bookboon
The eBook company

Artificial Intelligence – Agent Behaviour

William John Teahan

William John Teahan

Artificial Intelligence – Agent Behaviour I

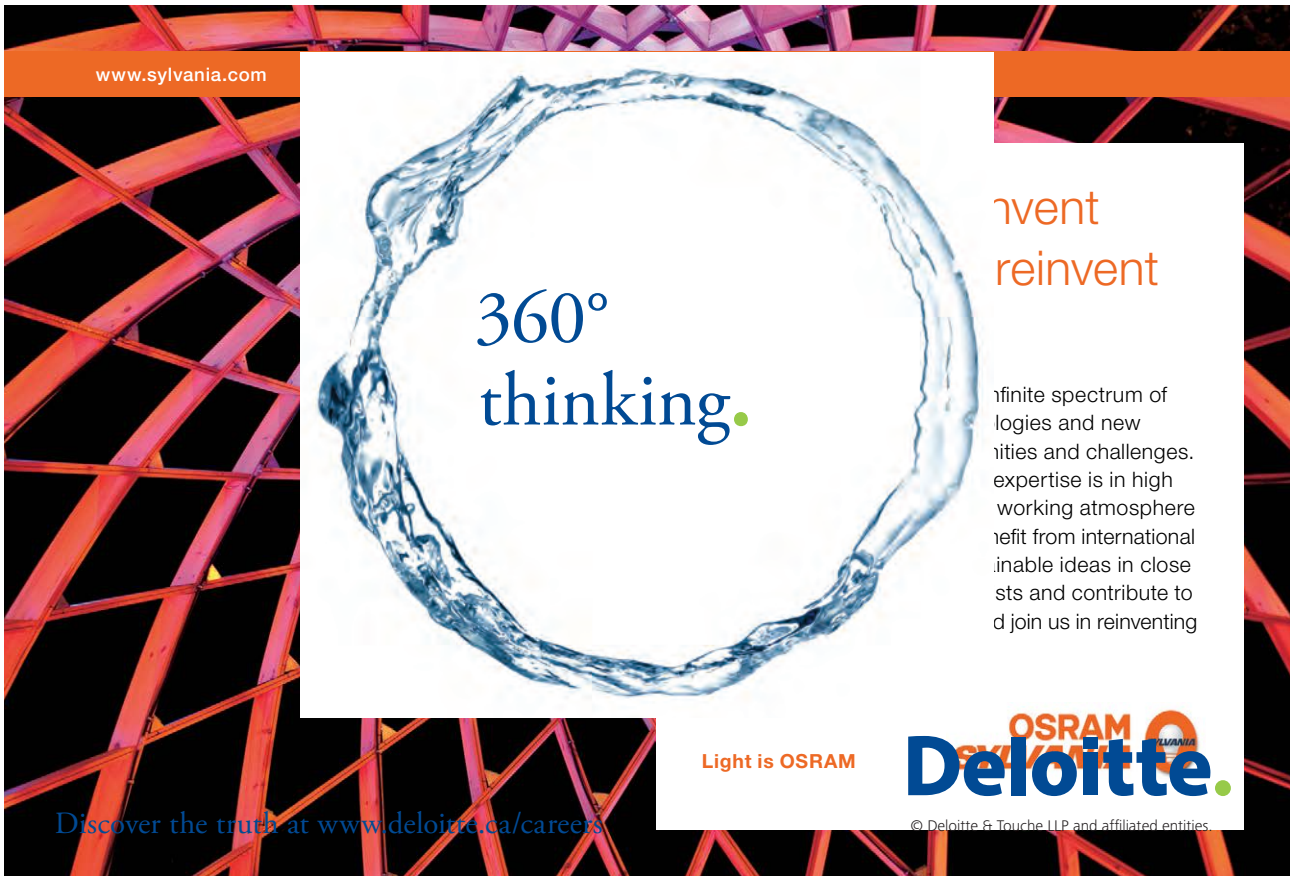
Download free eBooks at bookboon.com

Artificial Intelligence – Agent Behaviour I
1st edition
© 2010 William John Teahan & bookboon.com
ISBN 978-87-7681-559-2

Download free eBooks at bookboon.com

Contents

	Part 2 Agent Behaviour I	**8**
6	**Behaviour**	**9**
6.1	What is behaviour?	10
6.2	Reactive versus Cognitive Agents	11
6.3	Emergence, Self-organisation, Adaptivity and Evolution	13
6.4	The Frame of Reference Problem	20
6.5	Stigmergy and Swarm Intelligence	22
6.6	Implementing behaviour of Turtle Agents in NetLogo	24
6.7	Boids	36
6.8	Summary	52
7	**Communication**	**54**
7.1	Communication, Information and Language	55
7.2	The diversity of human language	56
7.3	Communication via communities of agents	60
7.4	Communicating Behaviour	63



Click on the ad to read more

Download free eBooks at bookboon.com

7.5	The Small World Phenomenon and Dijkstra's algorithm	67
7.6	Using communicating agents for searching networks	75
7.7	Entropy and Information	81
7.8	Calculating Entropy in NetLogo	82
7.9	Language Modelling	88
7.10	Entropy of a Language	90
7.11	Communicating Meaning	96
7.12	Summary	102
8	**Search**	**103**
8.1	Search Behaviour	104
8.2	Search Problems	106
8.3	Uninformed (blind) search	109
8.4	Implementing uninformed search in NetLogo	117
8.5	Search as behaviour selection	124
8.6	Informed search	127
8.7	Local search and optimisation	135
8.8	Comparing the search behaviours	139
8.9	Summary and Discussion	144

SIMPLY CLEVER

ŠKODA

Do you like cars? Would you like to be a part of a successful brand?
We will appreciate and reward both your enthusiasm and talent.
Send us your CV. You will be surprised where it can take you.

Send us your CV on
www.employerforlife.com

Click on the ad to read more

Download free eBooks at bookboon.com

9	**Knowledge**	**147**
9.1	Knowledge and Knowledge-based Systems	147
9.2	Knowledge as justified true belief	151
9.3	Different types of knowledge	153
9.4	Some approaches to Knowledge Representation and AI	156
9.5	Knowledge engineering problems	163
9.6	Knowledge without representation	164
9.7	Representing knowledge using maps	166
9.8	Representing knowledge using event maps	171
9.9	Representing knowledge using rules and logic	175
9.10	Reasoning using rules and logic	182
9.11	Knowledge and reasoning using frames	195
9.12	Knowledge and reasoning using decision trees	203
9.13	Knowledge and reasoning using semantic networks	206
9.14	Summary and Discussion	211

Click on the ad to read more

Download free eBooks at bookboon.com

10	**Intelligence**	**213**
10.1	The nature of intelligence	213
10.2	Intelligence without representation and reason	217
10.3	What AI can and can't do	218
10.4	The Need for Design Objectives for Artificial Intelligence	224
10.5	What are Good Objectives?	225
10.6	Some Design Objectives for Artificial Intelligence	225
10.7	Towards believable agents	232
10.8	Towards computers with problem solving ability	239
10.9	Summary and Discussion	246
11	**References**	**248**

Click on the ad to read more

Download free eBooks at bookboon.com

Part 2
Agent Behaviour I

Download free eBooks at bookboon.com

6 Behaviour

The frame-of-reference problem has three main aspects:

1. *Perspective issue: We have to distinguish between the perspective of the observer and the perspective of the agent itself. In particular, descriptions of behavior from an observer's perspective must not be taken as the internal mechanisms underlying the described behavior.*
2. *Behavior-versus-mechanism issue: The behavior of an agent is always the result of a system-environment interaction. It cannot be explained on the basis of internal mechanisms only.*
3. *Complexity issue: The complexity we observe in a particular behavior does not always indicate accurately the complexity of the underlying mechanisms.*

Rolf Pfeifer and Christian Scheier. 2001.
Understanding Intelligence. Page 112. The MIT Press.

Part 1 of this volume has explored agents and environments and important aspects of agent-environment interaction, including movement, and embodiment, and how it affects behaviour. Part 2 explores agent behaviour, including different types of behaviour such as communication, searching, knowing (knowledge) and intelligence.

This chapter explores the topic of agent behaviour in more depth. The chapter is organised as follows. Section 6.1 provides a definition of behaviour. Section 6.2 revisits the distinction between reactive versus cognitive agents from a behavioural perspective. Sections 6.3 to 6.5 describe useful concepts related to behaviour: emergence, self-organisation, adaptivity, evolution, the frame of reference problem, stigmergy, and swarm intelligence. Section 6.6 looks at how we can implement various types of behaviour using turtle agents in NetLogo. One particular method called boids is discussed in Section 6.7.

Download free eBooks at bookboon.com

6.1 What is behaviour?

The way an agent behaves is often used to tell them apart and to distinguish what and who they are, whether animal, human or artificial. Behaviour can also be associated with groups of agents, not just a single agent. For example, human cultural behaviour relates to behaviour that is associated with a particular nation, people or social group, and is distinct from the behaviour of an individual human being or the human body. Behaviour also has an important role to play in the survival of different species and subspecies. It has been suggested, for example, that music and art formed part of a suite of behaviours displayed by our own species that provided us with the evolutionary edge over the Neanderthals.

In the two preceding chapters, we have talked about various aspects concerning behaviours of embodied, situated agents, such as how an agent's behaviour from a design perspective can be characterised in terms of its movement it exhibits in an environment, and how agents exhibit a range of behaviours from reactive to cognitive. We have not, however, provided a more concrete definition of what behaviour is. From the perspective of designing embodied, situated agents, behaviour can b defined as follows. A particular behaviour of an embodied, situated agent is a series of actions it performs when interacting with an environment. The specific order or manner in which the actions' movements are made and the overall outcome that occurs as a result of the actions defines the type of behaviour. We can define an action as a series of movements performed by an agent in relation to a specific outcome, either by volition (for cognitive-based actions) or by instinct (for reactive-based actions).

With this definition, movement is being treated as a fundamental part of the components that characterise each type of behaviour – in other words, the actions and reactions the agent executes as it is performing the behaviour. The distinction between a movement and an action is that an action comprises one or more movements performed by an agent, and also that there is a specific outcome that occurs as a result of the action. For example, a human agent might wish to perform the action of turning a light switch on. The outcome of the action is that the light gets switched on. This action requires a series of movements to be performed such as raising the hand up to the light switch, moving a specific finger up out of the hand, then using that finger to touch the top of the switch, then applying pressure downwards until the switch moves. The distinction between an action and a particular behaviour is that a behaviour comprises one or more actions performed by an agent in a particular order or manner. For example, an agent may prefer an energy saving type of behaviour by only switching lights on when necessary (this is an example of a cognitive type of behaviour as it involves a conscious choice). Another agent may always switch on the light through habit as it enters a room (this is an example of a mostly reactive type of behaviour).

Behaviour is the way an agent acts in a given situation or set of situations. The situation is defined by the environmental conditions, its own circumstances and the knowledge the agent currently has available to it. If the agent has insufficient knowledge for a given situation, then it may choose to search for further knowledge about the situation. Behaviours can be made up of sub-behaviours. The search for further knowledge is itself a behaviour, for example, and may be a component of the original behaviour.

Download free eBooks at bookboon.com

There are also various aspects to behaviour, including the following: sensing and movement (sensory-motor co-ordination); recognition of the current situation (classification); decision-making (selection of an appropriate response); performance (execution of the response).

Behaviours range from the fully conscious (cognitive) to the unconscious (reactive), from overt (done in an open way) to covert (done in a secretive way), and from voluntary (the agent acts according to its own free will) to involuntary (done without conscious control or done against the will of the agent). The term 'behaviour' also has different meanings depending on the context (Reynolds, 1987). The above definition is applicable when the term is being used in relation to the actions of a human or animal, but it is also applicable in describing the actions of a mechanical system, or the complex actions of a chaotic system, if the agent-oriented perspective is considered (here the agents are humans, animals, mechanical systems or complex systems). However, in virtual reality and multimedia applications, the term can sometimes be used as a synonym for computer animation. In the believable agents and artificial life fields, behaviour is used "to refer to the improvisational and life-like actions of an autonomous character" (Reynolds, 1987). We also often anthropomorphically attribute human behavioural characteristics with how a computer operates when we say that a computer system or computer program is behaving in a certain way based on responses to our interaction with the system or program. Similarly, we often (usually erroneously) attribute human behavioural characteristics with animals and inanimate objects such as cars.

6.2 Reactive versus Cognitive Agents

In this section, we will further explore the important distinction between reactive and cognitive behaviour that was first highlighted in the previous chapter. Agents can be characterised by where they sit on a continuum as shown in Figure 6.1. This continuum ranges from purely reactive agents that exhibit no cognitive abilities (such as ants and termites), to agents that exhibit cognitive behaviour or have an ability to think. Table 6.1 details the differences between the two types of agents. In reality, many agents exhibit both reactive and cognitive behaviours to varying degrees, and the distinction between reactive and cognitive can be arbitrary.

