbols are all equiprobable. If one of the symbols have a higher count than the others, then the entropy will be correspondingly higher than the minimum when all the counts are equal.

WHAT IS ITS PURPOSE?

The purpose of this model is show how entropy calculations are made given a probability distribution.

HOW IT WORKS

A `symbol` turtle agent is used to represent the symbols in the distribution. Each symbol owns a frequency variable which is its count. The entropy is calculated by summing all frequency counts for all symbols first to find out the total count, then by calculating the sum of the negative log (to the base 2) of the symbol probabilities.

HOW TO USE IT

First select the size of the alphabet using the `alphabet-size` slider. Then select the frequency counts and desired colours using the respective sliders for each symbol. The sliders for symbols whose number is greater than or equal to the alphabet size will be ignored.

Then press the `setup` button followed by the `calculate-entropy` button.

THE INTERFACE

The model's `Interface` buttons are defined as follows:

- `setup`: This clears the environment and then draws coloured circles to represent the balls in the distribution.
- `calculate-entropy`: This calculates the entropy for the distribution.

The model's `Interface` sliders are defined as follows:

- `alphabet-size`: This sets the number of symbols (coloured balls) in the distribution.

The remaining sliders have the following naming convention:

`<symbol>-<number>-<count> and <symbol>-<number>-<colour>.`

This sets the count and colour for the specified symbol. If a number of the symbol is ≥ the `alphabet-size`, then these symbols are ignored in the calculations.

These counts are used to determine the probability for a specific symbol using the following formula:

$$P(symbol) = C(symbol) / C_{total}$$

where $C(symbol)$ is the count for the symbol and C_{total} is the sum of all counts for all symbols.

The model's Output is used to show how the entropy value is calculated.

THINGS TO TRY

Try changing the distribution counts in the sliders to see what affect this has on the entropy calculations.

Try to find out the conditions when the entropy is minimised.

Then try to determine if there is a maximum entropy value. To do this, edit one of the sliders to increase the maximum frequency count for a particular symbol. Increase the maximum count from 100 to 1000. What does this do to the entropy as a result? Then increase it to 1000000. What happens?

Solution to Exercise 7.7.2:

Minimum entropy occurs when all the symbols are equiprobable.

Solutions to Exercise 8.3.1, 8.6.2 and 8.7.1:

Solution to these exercises can be obtained by running the Searching For Kevin Bacon 2 model. Press the `Load Network from a File` button in the `Interface`, then load the following data file for Exercises 8.3.1 and 8.6.2:

Network-1	http://files.bookboon.com/ai/Network-1

Or load the following data file for Exercise 8.7.1:

Network-2	http://files.bookboon.com/ai/Network-2

Set the `search-behaviour` chooser to the relevant search algorithm. Also set the value of the `start-node-number` slider to 1 and make sure the `output-trace?` switch is set to `On`. For the informed searches, set the `heuristic` chooser to `Manhattan Distance`.

Descriptions of this exercise for each search algorithm can be generated using the `Generate Problem Description` button. Then to generate a trace of the search as it proceeds, first press the `setup-searcher` button, and then press the `go-searchers` button.

The output produced for the Breadth First Search on Network-1 is as follows:

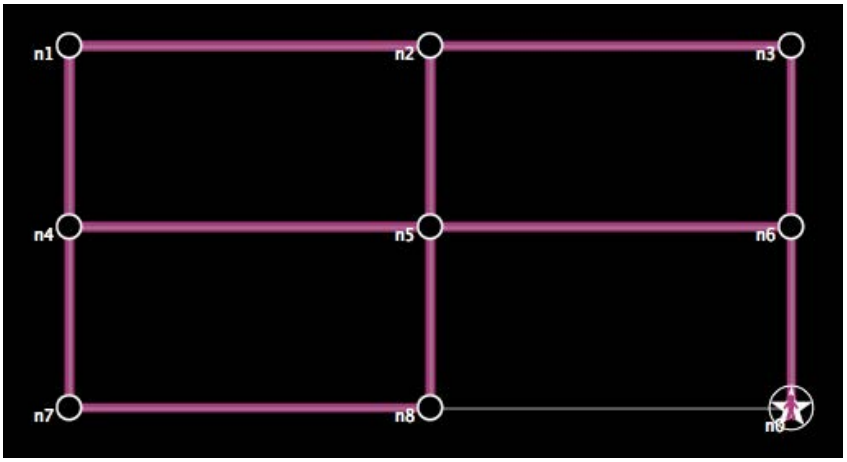


Figure 8.3.1.1 Screenshot of the Environment for the Searching for Kevin Bacon 2 model showing how the Breadth First Search searches the network shown loaded from the Network-1 data file.

The searcher agent 9 is starting at node 1.
Expanding the search for searcher 9 by hatching new searcher 10 at node n2.
Expanding the search for searcher 9 by hatching new searcher 11 at node n4.
Expanding the search for searcher 11 by hatching new searcher 12 at node n1.
Expanding the search for searcher 11 by hatching new searcher 13 at node n5.
Expanding the search for searcher 11 by hatching new searcher 14 at node n7.
Expanding the search for searcher 10 by hatching new searcher 15 at node n1.
Expanding the search for searcher 10 by hatching new searcher 16 at node n3.
Expanding the search for searcher 10 by hatching new searcher 17 at node n5.
Expanding the search for searcher 16 by hatching new searcher 18 at node n2.
Expanding the search for searcher 16 by hatching new searcher 19 at node n6.
Expanding the search for searcher 13 by hatching new searcher 20 at node n2.
Expanding the search for searcher 13 by hatching new searcher 21 at node n4.
Expanding the search for searcher 13 by hatching new searcher 22 at node n6.
Expanding the search for searcher 13 by hatching new searcher 23 at node n8.
Expanding the search for searcher 12 by hatching new searcher 24 at node n2.
Expanding the search for searcher 12 by hatching new searcher 25 at node n4.
Expanding the search for searcher 15 by hatching new searcher 26 at node n2.
Expanding the search for searcher 15 by hatching new searcher 27 at node n4.
Expanding the search for searcher 14 by hatching new searcher 28 at node n4.
Expanding the search for searcher 14 by hatching new searcher 29 at node n8.
Expanding the search for searcher 17 by hatching new searcher 30 at node n2.
Expanding the search for searcher 17 by hatching new searcher 31 at node n4.
Expanding the search for searcher 17 by hatching new searcher 32 at node n6.
Expanding the search for searcher 17 by hatching new searcher 33 at node n8.
Expanding the search for searcher 32 by hatching new searcher 34 at node n0.
Searcher agent 34 has found the goal node n0.

The output produced for the Depth First Search on Network-1 is as follows:

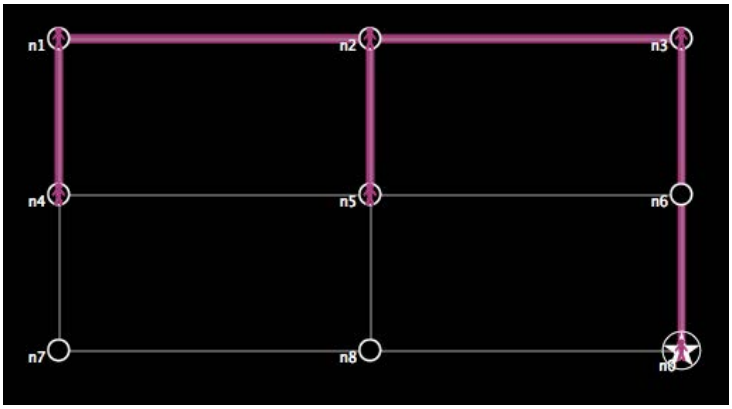


Figure 8.3.1.2 Screenshot of the Environment for the Searching for Kevin Bacon 2 model showing how the Depth First Search searches the network shown loaded from the Network-1 data file.

The searcher agent 9 is starting at node 1.
Expanding the search for searcher 9 by hatching new searcher 10 at node n2.
Expanding the search for searcher 9 by hatching new searcher 11 at node n4.
Expanding the search for searcher 10 by hatching new searcher 12 at node n1.
Expanding the search for searcher 10 by hatching new searcher 13 at node n3.
Expanding the search for searcher 10 by hatching new searcher 14 at node n5.
Expanding the search for searcher 12 by hatching new searcher 15 at node n2.
Expanding the search for searcher 12 by hatching new searcher 16 at node n4.
Expanding the search for searcher 15 by hatching new searcher 17 at node n1.
Expanding the search for searcher 15 by hatching new searcher 18 at node n3.
Expanding the search for searcher 15 by hatching new searcher 19 at node n5.
Expanding the search for searcher 18 by hatching new searcher 20 at node n2.
Expanding the search for searcher 18 by hatching new searcher 21 at node n6.
Expanding the search for searcher 21 by hatching new searcher 22 at node n0.
Searcher agent 22 has found the goal node n0.

The output produced for the Greedy Best First Search on Network-1 is as follows:

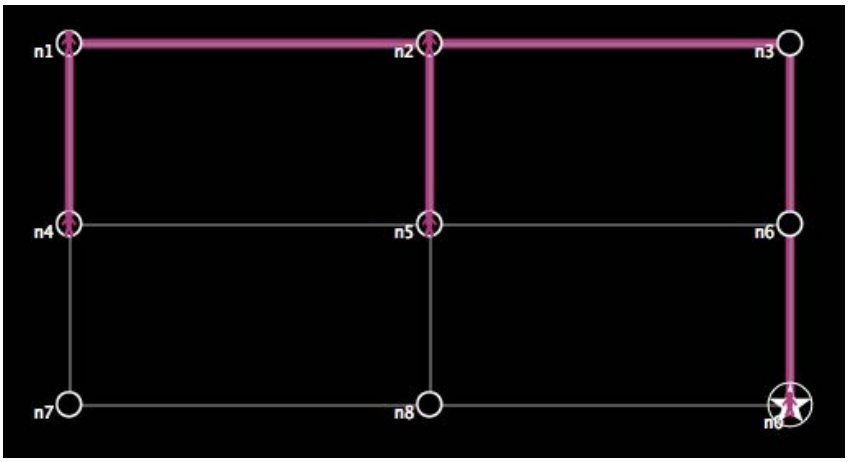


Figure 8.3.1.3 Screenshot of the Environment for the Searching for Kevin Bacon 2 model showing how the Greedy Best First Search using the heuristic Manhattan Distance searches the network shown loaded from the Network-1 data file.

The searcher agent 9 is starting at node 1.
Expanding the search for searcher 9 by hatching new searcher 10 at node n2.
Path cost of searcher 10 = 20
Expanding the search for searcher 9 by hatching new searcher 11 at node n4.
Path cost of searcher 11 = 10
Expanding the search for searcher 10 by hatching new searcher 12 at node n1.
Path cost of searcher 12 = 40
Expanding the search for searcher 10 by hatching new searcher 13 at node n3.
Path cost of searcher 13 = 40
Expanding the search for searcher 10 by hatching new searcher 14 at node n5.
Path cost of searcher 14 = 30
Expanding the search for searcher 13 by hatching new searcher 15 at node n2.
Path cost of searcher 15 = 60
Expanding the search for searcher 13 by hatching new searcher 16 at node n6.
Path cost of searcher 16 = 50
Expanding the search for searcher 16 by hatching new searcher 17 at node n0.
Path cost of searcher 17 = 60
Searcher agent 17 has found the goal node n0.