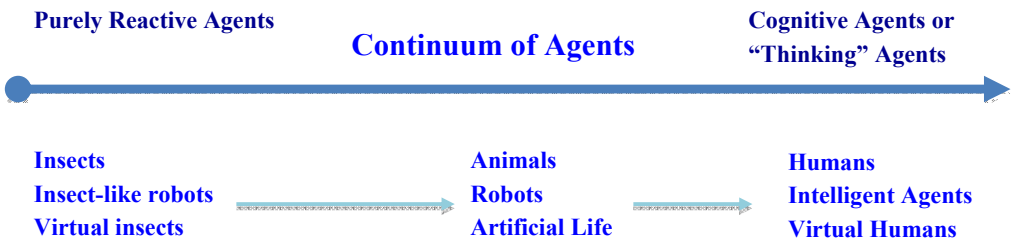


Figure 6.1 The continuum of agents, from reactive to cognitive (based on Ferber, J. 1999; page 20).

Download free eBooks at bookboon.com

Comparing the abilities of reactive agents with cognitive agents listed in Table 6.1, it is clear that reactive agents are very limited in what they can do as they do not have the ability to plan, co-ordinate between themselves or set and understand specific goals; they simply react to events when they occur. This does not preclude them from having a role to play in producing intelligent behaviour. The reactive school of thought is that it is not necessary for agents to be individually intelligent. However, they can work together collectively to solve complex problems. Their power comes from the power of the many – for example, colony based insects such as ants and termites have an ability to perform complex tasks such as finding and communicating the whereabouts of food, fighting off invaders, and building complex structures. But they do this at the population level, not at the individual level, using very rigid repetitive behaviours.

In contrast, the cognitive school of thought seeks to build agents that exhibit intelligence in some manner. In this approach, individual agents have goals, and can develop plans on how to achieve them. They use more sophisticated communication mechanisms, and can intentionally co-ordinate their activities. They also map their environment in some manner using an internal representation or knowledge base that they can refer to and update through learning mechanisms in order to help guide their decisions and actions. As a result, they are much more flexible in their behaviour compared to reactive agents.

Reactive Agents	**Cognitive Agents**
• Use simple behaviours.	• Use complex behaviours.
• Have low complexity.	• Have high complexity.
• Are not capable of foreseeing the future.	• Anticipate what is going to happen.
• Do not have goals.	• Have specific goals.
• Do not plan or co-ordinate amongst themselves.	• Make plans and can co-ordinate with each other.
• Have no representation of the environment.	• Map their environment (i.e. build internal representations of their environment).
• Do not adapt or learn.	• Exhibit learned behaviour.
• Can work together to resolve complex problems.	• Can resolve complex problems both by working together and by working individually.

Table 6.1 Reactive versus cognitive agents.

Download free eBooks at bookboon.com

In Artificial Intelligence, the behavioural approach to building intelligent systems is called Behaviour Based Artificial Intelligence (BBAI). In this approach, first proposed by Rodney Brooks, intelligence is decomposed into a set of independent semi-autonomous modules. These modules were originally conceived of as running on a separate device with their own processing threads and can be thought of as separate agents. Brooks advocated a reactive approach to AI and used finite state machines (similar to those shown in Section 6.3 and below) to implement the behaviour modules. These finite state machines have no conventional memory, and do not directly provide for higher-level cognitive functions such as learning and planning. They specify the behaviour in a reactive way, with the agent reacting directly with the environment rather than building a representation of it in some manner such as a map. The behaviour-based approach to AI has become popular in robotics, but is also finding other applications in the areas of computer animation and intelligent virtual agents, for example.

6.3 Emergence, Self-organisation, Adaptivity and Evolution

This section discusses several features of autonomous agents that are important from a behavioural perspective – emergent, self-organising, adaptive and evolving behaviour.

Click on the ad to read more

Download free eBooks at bookboon.com

A *complex system* is a system comprising many components which when they interact with each other produce activity that is greater than what is possible by the components acting individually. A multi-agent system is a complex system if the agents exhibit behaviours that are *emergent*. Emergence in a complex system is the appearance of a new higher-level property that is not a simple linear aggregate of existing properties. For example, the mass of an aeroplane is not an emergent property as it is simply the sum of the mass of the plane's individual components. On the other hand, the ability to fly is an emergent property as this property disappears when the plane's parts are disassembled. Emergent properties are also common in real life – for example, cultural behaviour in humans, food foraging behaviour in ants and mound building behaviour in termites. Emergent behaviour is the appearance of behaviour of a multi-agent system that was not previously observed and that is not the result of a simple linear combination of the agents' existing behaviour.

Some people believe that intelligence is an emergent property (see Chapter 10), the result of agent-agent and agent-environment interactions of reactive, embodied, situated agents. If this is so, then this provides an alternative path for producing intelligent behaviour – rather than building cognitive agents by explicitly programming higher cognitive abilities such as reasoning and decision-making, the alternative is to build agents with reactive abilities such as pattern recognition and learning, and this will lead to intelligent behaviour as a result. This approach, however, has yet to bear fruit as the mechanisms behind humans' pattern recognition and learning abilities is yet to be fully understood and we do not have sophisticated enough algorithms in this area for agent's to learn the way humans do, for example, such as a young child's ability to acquire language. However, the more traditional route to artificial intelligence – that of designing agents with explicit higher-level cognitive abilities – also has yet to bear fruit.

A system is said to *self-organise* when a pattern or structure in the system emerges spontaneously that was not the result of any external pressures. A multi-agent system displays self-organising behaviour as a result of applying local rules when a pattern or structure forms as a result of its interaction that was not caused by an external agent.

Zebras in front of a termite mound in Tanzania.

Download free eBooks at bookboon.com

Self-organising systems typically display emergent properties. Many natural systems exhibit self-organising behaviour. Some examples are: swarms of birds and fish, and herds of animals such as cattle, sheep, buffalo and zebras (biology); the formation and structure of planets, stars, and galaxies (from the field of astrophysics); cloud formations and cyclones (meteorology); surface structure of the earth (geophysics); chemical reactions (chemistry); autonomous movements of robots (robotics); social networks (Internet); computer and traffic networks (technology); naturally occurring fractal patterns such as ferns, snowflakes, crystalline structures, landscapes, fiords (natural world); patterns occurring on fur, butterfly wings, insect skin and blood vessels inside the body (biology); population growth (biology); the collective behaviour of insect colonies such as termites and ants (biology); mutation and selection (evolution); and competition, stock markets and financial markets (economics).

The NetLogo Models Library contains a number of models that simulate self-organisation. For example, the Flocking model mimics flocking behaviour in birds – after running the model for some time, the turtle agents will self-organise into a few flocks where the birds all head in a similar direction. This is despite the individual agents' behaviour only consisting of a few local rules (see further details in Section 6.7 below). In the Fireflies model, the turtle agents are able to synchronise their flashing using only interactions between adjacent agents; again only local rules define the individual agents' behaviour. The Termites model and the State Machine Example Model simulate the behaviour of termites. After running these models for some time, the 'wood chip' patches will end up being placed in a few piles. Three screenshots of the State Machine Example Model are shown in Figure 6.2. The leftmost image shows the environment at the start of the simulation (number of ticks = 0). It shows agents placed randomly throughout the environment, with the yellow patches representing the wood chips, and the white shapes representing the termites. The middle and rightmost images shows the environment after 5,000 and 50,000 ticks, respectively. The orange shapes represent termites that are carrying wood chips, the white shapes those that are not. The two images show the system of termite agents, wood chips and the environment progressively self-organising so that the wood chips end up in a few piles.

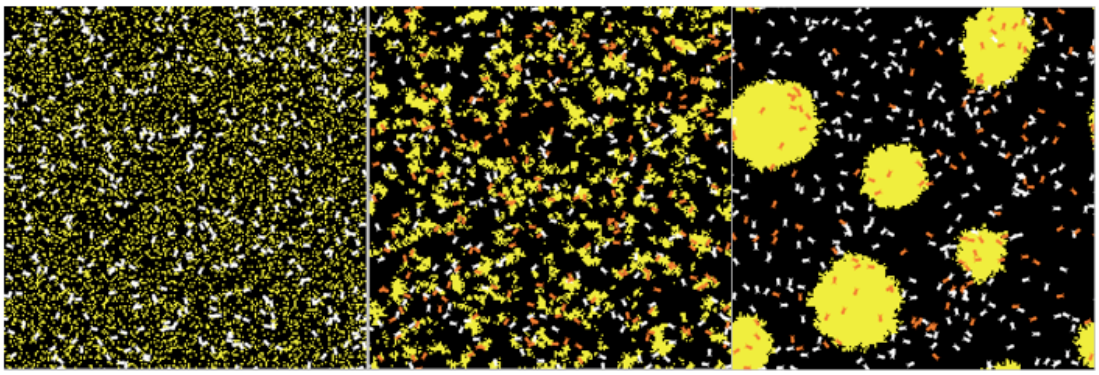


Figure 6.2 The State Machine Example Model simulates self-organising behaviour for termites.

The code for the State Machine Example Model is shown in NetLogo Code 6.1.

Download free eBooks at bookboon.com

```
turtles-own [
   task  ;;procedure name (a string) the turtle will run during this tick
   steps ;; …unless this number is greater than zero, in which
         ;; case this tick, the turtle just moves forward 1
]
to setup
   clear-all
   set-default-shape turtles "bug"
   ;; randomly distribute wood chips
   ask patches [
      if random-float 100 < density
      [ set pcolor yellow ]
   ]
   ;; randomly distribute termites
   crt number [
      set color white
      setxy random-xcor random-ycor
      set task "search-for-chip"
      set size 5 ;; easier to see
   ]
end
to go
   ask turtles
   [ ifelse steps > 0
      [ set steps steps - 1 ]
      [ run task
      wiggle ]
      fd 1 ]
   tick
end
to wiggle ;; turtle procedure
   rt random 50
   lt random 50
end

to search-for-chip ;; turtle procedure - "picks up chip" by turning orange
   if pcolor = yellow
      [ set pcolor black
        set color orange
        set steps 20
        set task "find-new-pile" ]
end
to find-new-pile ;; turtle procedure - look for yellow patches
   if pcolor = yellow
      [ set task "put-down-chip" ]
end
```


Download free eBooks at bookboon.com

```
to put-down-chip ;; turtle procedure - finds empty spot & drops chip
   if pcolor = black
      [ set pcolor yellow
      set color white
      set steps 20
      set task "get-away" ]
end
to get-away ;; turtle procedure - get out of yellow pile
      if pcolor = black
      [ set task "search-for-chip" ]
end
```

NetLogo Code 6.1 Code defining the State Machine Example model.

Click on the ad to read more

Download free eBooks at bookboon.com

The `setup` procedure randomly distributes the yellow patch agents and the termite agents throughout the environment. The `ask` command in the `go` procedure defines the behaviour of the termite agents. The approach used is to represent the behaviour as a finite state machine consisting of four states with a different action or task that the agent performs providing the transition to the next state. These tasks are: searching for a wood chip; finding a new pile; putting down a wood chip; and getting out of the pile. A simplified finite state machine for this model is depicted in Figure 6.3.

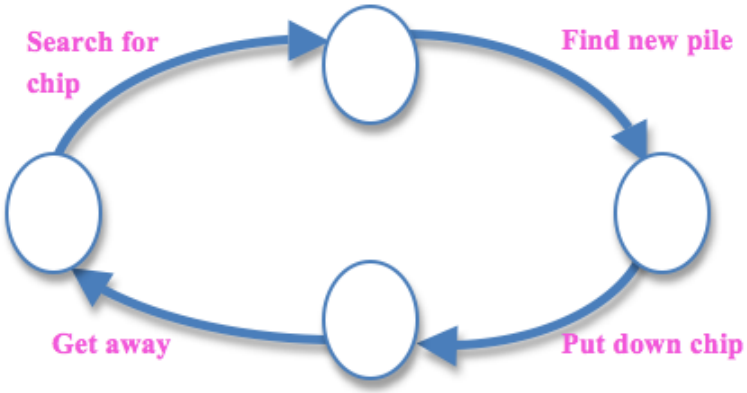
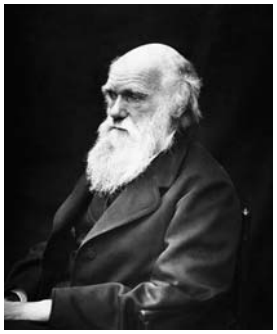


Figure 6.3 The behaviour of NetLogo Code 6.1 converted to a finite state machine.



Charles Darwin

A system in general sense is said to *evolve* if it adapts or changes over time usually from a simple to a more complex form. The term 'evolve' has different meanings in different contexts and this can cause some confusion. A more specific meaning relates the term 'evolve' to Darwin's theory of evolution – a species is said to evolve when a change occurs in the DNA of its population from one generation to the next. The change is passed down to offspring through reproduction. These changes may be small, but over many generations, the combined effects can lead to substantial changes in the organisms.

Download free eBooks at bookboon.com

In order to differentiate the different meanings of the term 'evolve', we can define adaptive and evolving behaviour separately in the following way. An agent exhibits adaptive behaviour when it has the ability to change its behaviour in some way in response to changes in the environment. If the environment changes, behaviour that is well-adapted to the previous environment may no longer be so well-adapted; for example, in Section 5.4, it was shown how some behaviour suited to solving the Hampton Court Palace maze environment is not so well suited to the Chevening House maze environment and vice versa.

Evolving behaviour, on the other hand, occurs in a population when its genetic makeup has changed from one generation to the next. Evolution in a population is driven by two major mechanisms – natural selection and genetic drift. Natural selection is a process whereby individuals with inheritable traits that are helpful for reproduction and survival in the environment will become more common in the population, whereas harmful traits will become more rare. Genetic drift is the change in the relative frequency of inheritable traits due to the role of chance in determining which individuals survive and reproduce.

Mt. Everest as seen from the Rongbuk valley, close to base camp at 5,200m.

Evolution of humans and animal species occurs over hundreds of thousands of years, and sometimes millions. To put these time scales into perspective, and to illustrate how small changes can have epoch-changing effects, we can use the example of the Himalaya Mountain Range. A fault line stretches from one end of the Himalayas to the other as it sits on the boundary between the Eurasian and Indo-Australian tectonic plates, and as a consequence it is one of the most seismically active regions in the world. Studies have shown that the Himalayas are still rising at the rate of about 1cm per year. Although a 1cm rise per year may seem negligible, if we project this far into the future, then the accumulative effect can be remarkable. After 100 years, it will have risen by only a metre; after 1000 years, 10m; after 10000 years, just 100m, still not especially significant when compared to the overall average height of the mountain range. However, after 100,000 years, it will have risen by 1 km – that is over 10% of the current height of Mt. Everest which is 8,848 metres. After a million years, the rise in height will be 10 km, which more than *doubles* the current height of Mt. Everest.

Download free eBooks at bookboon.com

A process that produces very little change from year to year, if continual, will produce dramatic changes over the course of a million years. Mt. Everest rising constantly for a million years is clearly a hypothetical situation because there are other forces at work such as erosion and tectonic plate movement. In contrast, the rise of the seas, even by as small amount as 1cm per year, can result in dramatic change in a much shorter period of time. Continental drift has also caused significant change in the world's landscape. The flight distance between Sydney, Australia and Wellington, New Zealand, for example, is 2220 km. If New Zealand has been moving apart from Australia at the rate of 1 cm per year, then this has occurred over a period of 222 million years.

No matter how well suited a particular species may be at surviving in its current environment, it will need to adapt to epoch-level changes if it is to survive for a very long time.

6.4 The Frame of Reference Problem

It is important not to attribute the wrong explanations from observations to the mechanisms behind the behaviour of an embodied agent situated within an environment. The frame of reference problem highlights the difference between the perspective of the observer and the perspective of the observed due to their different embodiment. Each real-life agent is unique with its own body and brain, with a unique set of sensing capabilities, and has a unique location within the environment (since in real-life environments no two bodies can occupy the same space at the same time). Hence, each agent has a unique perspective of its environment; therefore, the perspective of the agent doing the observing will be very different to the perspective of the agent that is being observed. The disparity in frames of reference will be most pronounced between species with vastly different embodiments, for example, between humans and insects. Often humans as observers will make the mistake of attributing human-like capabilities when describing the mechanisms behind the behaviour that is being observed. For example, magnetic termite mounds in northern Australia all face north and from a distance look like tombstones in a graveyard. In this case, it is easy to make the mistake that these were created according to some central plan, but the termite mounds are the result of many individual agents applying simple reactive behaviour.

Imagine what the world looks like to an ant making its way through long grass…

Download free eBooks at bookboon.com

Rolf Pfeifer and Christian Scheier (1999) state there are three main aspects to the frame of reference problem: the perspective issue; the behaviour-versus-mechanism issue; and the complexity issue (see quote at the beginning of this chapter). The perspective issue concerns the need to distinguish between the perspectives of the observer and the observed, and not to attribute descriptions of the mechanisms from the observer's point of view. The behaviour-versus-mechanism issue states that the behaviour of an agent is not just the result of internal mechanisms only; the agent-environment interaction also has an important role to play. The complexity issue points out that complex behaviour is not necessarily the result of complex underlying mechanisms.

Rolf Pfeifer and Christian Scheier use the thought experiment of an ant on a beach first proposed by Simon (1969) to illustrate these issues. A similar scenario is posed in Thought Experiment 6.1.

Click on the ad to read more

Download free eBooks at bookboon.com

Thought Experiment 6.1 An Ant on the Beach.

Imagine an ant returning to its nest on the edge of a beach close to a forest. It encounters obstacles along the way in the forest, such as long grass (see image), fallen leaves and branches, and then it speeds up once it reaches the beach that is near to its nest. It follows a specific trail along the beach and encounters further obstacles such as small rocks, driftwood, dried seaweed and various trash such as discarded plastic bottles and jetsam washed up from the sea. The ant seems to be following a specific path, and to be reacting to the presence of the obstacles by turning in certain directions, as if guided by a mental map it has of the terrain. Most of the ants following the original ant also travel the same way. Eventually, all the ants return to the nest, even the ones that seemed to have gotten lost along the way.

A boy walking down the beach notices the trail of ants. He decides to block their trail by building a small mound of sand in their path. The first ant that reaches the new obstacle seems to immediately recognize that there is something new in its path that wasn't there before. It repeatedly turns right, then left, as if hunting for a path around the obstacle. Other ants also arrive, and together they appear to be co-ordinating the hunt for a path around the obstacle. Eventually the ants are able to find a path around the obstacle and resume their journey back to the nest. After some further time, one particular path is chosen which is close to the shortest path back to the nest.

From an observer's point of view, the ants seem to be exhibiting intelligent behaviour. Firstly, they appear to be following a specific complex path, and seem to have the ability the recognize landmarks along the way. Secondly, they appear to have the ability to communicate information amongst themselves. For example, they quickly transmit the location of a new food source so that other ants can follow. Thirdly, they can find the shortest path between two points. And fourthly, they can cope with a changing environment.

However, it would be a mistake to attribute intelligence to the ants' behaviour. Studies have shown that the ants are just executing a small set of rules in a reactive manner. They have no ability to create a map of their environment that other ants can follow at a latter time. They are not aware of the context of their situation, such as they have come a long way, but are now close to the nest, so can speed up as a result in order to get back quicker. They cannot communicate information directly except by chemical scent laid down in the environment. They cannot discuss and execute a new plan of attack when things don't go according to plan, and there is no central co-ordinator. Contrast this with human abilities such as an orienteer using a map to locate control flags placed by someone else in the forest or beach, or a runner speeding up at the end of a long run because she knows it is nearing completion, or a hunter in a tribe returning to the camp to tell other hunters where there is more game, or the chief of the tribe telling a group of hunters to go out and search for more food.

Now further imagine that the ant has a giant body, the same size as a human's. It is most likely that the behaviour of the giant ant will be quite different to the normal sized ant as a result. Small objects that were obstacles for the ant with a normal sized body would no longer pose a problem. In fact, these would be ignored in all likelihood and the giant ant would return more directly back to the nest. Other objects that the normal sized ant would not have been aware of as being distinct, such as a tree, would now pose a different problem for the giant ant in relation to its progress through the terrain. And it may now be difficult for the giant ant to sense the chemical scent laid down on the ground. In summary, the change in the ant's body dramatically alters its perspective of its environment.

6.5 Stigmergy and Swarm Intelligence

In the Termites and Ants models described previously, we have already seen several examples of how a collection of reactive agents can perform complex tasks that are beyond the abilities of any of the agents acting singly. From our own frame of reference, these agents appear collectively to be exhibiting intelligent behaviour, although as explained in the previous section, this would be incorrect. The NetLogo models illustrate how the mechanisms behind such behaviour are very simple – just a few rules defining how the agents should interact with the environment.

Download free eBooks at bookboon.com

This section defines two important concepts related to the intelligence of a collection of agents: stigmergy, and swarm intelligence.

A collection of agents exhibit stigmergy when they make use of the environment in some manner, and as a result, are able to co-ordinate their activities to produce complex structures through self-organisation. The key idea behind stigmergy is that the environment can have an important influence on the behaviour of an agent and vice versa. In other words, the influence between the environment and agent is bi-directional. In real-life, stigmergy occurs amongst social insects such as termites, ants, bees and wasps. As we have seen with the Termites and Ants models, stigmergy can occur between very simple reactive agents that only have the ability to respond in a localised way to local information. These agents lack intelligence and mutual awareness in the traditional sense, they do not use memory, and do not have the ability to plan, control or directly communicate with each other. Yet they have the ability to perform higher-level tasks as a result of their combined activities.

Stigmergy is not restricted to natural life examples – the Internet is one obvious example. Many computer systems also make use of stigmergy – for example, the ant colony optimization algorithm is a method for finding optimal paths as solutions to problems. Some computer systems use shared data structures that are managed by a distributed community of clients that supports emergent organization. One example is the blackboard architecture as used in AI systems first developed in the 1980s. A blackboard makes use of communication via a shared memory that can be written to independently by an agent then examined by other agents much like a real-life blackboard can in a lecture room. Blackboards are now being used in first-person shooter video games, and as a means of communication between agents in a computer network (the latter is discussed in more detail in section 7.9).

A colony of ants, and a swarm of bees – both use stigmergic local knowledge to co-ordinate their activities.

Download free eBooks at bookboon.com

A collection of agents exhibit swarm intelligence when they make use of stigmergic local knowledge to co-ordinate their activities and to produce complex structures through self-organisation. The mechanisms behind swarm intelligence exhibited by social insects are robust since there is no centralised control. They are also very effective – as demonstrated by the size of worldwide populations and the mirroring of solutions across different species. A few numbers emphasize this point. Scientists estimate that there are approximately 9000 species of ants and one quadrillion (10^{15}) ants living in the world today (Elert, 2009). Also, it has been estimated that colonies of harvester ants, for example, have a similar number of neurons as a human brain. Humans also make use of swarm intelligence in many ways. The online encyclopaedia, Wikipedia, is just one example which results from the collective intelligence of humans acting individually with minimal centralised control. Social networking via online websites is another. Both of these make use of stigmergic local information laid down in the cloud.

6.6 Implementing behaviour of Turtle Agents in NetLogo

In NetLogo, the behaviour of an agent is specified explicitly by the ask command. This defines the series of commands that each agent or agentset executes, in other words, the procedure that the agent is to perform. A procedure in a computer program is a specific series of commands that are executed in a precise manner in order to produce a desired outcome. However, we have to be careful to make a distinction between the actual behaviour of the agent and the mechanics of the NetLogo procedure that is used to define the behaviour. The purpose of much of the procedural commands is to manipulate internal variables including global variables and the agent's own variables. The latter reflects the state of the agent and can be represented as points in an *n*-dimensional space. However, this state is insufficient to describe the behaviour of the agent. Its behaviour is represented by the actions the agent performs which results in some change to its own state, to the state of other agents or to the state of the environment. The type of change that occurs represents the outcome of the behaviour.

Some example behaviours that we have already seen exhibited by agents in Netlogo models are: the food foraging behaviour of ant agents in the Ants model which results in the food being returned efficiently to the nest as an outcome; the nest building behaviour of termite agents in the Termites and State Machine Example models which results in the wood chips being placed in piles as an outcome; and the wall following behaviour of the turtle agents in the Wall Following Example model which results in the turtle agents all following walls in a particular direction as an outcome.

Download free eBooks at bookboon.com

The Models Library in NetLogo comes with many more examples where agents exhibit very different behaviours. In most of these models, the underlying mechanisms are due to the mechanical application of a few local rules that define the behaviour. For example, the Fireflies model simulates the ability of a population of fireflies using only local interactions to synchronise their flashing as an outcome. The Heatbugs model demonstrates how several kinds of emergent behaviour can arise as an outcome from agents applying simple rules in order to maintain an optimum temperature around themselves. The Flocking model mimics the behaviour of the flocking of birds, which is also similar to schooling behaviour of fish and the herding behaviour of cattle and sheep. This outcome is achieved without a leader, with each agent executing the same set of rules. The compactness of the NetLogo code in these models reinforces that complexity of behaviour does not necessarily correlate with the complexity of the underlying mechanisms.

Behaviour can be specified by various alternatives, such as by NetLogo procedures and commands, and by finite state automata as outlined in Section 6.3. The latter is an abstract model of behaviour with a limited internal memory. In this format, behaviour can be considered as the result of an agent moving from one state to another state – or points in an n-dimensional space – as it can be represented as a directed graph with states, transitions and actions. In order to make the link between a procedure implemented in a programming language such as NetLogo and finite state automata (and therefore re-emphasize the analogy between behaviour and movement of an agent situated in an environment), the wall following behaviour of NetLogo Code 5.7, repeated below, has been converted to an equivalent finite state machine in Figure 6.4.

Click on the ad to read more

Download free eBooks at bookboon.com

```
to behaviour-wall-following
; classic 'hand-on-the-wall' behaviour
   if not wall? (90 * direction) 1 and wall? (135 * direction) (sqrt 2)
      [ rt 90 * direction ]

;; wall straight ahead: turn left if necessary (sometimes more than once)
while [wall? 0 1] [ lt 90 * direction]

   ;; move forward
   fd 1
end
```

NetLogo Code 6.2 The wall following behaviour extracted from NetLogo Code 5.7.