The output produced for the A* Search on Network-1 is as follows:

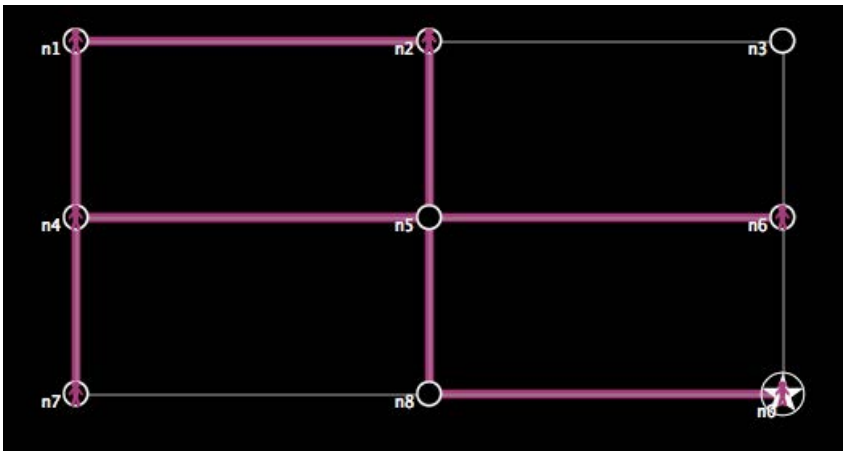


Figure 8.3.1.4 Screenshot of the Environment for the Searching for Kevin Bacon 2 model showing how the A* Search using the heuristic Manhattan Distance searches the network shown loaded from the Network-1 data file.

The searcher agent 9 is starting at node 1.
Expanding the search for searcher 9 by hatching new searcher 10 at node n2.
(Path cost, estimated cost, combined cost) of searcher 10 = (20, 40, 60)
Expanding the search for searcher 9 by hatching new searcher 11 at node n4.
(Path cost, estimated cost, combined cost) of searcher 11 = (10, 50, 60)
Expanding the search for searcher 11 by hatching new searcher 12 at node n1.
(Path cost, estimated cost, combined cost) of searcher 12 = (20, 60, 80)
Expanding the search for searcher 11 by hatching new searcher 13 at node n5.
(Path cost, estimated cost, combined cost) of searcher 13 = (30, 30, 60)
Expanding the search for searcher 11 by hatching new searcher 14 at node n7.
(Path cost, estimated cost, combined cost) of searcher 14 = (20, 40, 60)
Expanding the search for searcher 13 by hatching new searcher 15 at node n2.
(Path cost, estimated cost, combined cost) of searcher 15 = (40, 40, 80)
Expanding the search for searcher 13 by hatching new searcher 16 at node n4.
(Path cost, estimated cost, combined cost) of searcher 16 = (50, 50, 100)
Expanding the search for searcher 13 by hatching new searcher 17 at node n6.
(Path cost, estimated cost, combined cost) of searcher 17 = (50, 10, 60)
Expanding the search for searcher 13 by hatching new searcher 18 at node n8.
(Path cost, estimated cost, combined cost) of searcher 18 = (40, 20, 60)
Expanding the search for searcher 18 by hatching new searcher 19 at node n0.
(Path cost, estimated cost, combined cost) of searcher 19 = (60, 0, 60)
Searcher agent 19 has found the goal node n0.

The output produced for the Hill Climbing Search on Network-2 is as follows:

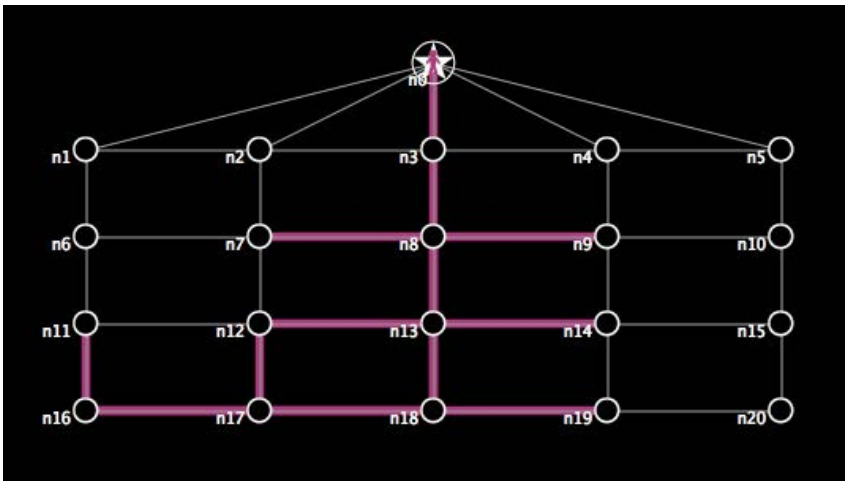


Figure 8.3.1.5 Screenshot of the Environment for the Searching for Kevin Bacon 2 model showing how the Hill Climbing Search using the heuristic Manhattan Distance searches the network shown loaded from the Network-2 data file.