The code has been converted to a finite state machine by organising the states into the 'sense – think – act' mode of operation as outlined in Section 5.5. Note that we are not restricted to doing the conversion in this particular way – we are free to organise the states and transitions in whatever manner we wish. In this example, the states and transitions as shown in the figure have been organised to reflect the type of action (sensing, thinking or acting) the agent is about to perform during the next transition out of the state. Also, regardless of the path chosen, the order that the states are traversed is always a sensing state followed by a thinking state then an acting state. This is then followed by another sensing state and so on. For example, the agent's behaviour starts by a sensing state (labelled Sensing State 1) on the left middle of the figure. There is only one transition out of this state, and the particular sense being used is vision as the action being performed is to look for a wall on the preferred side (that is, the right side if following right hand walls, and the left side if following left hand walls). The agent then moves onto a thinking state (Thinking State 1) that considers the information it has received from what it has just sensed. The thinking action the agent performs is to note whether there is a wall nearby or not. If there wasn't, then the agent moves to an acting state (Acting State 1) that consists of performing the action of turning 90° in the direction of the preferred side. If there was a wall, then no action is performed (Acting State 2). Note that doing nothing is considered an action, as it is a movement of zero length. The agent will then move to a new sensing state (Sensing State 2) that involves the sensing action of looking for a wall ahead. It will repeatedly loop through the acting state (Acting State 3) of turning 90° in the opposite direction to the preferred side and back to Sensing State 2 until there is not a wall ahead. Then it will move to the acting state (Acting State 4) of moving forward 1 step and back to the start.

Download free eBooks at bookboon.com

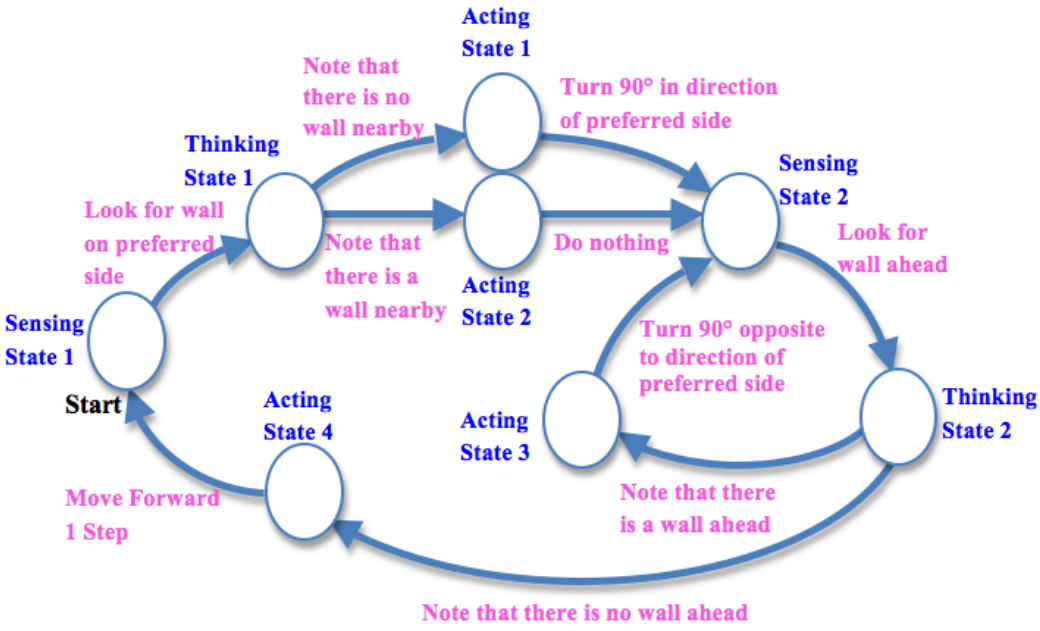


Figure 6.4 The wall following behaviour of NetLogo Code 6.2 converted to a finite state machine.

As pointed out in Section 5.5, the 'Sense – Think – Act' method of operation has limitations when applied to modelling real-life intelligent or cognitive behaviour, and an alternative approach embracing embodied, situated cognition was suggested. However, a question remains concerning how to implement such an approach since it effectively entails sensing, thinking and acting all occurring at the same time i.e. concurrently, rather than sequentially. Two NetLogo models have been developed to illustrate one way this can be simulated. The first model (called Wall Following Example 2) is a modification of the Wall Following Example model described in the previous chapter. The modified interface provides a chooser that allows the user to select the standard wall following behaviour or a modified variant. The modified code is shown in NetLogo Code 6.3.

Download free eBooks at bookboon.com

```
turtles-own
[direction ;; 1 follows right-hand wall, -1 follows left-hand wall
way-is-clear? ;; reporter - true if no wall ahead
checked-following-wall?] ;; reporter - true if checked following wall

to go
   if-else (behaviour = "Standard")
   [ ask turtles [ walk-standard ]]
   [ ask-concurrent turtles
      [ walk-modified shuffle [1 2 3]]
   ]
   tick
end

to walk-standard ;; standard turtle walk behaviour
   ;; turn right if necessary
   if not wall? (90 * direction) and wall? (135 * direction)
   [ rt 90 * direction ]
   ;; turn left if necessary (sometimes more than once)
   while [wall? 0] [ lt 90 * direction ]
    ;; move forward
   fd 1
end

to walk-modified [order] ;; modified turtle walk behaviour
   ifelse (choose-sub-behaviours = "Choose-all-in-random-order")
   [
      foreach order
      [ if (? = 1) [ walk-modified-1 ]
      if (? = 2) [ walk-modified-2 ]
      if (? = 3) [ walk-modified-3 ]]
   ]
   [
      let ord one-of order
      if (ord = 1) [ walk-modified-1 ]
      if (ord = 2) [ walk-modified-2 ]
      if (ord = 3) [ walk-modified-3 ]
   ]
end

to walk-modified-1 ;; modified turtle walk sub-behaviour 1
   ;; turn right if necessary
   if not wall? (90 * direction) and wall? (135 * direction)
   [ rt 90 * direction ]
   set checked-following-wall? true
end
to walk-modified-2 ;; modified turtle walk sub-behaviour 2
   ;; turn left if necessary (sometimes more than once)
   ifelse (wall? 0)
   [ lt 90 * direction
      set way-is-clear? false ]
   [ set way-is-clear? true ]
end
```


Download free eBooks at bookboon.com

```
to walk-modified-3 ;; modified turtle walk sub-behaviour 3
   ;; move forward
   if way-is-clear? and checked-following-wall?
   [ fd 1
      set way-is-clear? false
      set checked-following-wall? false ]
end
```

NetLogo Code 6.3 Code defining the modified wall following behaviour in the Wall Following Example 2 model.

In order to simulate the concurrent nature of the modified behaviour, the original wall following behaviour has been split into three sub-behaviours – these are specified by the `walk-modified-1`, `walk-modified-2` and `walk-modified-3` procedures in the above code. The first procedure checks whether the agent is still following a wall, and turns to the preferred side if necessary. It then sets an agent variable, **`checked-following-wall?`** to `true` to indicate it has done this. The second procedure checks whether there is a wall ahead, turns in the opposite direction to the preferred side if there is, and then sets the new agent variable `way-is-clear?` to indicate whether there is a wall ahead or not. The third procedure moves forward 1 step but only if both the way is clear ahead and the check for wall following has been done.

American online
LIGS University
is currently enrolling in the
Interactive Online BBA, MBA, MSc,
DBA and PhD programs:

- enroll by September 30th, 2014 and
- save up to 16% on the tuition!
- pay in 10 installments / 2 years
- Interactive Online education
- visit www.ligsuniversity.com to find out more!

Note: LIGS University is not accredited by any nationally recognized accrediting agency listed by the US Secretary of Education.
More info here.

Click on the ad to read more

Download free eBooks at bookboon.com

Essentially the overall behaviour is the same as before since all we have done is to split the original behaviour into three sub-behaviours – in other words, just doing this by itself does not achieve anything new. The reason for doing this is to allow us to execute the sub-behaviours in a non-sequential manner, independently of each other, in order to simulate 'sensing & thinking & acting' behaviour where '&' indicates each is done concurrently, in no particular order. This can be done in NetLogo using the `ask-concurrent` command as shown in the `go` procedure in the code. This ensures that each agent takes turns executing the `walk-modified` procedure's commands. The main difference compared to the standard behaviour is evident in this procedure. The interface to the model provides another chooser that allows the user to set a `choose-sub-behaviours` variable that controls how the sub-behaviours are executed. If this variable is set to '`Choose-all-in-random order`', then all the three sub-behaviours will be executed as with the standard behaviour, but this time in a random order; otherwise, the variable is set to '`Choose-one-at-random`', and only a single sub-behaviour is chosen.

Clearly the way the modified behaviour is executed is now discernibly different to the standard behaviour – although the former executes the same sub-behaviours of the latter, this is either done in no particular order, or only one out of three sub-behaviours is chosen each tick. And yet when running the model, the same overall results are achieved regardless of which variant of the model is chosen – each agent successfully manages to follow the walls that they find in the environment. There are minor variations between each variant, such as repeatedly going back and forth down short cul-de-sacs for the modified variants. The ability of the modified variants, however, to achieve a similar result as the original is interesting since the modified method is both effective and robust – regardless of when, and in what order the sub-behaviours are executed, the overall result is still the same.

A second NetLogo model, the Wall Following Events model, has been created to conceptualise and visualise the modified behaviour. This model considers that an agent simultaneously recognizes and processes multiple streams of 'events' that reflect what is happening to itself and in the environment (in a manner similar to that adopted in Event Stream Processing (ESP) (Luckham, 1988). These events occur in any order and have different types but are treated as being equivalent to each other in terms of how they are processed. Behaviour is defined by linking together a series of events into a forest of trees (one or more acyclic directed graphs) as shown in Figure 6.5. The trees link together series of events (represented as nodes in the graph) that must occur in conjunction with each other. If a particular event is not recorded on the tree, then that event is not recognized by the agent (i.e. it is ignored and has no effect on the agent's behaviour). The processing of the events is done in a reactive manner – that is, a particular path in the tree is traversed by successively matching the events that are currently happening to the agent against the outgoing transitions from each node. If there are no outgoing transitions or none match, then the path is a dead end, at which point the traversal will stop. This is done simultaneously for every event; in other words, there are multiple starting points and therefore simultaneous activations throughout the forest network.

Download free eBooks at bookboon.com

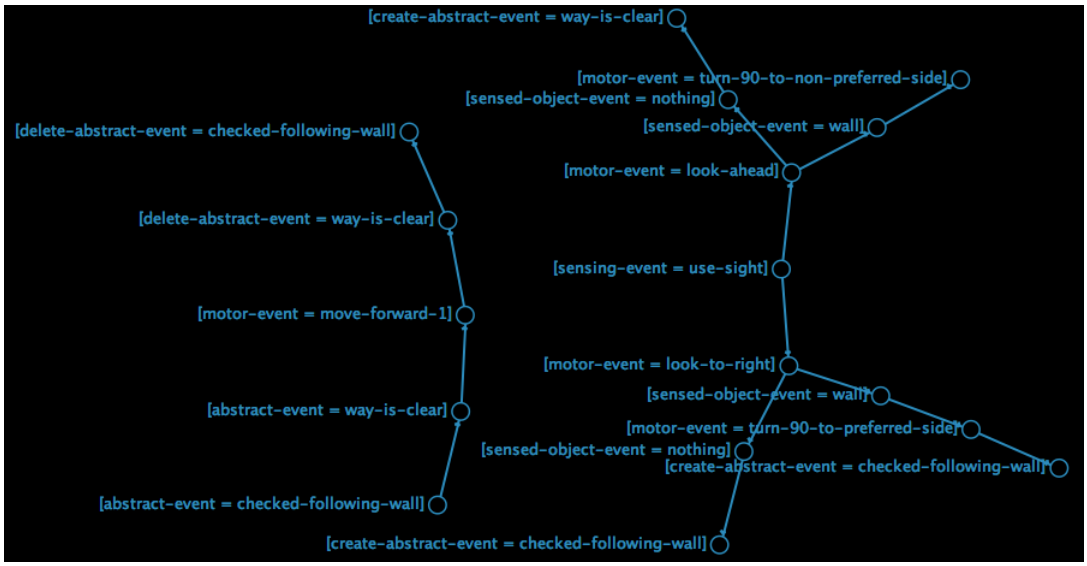


Figure 6.5 Screenshot of the Wall Following Events model defining the modified wall following behaviour.

In the figure, the event trees have been defined in order to represent the modified wall following behaviour defined above. Each node in the graph represents an event that is labelled by a stream identifier, separated by an "=", followed by an event identifier. For example, the node labelled `[motor-event = move-forward-1]` identifies the motor event of moving forward 1 step. For this model of the behaviour, there are four types of events – sensing events, where the agent begins actively sensing on a specific sensory input stream (such as sight as in the figure); motor events, where the agent is performing some motion or action; sensed-object-events, which occur when a particular object is recognised by the agent; and abstract events, which are abstract situations that are the result of one or more sensory, motor and abstract events, and which can also be created or deleted by the agent from its internal memory (which records which abstract events are currently active). If a particular abstract event is found in memory, then it can be used for subsequent matching by the agent along a tree path.

For example, the node labelled `[sensing event = use-sight]` towards the middle right of the figure represents an event where the agent is using the sense of sight. Many events can occur on this sensory input channel, but only two events in particular are relevant for defining the wall following behaviour – these are both motor events, one being the action of looking ahead, and the other being the action of looking to the right. Then depending on which path is followed, different sensed-object events are encountered in the tree, either that a wall object is sensed, or nothing is sensed. These paths continue until either a final motor event is performed (such as turning 90° to the non-preferred side at the top right of the figure) or an abstract event is created (such as whether the wall is being followed has been checked at the bottom of the figure).

Download free eBooks at bookboon.com

Note that unlike the Sense – Think – Act model depicted in Figure 6.3, this model of behaviour is not restricted to a particular order of events. Any type of event can 'follow' another, and two of the same type are also possible – for example in the path that starts on the left of the figure there are two abstract events after one another. Also note that use of the word 'follow' is misleading in this context. Although it adequately describes that one link comes after another on a particular path in the tree model, the event may in fact occur simultaneously, and the order as specified by the tree path is arbitrary and just describes the order that the agent will recognize the presence of multiply occurring events. For example, there is no reason why the opposite order cannot also be present in the tree; or an alternative order that will lead to the same behaviour (e.g. swapping the two abstract events at the bottom of the left hand path in the figure will have no effect on the agent's resultant behaviour).

The code used to create the screenshot is shown in NetLogo Code 6.4 below.

Click on the ad to read more

Download free eBooks at bookboon.com

```
breed [states state]
directed-link-breed [paths path]

states-own
[ depth ;; depth in the tree
   stream ;; the name of the stream of sensory or motor events
   event ;; the sensory or motor event
]

globals
[ root-colour node-colour link-colour ]
;; defines how the event tree gets visualised

to setup
     clear-all ;; clear everything

     set-default-shape states "circle 2"
     set root-colour sky
     set node-colour sky
     set link-colour sky

     add-events (list ["sensing-event" "use-sight"]
          (list "motor-event" "look-to-right")
          (list "sensed-object-event" "wall")
          (list "motor-event" "turn-90-to-preferred-side")
          (list "create-abstract-event" "checked-following-wall"))
     add-events (list ["sensing-event" "use-sight"]
          (list "motor-event" "look-to-right")
          (list "sensed-object-event" "nothing")
          (list "create-abstract-event" "checked-following-wall"))
     add-events (list ["sensing-event" "use-sight"]
          (list "motor-event" "look-ahead")
          (list "sensed-object-event" "wall")
          (list "motor-event" "turn-90-to-non-preferred-side"))
     add-events (list ["sensing-event" "use-sight"]
          (list "motor-event" "look-ahead")
          (list "sensed-object-event" "nothing")
          (list "create-abstract-event" "way-is-clear"))
     add-events (list ["abstract-event" "checked-following-wall"]
          (list "abstract-event" "way-is-clear")
          (list "motor-event" "move-forward-1")
          (list "delete-abstract-event" "way-is-clear")
          (list "delete-abstract-event" "checked-following-wall"))
reset-layout
end
to reset-layout
   repeat 500
   [ layout-spring states paths spring-constant spring-length
   repulsion-constant ]

    ;; leave space around the edges
    ask states [ setxy 0.95 * xcor 0.95 * ycor ]
end
```


Download free eBooks at bookboon.com

```
to change-layout
   reset-layout
   display
end

to set-state-label
;; sets the label for the state
   set label (word "[" stream " = " event "] ")
end

to add-events [list-of-events]
;; add events in the list-of-events list to the events tree.
;; each item of the list-of-events list must consist of a two itemed list.
;; e.g. [[hue 0.9] [brightness 0.8]]

   let this-depth 0
   let this-stream ""
   let this-event ""
   let this-state nobody
   let next-state nobody
   let these-states states
   let matching-states []
   let matched-all-so-far true

    foreach list-of-events
    [ set this-stream first ?
      set this-event last ?

       ;; check to see if state already exists
       set matching-states these-states with
       [stream = this-stream and event = this-event]
       ifelse (matched-all-so-far = true) and (count matching-states > 0)
      [
              set next-state one-of matching-states
              ask next-state [ set-state-label ]
              set these-states [out-path-neighbors] of next-state ]
      [ ;; state does not exist - create it
          set matched-all-so-far false
          create-states 1
       [
           set size 8
           set depth this-depth
           set stream this-stream
           set event this-event
           set-state-label
           ifelse (depth = 0)
              [ set label-color root-colour ]
              [ set label-color node-colour ]
```


Download free eBooks at bookboon.com

```
ifelse (depth = 0)
[ set color root-colour ]
[ set color node-colour ]
set next-state self
]
]
if (this-state != nobody)
[ ask this-state
[ create-path-to next-state [ set color link-colour ]]]

 ;; go down the tree
 set this-state next-state
 set this-depth this-depth + 1
 ]
 ask links [ set thickness 1.3 ]
end
```

NetLogo Code 6.4 Code for the Wall Following Events model used to produce the screenshot in Figure 6.5.

Maastricht University Leading in Learning!

Join the best at the Maastricht University School of Business and Economics!

Top master's programmes

- 33rd place Financial Times worldwide ranking: MSc International Business
- 1st place: MSc International Business
- 1st place: MSc Financial Economics
- 2nd place: MSc Management of Learning
- 2nd place: MSc Economics
- 2nd place: MSc Econometrics and Operations Research
- 2nd place: MSc Global Supply Chain Management and Change

Sources: Keuzegids Master ranking 2013; Elsevier 'Beste Studies' ranking 2012; Financial Times Global Masters in Management ranking 2012

Maastricht University is the best specialist university in the Netherlands (Elsevier)

Visit us and find out why we are the best!
Master's Open Day: 22 February 2014

www.mastersopenday.nl

Click on the ad to read more

Download free eBooks at bookboon.com

The code first defines two breeds, states and paths, which represent the transitions between states. Each state agent has three variables associated with it – `depth`, which is the distance from the root state for the tree; `stream`, which identifies the name of the specific type of `event` it is; and event, which is the name of the event. The event type is called a 'stream' as we are using an analogy of the appearance of the events as being similar to the flow of objects down a stream. Many events can 'flow' past, some appear simultaneously, but there is also a specific order for the arrival of the events in that if we choose to ignore a particular event, it is lost – we need to deal with it in some manner.

The `setup` procedure initialises the event trees by calling the `add-events` procedure for each path. This procedure takes a single parameter as input, which is a list of events, specified as pairs of stream names and event names. For example, for the first `add-events` call, the list contains five events: the first is a `use-sight` event on the `sensing-event` stream; the second is a `look-to-right` event on the `motor-event` stream; and so on. A directed path containing all the events in the event list is added to the event trees. If the first event in the list does not occur at the root of any existing tree, then the root of a new tree is created, and a non-branching path from the root is added to include the remaining events in the list. Otherwise, the first events on the list are matched against existing path, with new states added at the end when the events no longer match.

The `add-events` procedure will also be used in The Language Modelling model discussed in Section 7.5, and in visualising the different methods of knowledge representation discussed in Chapter 9.

6.7 Boids

In 1986, Craig Reynolds devised a distributed model for simulating animal behaviour that involves co-ordinated motion such as flocking for birds, schooling for fish and herding for mammals. Reynolds observed the flocking behaviour of blackbirds, and wondered whether it would be possible to get virtual creatures to flock in the same way in a computer simulation in real-time. His hypothesis was that there were simple rules responsible for this behaviour.

The model he devised uses virtual agents called boids that have a limited form of embodiment similar to that used by the agents in the Vision Cone model described in Section 5.3. The behaviour of the boids is divided into three layers – action selection, steering and locomotion – as shown in Figure 6.6. The highest layer concerns action selection that controls behaviours such as strategy, goal setting, and planning. These are made up from steering behaviours at the next level that relate to more basic path determination tasks such as path following, and seeking and fleeing. These in turn are made up of locomotion behaviours related to the movement, animation and articulation of the virtual creatures.

Download free eBooks at bookboon.com

To describe his model, Reynolds (1999) uses the analogy of cowboys tending a herd of cattle out on the range when a cow wanders away from the herd. The trail boss plays the role of action selection – he tells a cowboy to bring the stray back to the herd. The cowboy plays the role of steering, decomposing the goal into a series of sub-goals that relate to individual steering behaviours carried out by the cowboy-and-horse team. The cowboy steers his horse by control signals such as vocal commands and the use of the spurs and reins that result in the team moving faster or slower or turning left or right. The horse performs the locomotion that is the result of a complex interaction between the horse's visual perceptions, the movements of its muscles and joints and its sense of balance.

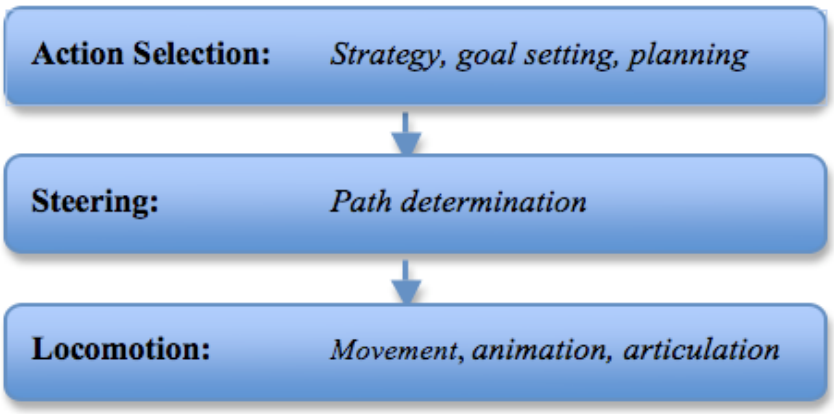


Figure 6.6 The hierarchy of motion behaviours used for the Boids model (Reynolds, 1987).

Note that the layers chosen by Reynolds are arbitrary and more of a design issue reflecting the nature of the modelling problem. Reynolds himself points out that alternative structures are possible and the one chosen for modelling simple flocking creatures would not be appropriate for a different problem such as designing a conversational agent or chatbot.

The flocking behaviour of birds is similar to the schooling behaviour of fish and herding behaviour of mammals such as antelope, zebras, bison, cattle and sheep.

Download free eBooks at bookboon.com

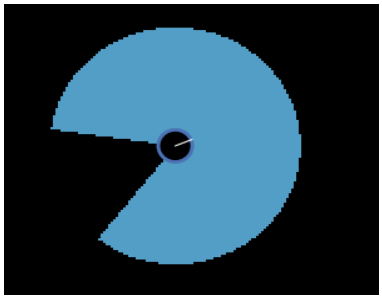


Figure 6.7 A boid in NetLogo with an angle of 300°.

Just as for real-life creatures, what the boids can see at any one point in time is determined by the direction they are facing and the extent of their peripheral vision as defined by a cone with a specific angle and distance. The cone angle determines how large a 'blind' spot they have – i.e. the part that is outside their range of vision directly behind their head opposite to the direction they are facing. If the angle of the cone is 360°, then they will be able to see all around them; if less than that, then the size of the blind spot is the difference between the cone angle and 360°.

Click on the ad to read more

Download free eBooks at bookboon.com

A boid can be easily implemented in NetLogo using the `in-cone` command as for the Vision Cone model. Figure 6.7 is a screenshot of a boid implemented in NetLogo (see the Obstacle Avoidance 1 model, for example). The image shows the vision cone coloured sky blue with an angle of 300° (the size of the blind spot is therefore 60°). The turtle is drawn using the "`directional-circle`" shape at the centre of the image and coloured blue, with the white radius line pointing in the same direction as the current heading of the turtle. The width of the cone is dependent on the length parameter passed to the `in-cone` command and the patch size for the environment.

Reynolds devised a number of steering behaviours as summarised in Table 6.2.

Steering Behaviour	**Description**
For individuals and pairs:	
Seek and Flee	A steering force is applied to the boid in relation to a specific target (towards the target for seek, and away from the target for flee).
Pursue and Evade	This is based on the underlying seeking and fleeing behaviours. Another boid becomes the moving target.
Wander	This is a form of random walking where the steering direction of the boid during one tick of the simulation is related to the steering direction used during the previous tick.
Arrival	The boid reduces its speed in relation to the distance it is from a target.
Obstacle Avoidance	The boid tries to avoid obstacles while trying to maintain a maximum speed by applying lateral and braking steering forces.
Containment	This is a generalised form of obstacle avoidance. The boid seeks to remain contained within the surface of an arbitrary shape.
Wall Following	The boid maintains close contact with a wall.
Path Following	The boid closely follows a path. The solution Reynolds devised was corrective steering by applying the seek behaviour for a moving target point further down the path.
Flow Field Following	The boid steers so that its motion is aligned to a tangent to a flow field (also called a force field or vector field).
Combined behaviours and groups:	
Crowd Path Following	The boids combine path following behaviour with a separation behaviour that tries to keep the boids from clumping together.
Leader Following	This combines separation and arrival steering behaviours to simulate boids trying to follow a leader.
Unaligned Collision Avoidance	The boids try to steer a collision-free path while moving through a crowd. The boid determines the nearest of any potential collisions, and will steer laterally to avoid it.
Queuing (at a doorway)	This simulates boids slowing down and queuing to get through a doorway. Approaching the door, each boid applies a braking steering behaviour when other boids are just in front of itself and moving slower. This behaviour combines seeking (for the door), avoidance (for the walls either side of door) and separation.
Flocking	Groups of boids combine separation, alignment and cohesion steering behaviours and flocking emerges as a result.