The searcher agent 21 is starting at node 16.
Expanding the search for searcher 21 by hatching new searcher 22 at node n11.
Height of searcher 22 = 35
Expanding the search for searcher 21 by hatching new searcher 23 at node n17.
Height of searcher 23 = 30
Expanding the search for searcher 23 by hatching new searcher 24 at node n12.
Height of searcher 24 = 25
Expanding the search for searcher 23 by hatching new searcher 25 at node n16.
Height of searcher 25 = 40
Expanding the search for searcher 23 by hatching new searcher 26 at node n18.
Height of searcher 26 = 20
Expanding the search for searcher 26 by hatching new searcher 27 at node n13.
Height of searcher 27 = 15
Expanding the search for searcher 26 by hatching new searcher 28 at node n17.
Height of searcher 28 = 30
Expanding the search for searcher 26 by hatching new searcher 29 at node n19.
Height of searcher 29 = 30
Expanding the search for searcher 27 by hatching new searcher 30 at node n8.
Height of searcher 30 = 10
Expanding the search for searcher 27 by hatching new searcher 31 at node n12.
Height of searcher 31 = 25
Expanding the search for searcher 27 by hatching new searcher 32 at node n14.
Height of searcher 32 = 25

Expanding the search for searcher 27 by hatching new searcher 33 at node n18.
Height of searcher 33 = 20
Expanding the search for searcher 30 by hatching new searcher 34 at node n3.
Height of searcher 34 = 5
Expanding the search for searcher 30 by hatching new searcher 35 at node n7.
Height of searcher 35 = 20
Expanding the search for searcher 30 by hatching new searcher 36 at node n9.
Height of searcher 36 = 20
Expanding the search for searcher 30 by hatching new searcher 37 at node n13.
Height of searcher 37 = 15
Expanding the search for searcher 34 by hatching new searcher 38 at node n0.
Height of searcher 38 = 0
Searcher agent 38 has found the goal node n0.

Solution to Exercise 9.1.2:

What is a "Knowledge Base?" and what must it be made of? There are many answers to these questions in the literature. For example, one possible answer might be as follows: "*A Collection of concepts about a domain or domains*". But what is a 'concept' and what is a 'domain'?

Another answer, from the Sci-Tech Dictionary, is: "*A computer system whose usefulness derives primarily from a data base containing human knowledge in a computerized format*". This definition does not explain what 'human knowledge' is – that is, it avoids answering the fundamental question at the heart of these questions – "What is 'Knowledge'?"

A third answer, provided by www.wikipedia.org, is as follows: "*Knowledge based systems are artificial intelligent tools working in a narrow domain to provide intelligent decisions with justification. Knowledge is acquired and represented using various knowledge representation techniques such as rules, frames and scripts*". The first sentence accurately describes the function of a knowledge base system, but not what it is made of. The second sentence uses the word '*knowledge*' twice without defining what it is.

A fourth answer can take an agent-oriented perspective where the knowledge-based system is considered as an agent whose task is to impart its knowledge to the users of the system. If 'information' is defined as 'data (i.e. numbers, text, images, etc.) that is helpful in answering a question', then an agent can be deemed to have 'knowledge' if it knows how to use information to answer a question in a particular context. In other words, the agent is deemed (by us as users of the system) to be 'knowledgeable' because it knows how to answer our questions in a knowledgeable manner. In this definition, 'knowledge' is explicitly associated with some agent; and question/answering is a key part to being "knowledgeable".

With this definition, a knowledge-based system must act like a knowledgeable agent. Humans judge the agent to be knowledgeable (or not as the case may be). i.e. A secondary agent is doing the testing, similar to the Turing Test. Just as importantly, it is the interaction between the testing agent and the (possibly) knowledgeable agent that is the most important part of the process.

This is a much stricter definition of what a knowledge-based system is, and in this light, a traditional logic-based definition (which states that rules of inference are the important distinction to distinguish between a knowledge-base and database) seems inadequate, as it requires someone to know the meaning of the output of the inference process i.e. what it all means, and how we can apply the knowledge gained through the inference.

Note: In expert systems, quite often the "knowledge-base" is defined as being the facts plus rules plus inferencing system, whereas the database is simply the facts. However, with this definition, can the system justifiably be called a repository of "knowledge" in the common meaning of the term?

Solution to 9.2.1:

Whether you agree or disagree with this statement (that "Knowledge cannot exist by itself, independent of us. It only exists in our heads") depends on your definition of what 'knowledge' is. A classical definition of knowledge pre-supposes that it is an object in and of itself, and therefore can be stored and retrieved in a repository. This relates to the classical/traditional definition of human memory which relies on common metaphors found in natural language of memory being a place which we store and retrieve information as in a database.

However, an embodied cognitivist perspective would have us believe that knowledge cannot exist outside our own heads. Here the important condition is one of embodiment. An agent exists as an object embodied within an environment, and must interact and react with that environment. In Gärdenfors' conceptual space theory (see Section 9.4), each embodied agent has his or her own definitions of concepts he or she believes are justified and relevant, and although cultural interactions will ensure that there will be close matches with the concepts held by other agents in that environment, there will be subtle differences with no exact match. Under this theory, then the concepts upon which knowledge arises must necessarily be within our own heads.

Another way of looking at this is to look at the word 'knowledge' itself. The base form of the word contains the word 'know'. This is a verb, and in common English language, the act of 'knowing' is associated with an agent, usually a human one. We say that we know something, not that a car or a tree knows something. So knowledge is necessarily associated with us as human agents because it is associated with the process of knowing.

One further point to note: Consider the scenario that the knowledge of how to construct a nuclear bomb has been lost (presumably because of a world-wide nuclear holocaust). Now the classical view would be that this knowledge exists independently of the people who know about it – i.e. what it means, and how to apply the knowledge. In this scenario, then it should be possible then for someone in the future to come along and then 'find' this knowledge again (let us say it was written down in a book somewhere and the book was found in some hidden library or archive). However, the process of recovering the knowledge, i.e. deciphering the meaning of the written text, requires that the future agent must develop its own understanding of the problems involved – and then the new knowledge is once again in someone's head.