Table 6.2 Steering Behaviours devised by Craig Reynolds (1999).

Download free eBooks at bookboon.com

We will now see how some of these behaviours can be implemented in NetLogo. Note that, as with all implementations, there are various ways of producing each of these behaviours. For example, we have already seen wall following behaviour demonstrated by the Wall Following Example model described in the previous chapter, and by the Wall Following Example 2 model described in this chapter. Although the behaviour is not exactly the same for both models, the outcome is effectively the same. Both models have agents that use the vision cone method of embodiment of Figure 6.7 that is at the heart of the boids behavioural model.

Two models have been developed to demonstrate obstacle avoidance. Some screenshots of the first model, called Obstacle Avoidance 1, are shown in Figure 6.8. They show a single boid moving around an environment trying to avoid the white rows of obstacles – an analogy would be a moth trying to avoid bumping into walls as it flies around. The extent of the boids vision is shown by the sky coloured halo surrounding the boid – it has been set at length 8 in the model with an angle of 300°. The image on the left shows the boid just after the `setup` button in the interface has been pressed heading towards the rows of obstacles. After a few ticks, the edge of the boid's vision cone bumps into the tip of the middle north-east pointing diagonal obstacle row (depicted by the change in the colour of the obstacle at the tip from white to red), then it turns around to its left approximately 80° and heads towards the outer diagonal. Its vision cone then hits near the tip of this diagonal as well, then finally the boid turns again and heads away from the obstacles in a north east facing direction as shown in the second image on the right.

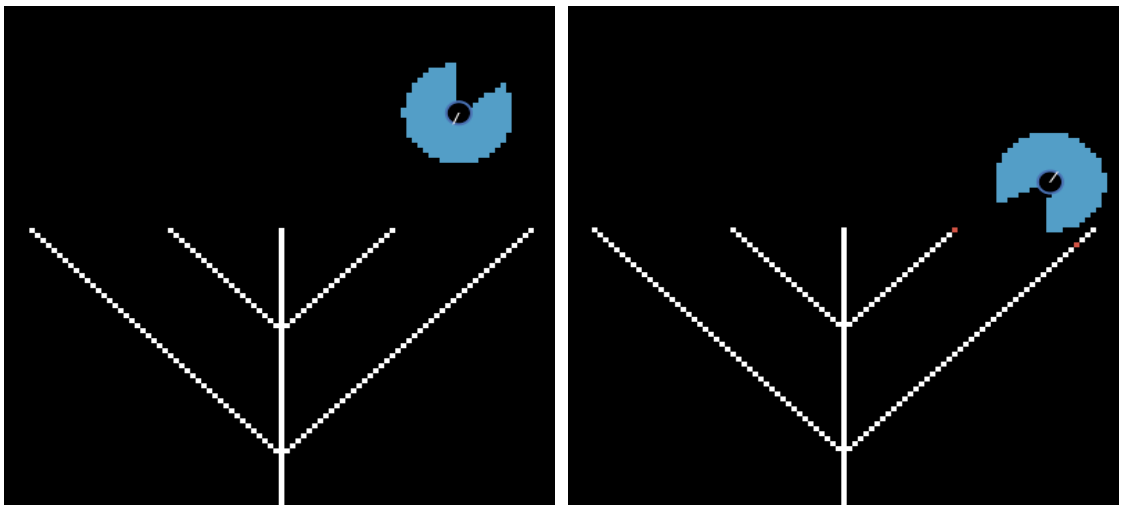


Figure 6.8 Screenshots of the Obstacle Avoidance 1 model.

The code for this is shown in NetLogo Code 6.5.

Download free eBooks at bookboon.com

```
breed [wanderers wanderer] ; name of the breed of boids
to setup
   clear-all
   set-default-shape wanderers "directional-circle"; sets shapes for the boid

   ; create colour, size and random location of single wanderer
   create-wanderers 1 [default blue ]
   draw-obstacles
end

to default [colour] ; creates default settings for boid
   print "Got here"
   set color colour              ; sets colour using passed parameter
   setxy random-xcor random-ycor ; sets an initial random position
   set size 5                    ; default turtle size
end

to draw-obstacles
ask patches with    [pxcor = 0 and pycor <= 15 or
                    abs pxcor = (pycor + 40) and pycor < 40 or
                    abs pxcor = (pycor + 15) and pycor < 6]
[ set pcolor white ]
end

to make-obstacle
   if mouse-down?
   [ ask patches
      [ if ((abs (pxcor - mouse-xcor)) < 1) and
           ((abs (pycor - mouse-ycor)) < 1)
        [ set pcolor white ]
      ]
   ]
end
to go
   ask wanderers ; wanderers instructions
   [
      rt random-float rate-of-random-turn
      lt (rate-of-random-turn / 2)
      ; randomly turns by up to left or right as defined by the
      ; random-rate-of-turn variable in the interface
      fd boid-speed
      avoid-patches
   ]
end
to avoid-patches
   ask patches with [pcolor = sky]
   [ set pcolor black ]
   ask patches in-cone radius-length radius-angle
   [ if pcolor = black
      [ set pcolor sky ]
      if pcolor = white
      [ set pcolor red ]]
```


Download free eBooks at bookboon.com

```
if count patches in-cone radius-length radius-angle with
   [pcolor = white or pcolor = red] > 0
[
   ask wanderer 0
   [
      bk boid-speed
      lt 90
   ]
 ]
end
```

NetLogo Code 6.5 The code for the Obstacle Avoidance 1 model shown in Figure 6.8.

The `setup` procedure places the boid at a random location in the environment, and calls the `draw-obstacles` procedure to draw the white obstacles in the bottom half of the environment. The `ask wanderers` command in the `go` procedure defines the behaviour of the boid. The boid will do a right and left turn of a random amount, then move forward a certain amount as specified by the variable `boid-speed` defined in the interface.

Click on the ad to read more

Download free eBooks at bookboon.com

Then the boid calls the `avoid-patches` procedure to perform the collision avoidance. In this procedure, first the sky coloured halo surrounding the boid is erased by setting sky coloured patches to black. Next, the vision halo is redrawn around the boid based on its current location – the rapid erasure followed by redrawing causes the boid to flicker much like a butterfly rapidly flapping its wings. The boid then performs the collision avoidance by backing away a distance equal to `boid-speed`, and does a left turn. The last part of the procedure sets the patches that have been collided with to red.

The Obstacle Avoidance 2 model illustrates obstacle avoidance in the same environment as the Look Ahead Model from the Models Library (discussed in Section 8.3). The screenshot of the model is shown in Figure 6.9. The boid can be seen towards the middle left of the environment. The path the boid has taken is plotted in red. This clearly shows that the boid has been successful in avoiding collisions with the walls and obstacles. The length of the boid's vision cone was set at 1, and angle 300°. The code is similar to the Obstacle Avoidance 1 model.

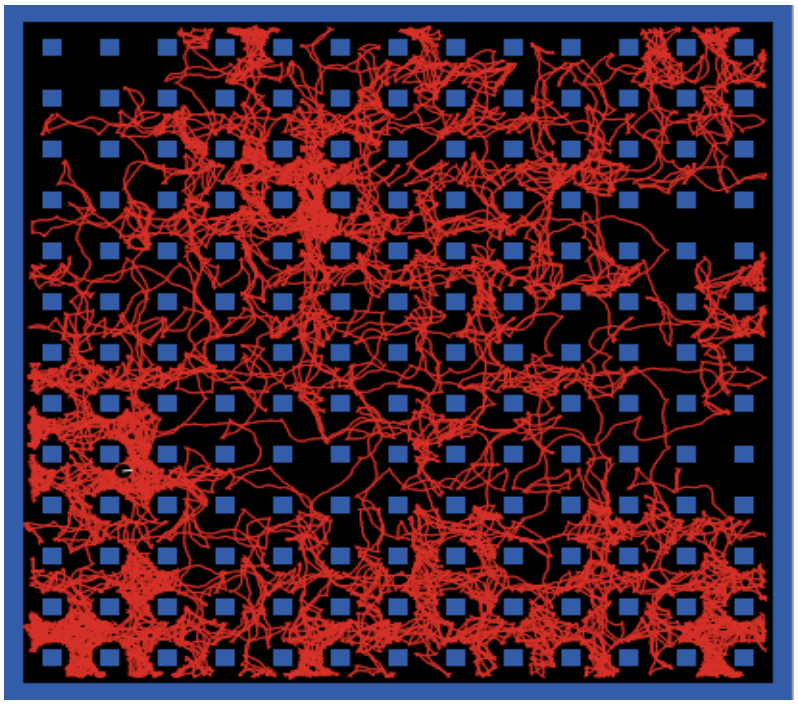


Figure 6.9 Screenshot of the Obstacle Avoidance 2 model.

Download free eBooks at bookboon.com

The Follow and Avoid model implements seeking and fleeing behaviours. Three screenshots of the model are shown in Figure 6.10. The left image shows when there is one wanderer agent coloured red, and 100 follower agents coloured green. It shows most of the agents pointing towards the wanderer agent. During the simulation, the follower agents all actively pursue the wanderer agent as it wanders around. The middle image shows 100 avoider agents coloured yellow that try to avoid the wanderer agent. In this case, most of the avoider agents are pointed away from the wanderer agent and will actively move away from it during the simulation. The right image shows the situation when 50 follower agents and 50 avoider agents are in the environment together with the wanderer agent. The image shows most of the follower agents in green pointing towards the wanderer, and most of the avoider agents in yellow pointing away from it. At the beginning, all agents are randomly spaced throughout the environment, but after the simulation has run for a short number of ticks, what usually emerges is that both the follower and avoider agents will start clumping together as shown in the image.

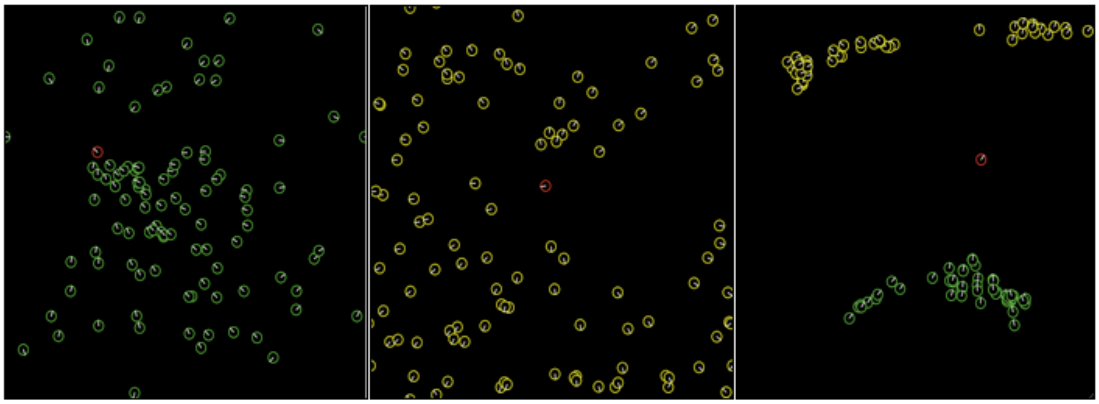


Figure 6.10 Screenshots of the Follow and Avoid model.

The code for the model is shown in NetLogo Code 6.6.

Download free eBooks at bookboon.com

```
breed [wanderers wanderer] ; wanders around
breed [followers follower] ; tries to follow the wanderer
breed [avoiders avoider] ; tries to avoid the wanderer

to setup

   clear-all

   ; set shapes for the breeds
   set-default-shape wanderers "directional-circle"
   set-default-shape followers "directional-circle"
   set-default-shape avoiders "directional-circle"

   ; create a wanderer, follower and an avoider at random locations
   create-wanderers 1 [ default red ]
   create-followers number-of-followers [ default green ]
   create-avoiders number-of-avoiders [ default yellow ]
end

to-report number-of-agents
   ; used to show that number of turtle agents is constant even when
   ; all the followers clump on top of each other
   report count turtles
end

to default [colour] ; creates default settings for boid
   set color colour    ; sets colour using passed parameter
   setxy random-xcor random-ycor ; sets an initial random position
   set size 3 ; default turtle size
end

to go
   ask wanderers ; wanderer's instructions
   [
      lt random 30 ; randomly turn to the left
      rt 15
      fd boid-speed ; variable user defined speed
   ]
   ask followers ; follower's instructions
   [
      fd boid-speed / speed-scale ; moves forward at user defined speed
      ; follower searches for wanderer in its radius
      if any? wanderers in-radius radius-detection
      [ set heading (towards wanderer 0) - random boid-random-heading
         + random boid-random-heading
   ; adjusts heading to point towards wanderer ]
]

ask avoiders ; avoiders' instructions
   [
      fd boid-speed / speed-scale ; moves forward at user defined speed
      ; avoider searches for wanderer in its radius
```


Download free eBooks at bookboon.com

```
      if any? wanderers in-radius radius-detection
      [ set heading (towards wanderer 0) + 180 - random boid-random-heading
      + random boid-random-heading ]
      ; adjusts heading to point away from wanderer
   ]
end
```

NetLogo Code 6.6 The code for the Follow and Avoid model shown in Figure 6.10.

The behaviour of the three breeds of agents is defined by the `wanderers`, `followers` and `avoiders` procedures. The first defines the behaviour for the wanderer agent so that it wanders around in a semi-random fashion. The second defines the behaviour for the follower agent that consists of the agent first moving forward a user-defined amount according to the interface variables `boid-speed` and `speed-scale`. Then it uses the NetLogo `in-radius` reporter to detect whether the wanderer is in its circular field of vision whose size is defined by the `radius-detection` Interface variable. If there is, then it will move toward it. The avoider agent's behaviour is defined in a similar manner, the only difference being that it heads in the opposite direction (180°) away from the wanderer instead of towards it.

Need help with your dissertation?

Get in-depth feedback & advice from experts in your topic area. Find out what you can do to improve the quality of your dissertation!

Get Help Now

Go to www.helpmyassignment.co.uk for more info

Helpmyassignment

Click on the ad to read more

Download free eBooks at bookboon.com

The Flocking With Obstacles model is a modification of the Flocking model provided in the NetLogo Models Library. The library model uses the standard method for implementing flocking behaviour devised by Craig Reynolds. In this approach, the flocking emerges from the application of three underlying steering behaviours. These are: *separation*, where the boid tries to avoid getting too close to other boids; *alignment*, where the boid tries to move in the same direction as nearby boids; and *cohesion*, where the boid tries to move towards other boids unless they are too close. With the modified model, the user has the extra option of adding various objects into the environment, such as a coral reef, sea grass and a shark. This is in order to simulate what happens when the flock encounters one or more objects, and to better simulate the environment for a school of fish. Some screenshots of the modified model are shown in Figure 6.11.

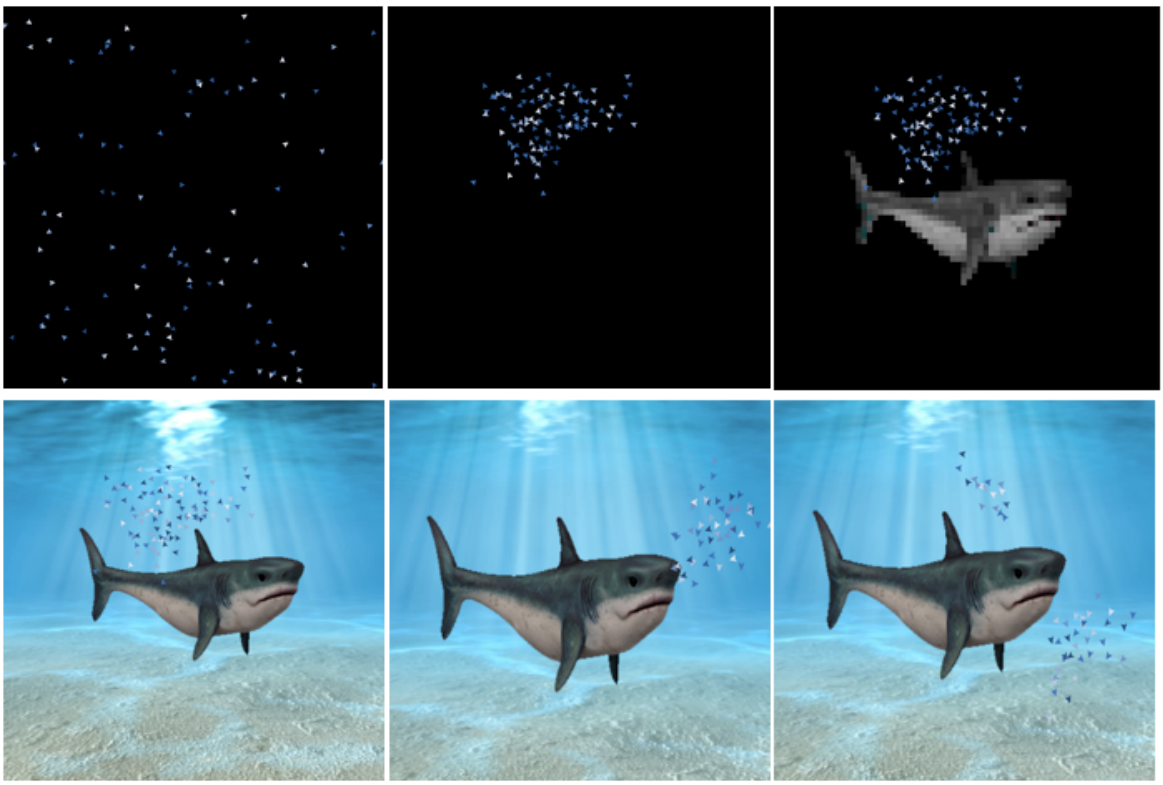


Figure 6.11 Screenshots of the Flocking With Obstacles model.

The top left image shows the model at the start after the `setup` button in the interface has been pressed. The middle top image shows the model after it has been run for a short while and a school of turtle agents has formed. The right top image shows the model with collision patches added in the middle in the shape of a shark loaded immediately after the previous image was taken. These patches cause the turtle agents to move away when they have collided with them. The bottom left image shows the background image overlaid onto the same image. The middle bottom image shows the school approaching the object from a different direction. The bottom right image shows the scene not long after – now the school has split into two sub-schools after the collision and are heading away from the object.

Download free eBooks at bookboon.com

The collision patches work by the model stipulating that any patch that is not black is to be avoided; that is, all colours excluding black forces the boids to turn around 180° in order to avoid a collision. Another change to the model is that the speed of the boids can now be controlled from the interface to enable greater testing of individual movements and this also provides a means of analysing the reactions of the boids.

The code for the relevant parts of the modified model that define the behaviour of the agents is listed in NetLogo Code 6.7.

Click on the ad to read more

Download free eBooks at bookboon.com

```
turtles-own [
   flockmates ;; agentset of nearby turtles
   nearest-neighbor ;; closest one of our flockmates
]

to setup
   clear-all
   crt population
      [ set color blue - 2 + random 7
      set size 1.5
      setxy random-xcor random-ycor ]
end
to go
   ask turtles [ flock ]
   repeat 5 [ ask turtles [ fd set-speed / 200 ] display ]
   tick
end

to flock
   find-flockmates
   if any? flockmates
      [ find-nearest-neighbor
       ifelse distance nearest-neighbor < minimum-separation
      [ separate ]
      [ align
          cohere ] ]
   avoid-obstacles
end

to avoid-obstacles
; avoid anything nearby that is not black
   if (any? patches in-cone 2 300 with [pcolor != black])
   [ rt 180 ] ; head in opposite direction
end

to find-flockmates
   set flockmates other turtles in-radius vision
end

to find-nearest-neighbor
      set nearest-neighbor min-one-of flockmates [distance myself]
end

to separate
   turn-away ([heading] of nearest-neighbor) max-separate-turn
end

to align
   turn-towards average-flockmate-heading max-align-turn
end
```


Download free eBooks at bookboon.com

```
to-report average-flockmate-heading
      report atan sum [sin heading] of flockmates
                  sum [cos heading] of flockmates
end

to cohere
    turn-towards average-heading-towards-flockmates max-cohere-turn
end

to-report average-heading-towards-flockmates
    report atan mean [sin (towards myself + 180)] of flockmates
      mean [cos (towards myself + 180)] of flockmates
end

to turn-towards [new-heading max-turn]
   turn-at-most (subtract-headings new-heading heading) max-turn
end

to turn-away [new-heading max-turn]
   turn-at-most (subtract-headings heading new-heading) max-turn
end

to turn-at-most [turn max-turn]
   ifelse abs turn > max-turn
      [ ifelse turn > 0
      [ rt max-turn ]
      [ lt max-turn ] ]
      [ rt turn ]
end
```

NetLogo Code 6.7 The code for the Flocking With Obstacles model shown in Figure 6.11.

The `setup` procedure creates a random population of turtle agents. The ask command in the `go` procedure defines the behaviour of the agents – it simply calls the `flock` procedure. Here the agent first checks to see if there are any other agents within its cone of vision, then if there are any, it looks for the nearest neighbour, and then applies the separation steering behaviour as defined by the `separate` procedure if it is too close. Otherwise it applies the alignment steering behaviour as defined by the align procedure followed by the cohesion steering behaviour as defined by the `cohere` procedure. These three procedures make use of either the `turn-away` or `turn-towards` procedures that make the boid turn away from or towards a particular reference heading given the boid's current heading. The reference heading for the separation steering behaviour is the heading of the boid's nearest neighbour, for the alignment steering behaviour it is the average heading of the boid's flock mates, and for the cohesion steering behaviour it is the mean heading towards the boid's flock mates.

Download free eBooks at bookboon.com

In the simulation, a number of emergent phenomena can be witnessed. The flock quickly forms at the beginning when no obstacles have been loaded. A noticeable spinning effect can also be observed of the boids within the flock if the initial interface parameters are set as `minimum-separation` = 1.25 patches, `max-align-turn` = 15.00 degrees, `max-cohere-turn` = 15.00 degrees and `max-separate-turn` = 4.00 degrees. When the school encounters an obstacle, it changes direction as a group with individual boids usually reacquiring the flock very quickly if they become separated. When enough of the boids have altered their course, the remainder of the school follows suit without ever having been in collision with the obstacle. Occasionally, the school will split into two separate schools heading in different directions as shown in the bottom right image of Figure 6.10.

The Crowd Path Following model implements behaviour of boids in a crowd that are following a path. The behaviour of the boids in the model can have two variations – a basic crowd path following behaviour, and one with collision avoidance (this is set using the `behaviour` slider in the Interface). Figure 6.12 provides two screenshots of the model for the two different behaviours. The left image shows the 'crowd' of boids heading down the path roughly in the same direction from left to right. This occurred after the simulation had been running for a short while. Note that some of the agents are very close to each other – this is because there is no collision avoidance, unlike with the second behaviour shown on the right. Although the core code for these behaviours is similar (for the code, follow URL link at the bottom of this chapter) being based on using the `in-cone` command as with the other boid agent implementations above, the resulting behaviour with obstacle avoidance is noticeably different. For example, more sub-groups of agents now try to go against the flow, as shown in the right image.

Click on the ad to read more

Download free eBooks at bookboon.com

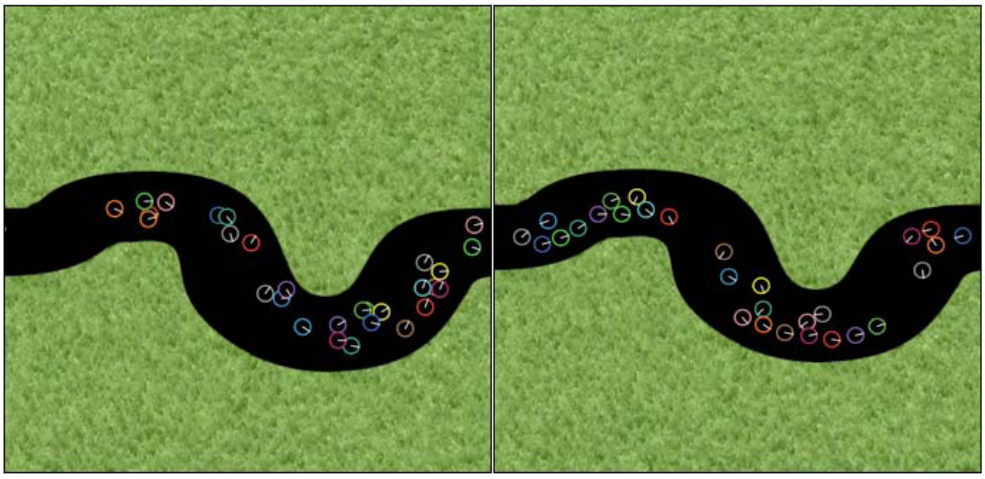


Figure 6.12 Screenshots of the Crowd Path Following model with population = 30: basic crowd following behaviour (left image); with collision avoidance (right image).

A faithful implementation of Reynolds' boids should 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 have shown how an approximation to Reynolds approach can be achieved relatively easy, and in many cases, the resultant behaviour of the boids is as desired.

6.8 Summary

Behaviour based Artificial Intelligence (BBAI) adopts the behavioural approach to building intelligent systems. This approach decomposes intelligence into separate independent semi-autonomous modules that describe distinct behaviours. Behaviour for an agent is a series of actions it performs when interacting with an environment. The specific order or manner in which the movements are performed and the overall outcome that occurs as a result of the actions defines the particular type of behaviour.