Notwithstanding this line of reasoning, someone could define knowledge as "*information that can be used to help solve a particular task*". Under this definition, the knowledge-as-an-object argument fits that definition. If, however, you define knowledge as "*knowing how to use information to help solve a particular task*", then this definition pre-supposes an agent is involved, who must define what it means to know something, and what information is meaningful.

Solution to Exercise 9.4.2:

[*Below is a short answer to this question. A full answer should provide the gist of the following answer, plus also demonstrate through examples an understanding of the fundamental differences in the three different approaches*].

In all three approaches, there are problems in describing the knowledge required to describe each of the five examples. Note that the five senses are all represented here – hearing, taste, feeling, smell and sight. Rather than dealing with the problems with each sense, it is simpler to deal with the overall problem of sensing. Sensing and perception is necessarily associated with an agent embodied in an environment. Hence, to describe the knowledge requires a cognitivist approach. In that light, conceptual spaces theory offers the best tool to give us some leverage into what is essentially a very difficult problem. There are major problems with the symbolic layer – not least to do with the issues to do with embodiment. i.e. The existence of the agent within an environment, and understanding of the interaction of the agent with everything that exists within it; and being able to perceive and sense changes of the environment. Related problems occur for the neural network/connectionist layer – it does not deal with the issue of embodiment. What is required is some concrete formalisation or representation of the knowledge (i.e. with physical properties related to embodiment).

For example, with conceptual spaces, the concept of colour is represented by three dimensions and taste by four dimensions. This gives a direct correspondence to physical properties which are based on the lower connectionist layers where the signals are being processed (of brightness for colour, for example). At the top level, concepts can be structured to form symbolic representations. [*For the other three senses, a full answer should look at some possible quality dimensions – see Gärdenfors conceptual spaces theory – that describe that particular sense well*].

Solution to Exercise 9.7.2: Map and Image Annotator Model

The Map and Image Annotator NetLogo model provides one solution to this problem:

Map and Image Annotator	http://files.bookboon.com/ai/Map-and-Image-Annotator.nlogo

Figure 9.7.2. Screenshot of the Interface for the Map and Image Annotator model with a sample image imported into the environment and sample annotation.

WHAT IS IT?

This model allows the user to annotate a map or an image. The model imports an image into the background for the environment. The user can then add arrows, circles, text and marks (i.e. x's). These can be later edited or erased if required.

HOW IT WORKS

The map or image is placed in the background using the `import-drawing` command. Separate turtle agent breeds – `arrow`, `circle`, `text` and `mark` – are used for drawing the corresponding object in the environment. A further `side` turtle agent is used for drawing the sides of the rectangular box used by the `select-region` and `erase-region` buttons for selecting and erasing multiple objects.

Arrows are drawn using two `arrow` turtle agents. The start arrow turtle agent is given a circle shape, and the end arrow turtle agent is drawn using NetLogo's default arrowhead shape. The shaft of the arrow is drawn by creating a link between the two arrow turtle agents. Circles and marks are standard turtle agents with a specific size, colour and thickness drawn at a specific location using the mouse. Text is drawn by setting the label of turtle agents that are drawn with size 0 - i.e. only the label of the agent is made visible, not the agent itself.

NetLogo's `distancxy` reporter is used to find the nearest object by the `select-object` and `erase-object` procedures. The `watch` command is used to highlight the selected object. The global variable `selected` is used by the `select-region` and `erase-region` procedures to hold an agentset of objects that are within the specified region. The entire agentset can then easily be dragged together or erased in unison using a single `ask` command.

HOW TO USE IT

Pressing `setup` clears the environment and then loads the image as specified by the `map-or-image-to-use` chooser. Then the user can place arrows, circles, text and marks over the top of the image using the `draw-arrow`, `draw-circle`, `draw-text` and `draw-mark` buttons by pointing then clicking using the mouse.

THE INTERFACE

The buttons in the `Interface` are defined as follows:

- `setup`: This clears the environment and loads a new image as specified by the `map-or-image-to-use` chooser
- `draw-arrow`: This will draw an arrow over the top of the image. If the user then clicks and drags the mouse, an arrow is drawn with the tail at the point the mouse was clicked, and the head at the point the mouse was released. (The mouse must be both clicked and dragged, otherwise no arrow is drawn). The colour and dimensions of the arrow are specified by the `arrow-colour`, `arrow-start-size`, `arrow-end-size` and `arrow-thickness` sliders. To get just a single arrow head shape without the shaft and arrow tail, then use the `erase-object` button to erase the end of the arrow. (This can also be used to erase the arrow head and leave just the tail if so desired).

- `draw-circle`: This will draw a circle over the top of the image at the point the mouse is next clicked. The colour and dimensions of the circle are specified by the `circle-colour`, `circle-size` and `circle-thickness` sliders.
- `draw-text`: This will draw text over the top of the image at the point the mouse is next clicked. Note that the right side of the text will be set to the mouse click point (rather than the left). The content of the text is specified by the `text-to-insert` input box. To update this, the user must click inside the input box, then edit the existing text or clear it and replace it with new text. The text update is only completed when the user types the return key. The colour of the text is specified by the `text-colour` slider.
- `draw-mark`: This will draw a mark over the top of the image at the point the mouse is next clicked.
- `select-object`: This will select an object if it is near to the next mouse click point. The object will have a halo placed around it to show when it has been selected. Subsequently dragging the mouse will result in the selected object being dragged around. An object is an arrow head or tail, a circle, text or a mark. When dragging the head or tail of an arrow, the other end will remain stationary (i.e. the arrow will pivot around the other end).
- `erase-object`: This will erase an object if it is near to the next mouse click point. If an arrow head or tail is erased, then the shaft is erased as well, and only the other end will remain.

- `select-region`: This allows multiple objects to be selected. Click using the mouse then drag. A rectangular box will delimit the region of objects that are selected. Temporarily stop clicking the mouse, then click back somewhere inside the rectangular box, then drag it and everything inside it using the mouse. All the objects inside the box will move to the point where the mouse is dragged to.

The `Interface`'s choosers, sliders and input box are defined as follows:

- `map-or-image-to-use`: This specifies which image gets loaded into the environment. If set to "`User-specified`", then the user will be prompted for an image filename. (This will not work when the model is invoked in an applet).
- `arrow-colour`: This sets the colour of the arrow that is drawn when the `draw-arrow` button is next pressed. The colour is specified in a range from 0 to 140 using NetLogo's colour numbering scheme. Sample colour numbers are shown in the small text note at the left bottom of the `Interface`.
- `arrow-start-size`: This sets the size of the arrow tail (drawn as a filled-in circle).
- `arrow-end-size`: This sets the size of the arrow head (drawn using the default arrowhead shape).
- `arrow-thickness`: This sets the thickness of the arrow's shaft.
- `circle-colour`: This sets the colour of the circle that is drawn when the `draw-circle` button is next pressed. The colour is specified in a range from 0 to 140 using NetLogo's colour numbering scheme. Sample colour numbers are shown in the small text note at the left bottom of the `Interface`.
- `circle-size`: This sets the size of the circle.
- `circle-thickness`: This sets the thickness of the circle.
- `text-to-insert`: This sets the text that is drawn when the `draw-text` is next pressed. The text is right justified rather than left justified. i.e. The end of the text (not the start) will end up at the point where the mouse is clicked. The size of the text can be specified globally using the `Settings` button at the top of the `Interface`.
- `text-colour`: This sets the colour of the text that is drawn when the `draw-text` button is next pressed. The colour is specified in a range from 0 to 140 using NetLogo's colour numbering scheme. Sample colour numbers are shown in the small text note at the left bottom of the `Interface`.
- `mark-colour`: This sets the colour of the mark (i.e. a letter 'x') that is drawn when the `draw-mark` button is next pressed. The colour is specified in a range from 0 to 140 using NetLogo's colour numbering scheme. Sample colour numbers are shown in the small text note at the left bottom of the `Interface`.
- `mark-size`: This sets the size of the mark.

THINGS TO NOTICE

Notice that some of the images do not fit the environment exactly (there are white unfilled areas above and below the image in the environment). This is because the `import-drawing` command maintains the same aspect ratio of the image (i.e. the width and height have the same ratio relative to each other as the image is scaled). The image, however, is sized to fit the environment, and therefore to keep the same aspect ratio, there will be some unfilled areas in the environment as a result.

THINGS TO TRY

Try using the model to annotate your own map or image, such as a photo you might have taken on holiday. Then use the `print screen` feature on your computer to save the annotated image.

EXTENDING THE MODEL

Try modifying the model so that it allows the user to draw lines and add further objects and.

NETLOGO FEATURES

The code makes use of the `watch`, `subject` and `perspective` commands when the `select-object` button is used to place a halo around the selected object.

RELATED MODELS

See the Mouse Drag Multiple Example and Mouse Drag One Example models.

CREDITS AND REFERENCES

The model makes use of code provided by Uri Wilensky in the Mouse Drag Multiple Example and Mouse Drag One Example models.

Solution to Exercise 9.9.1:

Sentence	Production Rules	First Order Logic
1. *Peter is an old man.*	*Fact database*: Peter is man, Peter is old.	*Man* (*Peter*) ∧ *Old* (*Peter*).
2. *Mary is the sister of Peter.*	*Fact database*: Mary is sister of Peter.	*Sister* (*Mary*, *Peter*).
3. *Some cats have large ears.*	*Fact database*: Unknown is cat, Unknown has large ears.	$\exists x\, Cat(x) \wedge Large\_ears(x).$
4. *If X is the sister of Y, and Y is a boy, then Y is the brother of X.*	*Rules database*: IF x is sister of y and y is boy THEN y is brother of x.	∀x∀y *Sister* (*x, y*) ∧ *Boy* (*y*) ⇒ *Brother* (*y*, *x*).
5. *Simba is a white elephant.*	*Fact database*: Simba is elephant, Simba is white.	*Elephant* (*Simba*) ∧ *White* (*Simba*).
6. *All elephants are mammals.*	*Rules database*: IF x is elephant THEN x is mammal.	∀x *Elephant* (*x*) ⇒ *Mammal* (*x*).
7. *Elephants have a trunk and large ears.*	*Rules database*: IF x is elephant THEN x has trunk AND x has large ears.	∀x *Elephant* (*x*) ⇒ *Has_trunk*(*x*) ∧ *Has_large_ears*(*x*).
8. *Ruby is a black and white cat*	*Fact database*: Ruby is cat, Ruby is black, Ruby is white.	*Cat* (*Ruby*) ∧ *Black* (*Ruby*) ∧ *White* (*Ruby*).
9. *Ruby is the mother of Pearl and Pearl is the daughter of Ruby.*	*Fact database*: Ruby is mother of Pearl, Pearl is daughter of Ruby.	*Mother* (*Ruby, Pearl*) ∧ *Daughter* (*Pearl, Ruby*).
10. *Some cats are small.*	*Fact database*: Unknown is cat, Unknown is small.	$\exists x(Cat(x) \wedge Small(x)).$
11. *All cats eat meat.*	*Rules database*: IF x is cat THEN x eats meat.	∀x *Cat* (x) ⇒ *Eats_meat* (x). *Or:* ∀x *Cat* (x) ⇒ *Carnivore* (x).
12. *Cats have whiskers and a tail.*	*Rules database*: IF x is cat THEN x has whiskers AND x has tail.	∀x *Cat* (x) ⇒ *Has_whiskers* (x) ∧ Has_tail (x).

Despite the relative simplicity of these examples (i.e. considering the text in the English sentences), many of these pose problems for the production rules representation and/or the first order logic representation. Where there is a straight relationship being expressed between nouns, such as in examples 2, 4, 6 and 9, then the representations seem well suited to expressing the knowledge. However, when there is an adjective modifying a noun, as in examples 1, 3, 5, 7, 8, and 10, then the adjective in the rules or logic statements plays the same role as the noun (i.e. *Cat* (*Ruby*) ∧ *Black* (*Ruby*)) rather than modifying it, and in so doing, this describes a slightly different relationship than was expressed in the original text. Similarly, when there is a verb other than the verbs 'is' or 'are' in the text, as in the examples 3, 7, 11 and 12, then both the production rules and first order logic statements have to use a contrived mechanism for expressing the knowledge – for example, 'eats meat' / '*Eats_meat*' and 'has whiskers' / '*Has_whiskers*'. Alternative representations in these cases also end up being less than satisfactory by not completely capturing the knowledge being expressed in the text.

Solution to Exercise 9.9.2:

(i) $\exists x(Car(x) \wedge Red(x))$
Some cars are red.

(ii) $\forall x(Cake(x) \Rightarrow Sweet(x))$
All cakes are sweet.

(iii) $\exists x \forall y(Person(x) \wedge Cake(y) \wedge \neg Likes(x,y))$
Not all people like cakes / some people do not like cakes.

Solution to Exercise 9.9.3:

(i) All cats like fish.
$\forall x(Cat(x) \Rightarrow Likes(x,Fish))$

(ii) Bananas are either green or yellow or brown.
$\forall x(Banana(x) \Rightarrow Green(x) \vee Yellow(x) \vee Brown(x))$

(iii) Some people eat everything.
$\exists x \forall y(Person(x) \wedge Eats(x,y))$

Solution to Exercise 9.10.1:

As a short-hand notation, we can convert the database and rules from the IF THEN format to the equivalent format shown below:

```
Database:
    A B C D
Rules base:
    Rule 1: X & Y & A ⇒ Z
    Rule 2: V & B ⇒ Y
    Rule 3: V & C ⇒ X
    Rule 4: D ⇒ V
```

Answer for i.: (Rules fired are marked by "*")

```
Cycle 1:
    Database:
        A B C D
    Rules base:
        Rule 1: X & Y & A ⇒ Z
        Rule 2: V & B ⇒ Y
        Rule 3: V & C ⇒ X
    *   Rule 4: D ⇒ V

Action: Add V to database.

Cycle 2 :
    Database:
        A B C D V
    Rules base:
        Rule 1: X & Y & A ⇒ Z
    *   Rule 2: V & B ⇒ Y
    *   Rule 3: V & C ⇒ X
        Rule 4: D ⇒ V

Action: Add Y to database.

Action: Add X to database.
```

Cycle 3 :

Database:

A B C D V Y X

Rules base:

* Rule 1: X & Y & A ⇒ Z

Rule 2: V & B ⇒ Y

Rule 3: V & C ⇒ X

Rule 4: D ⇒ V

Action: Add Z to database.

Cycle 4 :

Database:

A B C V Y X Z

Goal Z reached.

Answer for ii.: (Found rules with goal or sub-goal marked by "*"; Proven rules marked by "*")

Step 1 :

Goal Z.

Database:

A B C D

Rules base:

* Rule 1: X & Y & A ⇒ Z

Rule 2: V & B ⇒ Y

Rule 3: V & C ⇒ X

Rule 4: D ⇒ V

Step 2 :

Sub-Goal X (from Rule 1).

Database:

A B C D

Rules base:

Rule 1: X & Y & A ⇒ Z

Rule 2: V & B ⇒ Y

* Rule 3: V & C ⇒ X

Rule 4: D ⇒ V

Step 3 :

Sub-Goal V (from Rule 3).

Database:
A B C D
Rules base:
Rule 1: X & Y & A ⇒ Z
Rule 2: V & B ⇒ Y
Rule 3: V & C ⇒ X
* Rule 4: D ⇒ V

Step 4 :

Sub-Goal V (from Rule 3).

Database:
A B C D
Rules base:
Rule 1: X & Y & A ⇒ Z
Rule 2: V & B ⇒ Y
Rule 3: V & C ⇒ X
* Rule 4: D ⇒ V

Action: Add V to database.

Step 2 :

Sub-Goal X (from Rule 1).

Database:
A B C D V
Rules base:
Rule 1: X & Y & A ⇒ Z
Rule 2: V & B ⇒ Y
* Rule 3: V & C ⇒ X
Rule 4: D ⇒ V

Action: Add X to database.

Step 5 :

Sub-Goal Y (from Rule 1).

Database:
A B C D V X
Rules base:
Rule 1: X & Y & A ⇒ Z
* Rule 2: V & B ⇒ Y
Rule 3: V & C ⇒ X
Rule 4: D ⇒ V

Step 6 :

Sub-Goal Y (from Rule 1).

Database:
A B C D V X
Rules base:
Rule 1: X & Y & A ⇒ Z
* Rule 2: V & B ⇒ Y
Rule 3: V & C ⇒ X
Rule 4: D ⇒ V

Action: Add Y to database.

```
Step 1 :
      Sub-Goal Z (from Rule 1).

        Database:
            A B C D V X Y
        Rules base:
      *     Rule 1: X & Y & A ⇒ Z
            Rule 2: V & B ⇒ Y
            Rule 3: V & C ⇒ X
            Rule 4: D ⇒ V

Action: Add Z to database.

Goal Z proven
```

Solution to Exercise 9.11.1:

There are various ways the slots and values in the frames can be defined for each of these statements. Some possible answers are as follows (note that 'AKO' stands for ('A Kind Of'):

1. *Peter is an old man.*

Frame	**Peter**
AKO	Man
Age	Old

2. *Mary is the sister of Peter.*

Frame	**Peter**
AKO	Man
Age	Old
Sister	Mary

Frame	**Mary**
Brother	Peter

3. *Some cats have large ears.*

Frame	**Unknown-cat**
AKO	Cat
Ears	Large

4. *If X is the sister of Y, and Y is a boy, then Y is the brother of X.*

Frame	**Sister**
AKO	Person
Gender	Female

Frame	**Brother**
AKO	Person
Gender	Male

Frame	**Girl**
AKO	Person
Gender	Female

Frame	**Boy**
AKO	Person
Gender	Male

5. *Simba is a white elephant.*

Frame	**Simba**
AKO	Elephant
Colour	White

6. *All elephants are mammals.*

Frame	**Elephant**
AKO	Mammal

7. *Elephants have a trunk and large ears.*

Frame	**Elephant**
AKO	Mammal
Has_parts	Trunk, Ears
Ears	Large

8. *Ruby is a black and white cat.*

Frame	**Ruby**
AKO	Cat
Colour	Black, White

9. *Ruby is the mother of Pearl and Pearl is the daughter of Ruby.*

Frame	**Ruby**
AKO	Cat
Colour	Black, White
Daughter	Pearl

Frame	**Pearl**
AKO	Cat
Mother	Ruby

10. *Some cats are small.*

Frame	**Unknown-cat**
AKO	Cat
Size	Small

11. *All cats eat meat.*

Frame	**Cat**
AKO	Mammal
Food	Meat

12. *Cats have whiskers and a tail.*

Frame	**Cat**
AKO	Mammal
Food	Meat
Has_parts	Whiskers, Tail

As occurred in Exercise 9.9.1 with production rules and first order logic, the frames method of knowledge representation forces the knowledge contained in many of these English sentences to be recast in order to fit in with the method of representation being used. This recasting often ends up resulting in subtle changes of the knowledge being expressed, or in the meaning not being satisfactorily represented. For example, the problems to do with how to represent the meaning of adjectives using production rules and first order logic that arose in Exercise 9.9.1 seem to be more naturally solved using frames, although the addition of slot names has added further knowledge that was not originally there in the English sentences (such as the Age, Gender, Colour and Size slot names for examples 1, 2, 4, 5, and 10). Representing the concept 'all' is naturally achieved using inheritance and default values (as in examples 6 and 11), but the concept 'some' ends up being problematic (as in examples 3 and 10). Also problematic are the examples where there is a verb other than 'is' or 'are' – as in examples 3, 7, 11 and 12. One often-used frame-based solution for 'has' and 'have' verbs is to use a 'has-parts' slot as in the solutions provided. It is not clear, however, what the best way is for representing verbs such as 'eat' as in example 11. Explicit rules as in example 4 are not naturally represented using frames, and it is not clear how best to represent such knowledge. Possible solutions are to use daemons (methods associated with a particular slot) or use a different form of inferencing via frame matching or frame inheritance.

Solution to Exercise 10.7.2:

There are many shortcomings of Eliza-like solutions such as that provided in the Chatbot NetLogo model which is based on using regular expressions to find basic patterns in the text. However, there are more sophisticated approaches that are based on regular expressions that hold promise. A common solution used for information extraction, for example, is that each stage of linguistic processing is based on regular expressions that extract some information from the text. This is then passed on to the next stage, eventually ending up with a set of templates as the output. The Fastus system, for example, uses the following six stages: tokenisation → extraction of complex words → processing of basic phrases → processing of complex phrases → extraction of semantic patterns → merging of semantic patterns into templates. For question answering, there is the added step of applying the information extraction to the question text, and matching the extracted templates.

However, there are still major hurdles that need to be overcome if information extraction and question answering technology are to be applied successfully to the problem of designing a conversational agent. Some of the hurdles that need to be overcome before the agent is believable (in the sense that it can pass the Turing Test) are as follows:

- It must understand and be able to employ common sense reasoning.
- It must have access to world knowledge.
- It needs to have some form of understanding of the real world (Brooks argues that an A.I. will have to be physically embodied in the real world).
- It needs to understand the context of the conversation and respond accordingly.
- It needs to be able to answer questions such as "What are you thinking about?" in a realistic way.
- And so on.

Perhaps the main shortcoming of present-day conversational agents is the lack of understanding of the context of the conversation: what has been said before, what needs to be said next, what is appropriate, and so on. Part of the problem is the need for access to general knowledge, as well as common sense reasoning, and these are well known stumbling blocks in Artificial Intelligence. A conversational agent also needs to understand the rules of conversation as it has to have a conversation with humans e.g. speech and dialogue acts in A.I. are possible solutions, but only go a step of the way towards producing an agent-oriented system with the ability to hold a genuine conversation with a human.