Different perspectives lead to different explanations, and different ways things are understood. The frame of reference problem concerns the difficulty of understanding the behaviour in different species. Not only do the points of view look different to each agent, they sound, feel, smell and taste different as well, because of the different embodiment of the agent doing the observation compared to the agent being observed. (For example, try to imagine being an ant.) It is important not to attribute the wrong explanations from observations to the mechanisms behind the behaviour of an embodied agent situated within an environment, and especially important to avoid attributing complicated mechanisms (e.g. some form of "intelligence") to the observed agent's behaviour when it has a different frame of reference, i.e. a different embodiment.

Download free eBooks at bookboon.com

A summary of important concepts to be learned from this chapter is listed below:

- Simple rules may lie at the heart of complex systems.
- Simple reactive behaviour may be behind complex phenomena.
- Adaptive behaviour is behaviour where the agent has changed its behaviour in response to a change in the environment.
- Evolving behaviour is behaviour that has come about due to genetic evolution (i.e. this requires more than one generation where genetic traits are passed down to offspring).
- Emergent behaviour is some property that emerges from agent-agent interactions or agent-environment interactions that could not have arisen without those interactions and is not the result of a simple linear combination of those interactions.
- Self-organising behaviour for a multi-agent system occurs when the agents applying local rules creates some pattern or structure as an emergent property.
- Stigmergy is when agents use the environment to communicate and interact. For example, ants and bees use the environment to tell each other where to find sources of food. Humans use it to build complex information systems and tell each other where to find sources of information.
- Swarm intelligence is a collection of agents that use stigmergic local knowledge to self-organize and co-ordinate their behaviours.

The code for the NetLogo models described in this chapter can be found as follows:

Model	**URL**
Crowd Path Following	http://files.bookboon.com/ai/Crowd-Path-Following.nlogo
Flocking With Obstacles	http://files.bookboon.com/ai/Flocking-With-Obstacles.nlogo
Follow and Avoid	http://files.bookboon.com/ai/Follow-And-Avoid.nlogo
Obstacle Avoidance 1	http://files.bookboon.com/ai/Obstacle-Avoidance-1.nlogo
Obstacle Avoidance 2	http://files.bookboon.com/ai/Obstacle-Avoidance-1.nlogo
Wall Following Events	http://files.bookboon.com/ai/Wall-Following-Events.nlogo

Model	**NetLogo Models Library (Wilensky, 1999) and URL**
Fireflies	Biology > Fireflies http://ccl.northwestern.edu/netlogo/models/Fireflies
Flocking	Biology > Flocking http://ccl.northwestern.edu/netlogo/models/Flocking
Heatbug0	Biology > Heatbugs h000p://ccl.northwestern.edu/netlogo/models/Heatbugs
State Machine Example	Code Examples > State Machine Example http://ccl.northwestern.edu/netlogo/models/StateMachineExample
Termites	Biology > Termites http://ccl.northwestern.edu/netlogo/models/Termites
Wall Following Example	Code Examples > Wall Following Example; see modified code at: http://files.bookboon.com/ai/Wall-Following-Example-2.nlogo

Download free eBooks at bookboon.com

7 Communication

Greek manuscript

In considering human history, the language community is a very natural unit. Languages, by their nature as means of communication, divide humanity into groups; only through a common language can a group of people act in concert, and therefore have a common history. Moreover the language that a group shares is precisely the medium in which memories of their joint history can be shared. Languages make possible both the living of a common history, and also the telling of it.

Nicholas Ostler, Empires of the Word, 2005.

Click on the ad to read more

Download free eBooks at bookboon.com

The new method of estimating entropy exploits the fact that anyone speaking language possesses, implicitly, an enormous knowledge of the statistics of the language. Familiarity with the words, idioms, clichés and grammar enables him to fill in missing or incorrect letters in proof-reading, or to complete an unfinished phrase in conversation.

C.E. Shannon, Prediction and entropy of printed English, 1951.

The purpose of this chapter is to introduce the important topics of communication and language, and the vital role they play in agent-to-agent interaction. The chapter is organised as follows. Section 7.1 defines the relationship between communication, information and language. Section 7.2 highlights the diversity of human language. Section 7.3 looks at communication within communities of agents called social networks. Different types of communicating behaviour are discussed in Section 7.4. An important hallmark of social and computer networks called the small world phenomenon is described in Section 7.5, along with a method for measuring the average degree of separation in a network called Dijkstra's algorithm. Section 7.6 looks at how communicating agents can be used to search networks, and how different communicating behaviours can be more effective than others at performing the search. Section 7.7 defines entropy, a probabilistic method for measuring information, and Section 7.8 shows how to calculate it in NetLogo. Section 7.9 looks at a statistical approach to the modelling of language, and the entropy of a language is defined in Section 7.10. Section 7.11 highlights how the purpose of communication is to transmit the meaning of the message, the problems associated with defining what meaning is, and briefly looks at the complexities of meaning present in human language.

7.1 Communication, Information and Language

Communication may be defined as the process of sharing or exchanging of information between agents. An agent exhibits communicating behaviour when it attempts to transmit information to another agent.

A sender agent or agents transmits a message through some medium to a receiver agent or agents. The term communication in common English usage can also refer to interactions between people that involve the sharing of information, ideas and feelings. Communication is not unique to humans, though, since animals and even plants also have the ability to communicate with each other.

Language can be defined as the set of symbols that agents communicate with in order to convey information. In Artificial Intelligence, human language is often called 'natural language' in order to distinguish it from computer programming languages. Communicating using language is often considered to be a uniquely human behavioural trait. Human language, such as spoken, written or sign, is distinguished from animal communication systems in that it is learned rather than inherited biologically. Although various animals exhibit the ability to communicate, and some animals such as orangutans and chimpanzees even have the ability to use certain features of human language, it is the degree of sophistication and complexity in human language that distinguishes it from animal communication systems. Human language is based on the unique ability of humans to think abstractly, using symbols to represent concepts and ideas.

Download free eBooks at bookboon.com

Language is defined by a set of socially shared rules that define the commonly accepted symbols, their meaning and their structural relationships specified by rules of grammar. These rules describe how the symbols can be manipulated to create a potentially infinite number of grammatically correct symbol sequences. The specific symbols chosen are arbitrary and can be associated with any particular phoneme, grapheme or sign.

Linguistics is the scientific study of language which can be split into separate areas of study: grammar is the study of language structure; morphology is the study of how words are formed and put together; phonology is the study of systems of sounds; syntax concerns the rules governing how words combine into phrases and sentences; semantics is the study of meaning; and pragmatics concerns the study of language and use and the contexts in which it is used.

7.2 The diversity of human language

In all natural languages, there is a wide variation in usage as well as frequent lack of agreement amongst language users. For example, Table 7.1 lists some examples of acceptable 'English' sentences from various regions of the world (Newbrook, 2009). Each of the sentences is regarded as 'normal' English for the region shown on the right, and yet most people outside those regions would argue differently, and in many cases have difficulty in understanding their meaning.

'English' sentences	**Their origin**
Let's buy some food home!	Singapore
Don't smoke without causing an explosion!	South Wales
AIDS is very popular in Africa.	Hong Kong
My hair needs washed.	Scotland, Northern Ireland, parts of USA
Whenever my baby was born I was 26.	Northern Ireland
Her outlook is very beautiful.	Hong Kong
John smokes a lot anymore.	Mid-West USA
I am difficult to study.	Hong Kong
I might could do it.	Scotland, Northern England, India, parts of USA
My name is spelt with four alphabets.	Singapore
You must beware of your handbag!	Hong Kong
We'll be there nine while ten.	Lancashire, Yorkshire
He loves his car than his girlfriend.	India, parts of Africa
Come here till I punch you!	Ireland, Liverpool, Cumbria, parts of Scotland
I'm after losing my ticket.	Ireland
My brother helps me with my studies and so do my car.	Hong Kong
I been know your name.	USA (Black)
I use to live there now.	Singapore
My grandfather died for a long time.	Hong Kong
I am having a nice car.	India, Singapore
I'll give it him.	Northern England (standard)
Robots can do people not like jobs.	Hong Kong (low proficiency)

Table 7.1 Some "English" sentences and their origin (Newbrook, 1996).

Download free eBooks at bookboon.com

Concerning the English language, David Crystal (1988) states: "The English language stretches around the world: from Australia to Zimbabwe, over 300 million people speak it as their mother tongue alone… And yet, despite its astonishingly widespread use as a medium of communication, every profession and every province – indeed, every individual person – uses a slightly different variant." English has many different regional dialects (such as American, British, Australian and New Zealand English), as well as many sub-dialects within those regions. There are also dialects that cut across regional lines, for example, "Public School English" in Britain, Black English in America and Maori English in New Zealand. And in every country, there are countless social variations that "possess their own bewildering variety of cants, jargons and lingoes" (Claiborne 1990, page 20). One of the more colourful examples is a dictionary on Wall Street slang entitled *High steppers, fallen angels, and lollipops* (Odean, 1989). It illustrates how such language can become almost unintelligible to the uninitiated. (For example, what is a 'high stepper' or a 'fallen angel'?)

Hudson (1983, page 69) writes the following in *The language of the teenage revolution* about the resentment of older people to the language used by contemporary teenagers: "…perhaps, they dislike the fact that teenagers speak another kind of language, using a considerable number of expressions which they themselves find either incomprehensible or repulsive."

Click on the ad to read more

Download free eBooks at bookboon.com

As well as language being diverse, there are many different ways that language is expressed and used. Spoken language is markedly different from written language, as illustrated from the following example taken from Crystal (1981):

> "This is part of a lecture, and I chose it because it shows that even a professional speaker uses structure that would rarely if ever occur in written English, and displays a 'disjointedness' of speech that would be altogether lacking there. (Everyday conversation provides even more striking differences.) The dot (.) indicates a short pause, the dash a longer pause, and the *erm* is an attempt to represent the noises the speaker made when he was audibly hesitating.
>
> … – *and I want . very arbitrarily if I may to divide this into three headings --- and to ask .* erm . *three questions . assessment why – assessment of what – and assessment how . so this is really . means I want to talk about . first of all the purposes of assessment – why we are assessing at all – erm secondly the kind of functions and processes that are being assessed – and thirdly I want to talk about techniques – …*"

Baugh (1957, page 17) reminds us that language is not just "the printed page" relatively uniform and fixed, as many people think it to be. Language is "primarily speech" and writing "only a conventional device for recoding sounds." He further states that as the repeated muscular movements which generate speech are subject to gradual alteration on the part of the speaker:

> "each individual is constantly and quite unconsciously introducing slight changes in his speech. There is no such thing as uniformity in language. Not only does the speech of one community differ from that of another, but the speech of different individuals of a single community, even different members of the same family, is marked by individual peculiarities."

Some other distinctive forms of language are, for example, poetry, legal documents, newspaper reporting, advertising, letter writing, office correspondence, telegrams, telephone conversations, electronic mail, Usenet news articles, scientific papers and political speeches. Each has their own flavour, quirks and style. And within each form there are individual authors who have their own distinctive styles of language. The plays of Shakespeare and the science fiction novels of H.G. Wells are two examples of very distinct literary styles. In fact, every single person has their own style of language, or *idiolect* with its own unique characteristics that can be readily discerned by other people (Fromkin *et al.*, 1990).

Download free eBooks at bookboon.com

As well as the standard dialects there are many non-standard in use. Fromkin *et al.* (1990, page 253) state that "though every language is a composite of dialects, many people talk and think about a language as if it were a 'well-defined' fixed system with various dialects diverging from this norm". They state that these non-standard dialects are often regarded as being "deficient, illogical and incomplete" when compared to the standard dialects. They use the example of a prominent author on language, Mario Pei (1964), who once accused the editors of Webster's *Third New International Dictionary* (published in 1961) of confusing "to the point of obliteration the older distinction between standard, nonstandard, colloquial, vulgar and slang".

Crystal (1981), in *Linguistics*, states that "many people's knowledge of normal language is very shaky; and their ability to describe what it is they know is shakier still." Quite often even experts argue over the finer points of language use. There are numerous dictionaries explaining common usage or less common usage such as jargon, slang, clichés, idioms and figures of speech, for example *Up the boohai shooting pukekas: a dictionary of Kiwi slang* (McGill, 1988). Finding order in all this chaos has been a goal of many people ever since Samuel Johnson compiled his dictionary of English in 1755, and the first attempts were made to compile a grammar for English not long after (Crystal, 1988).

Click on the ad to read more

Download free eBooks at bookboon.com

7.3 Communication via communities of agents

Communication is by definition conducted between two or more agents. An important aspect of human communication is that it is conducted between communities and groups of agents. For human agents, the type of communication via human language that is deemed appropriate in each community can often be used to identify and distinguish that community or group. This is manifested by different languages, dialects, sub-dialects and sociolects (language used amongst a particular social group) that are distinguished by variations in vocabulary, grammar, and pronunciation often by the use of variations in the lexicon such as jargons, and slang.

Social networking amongst communities of agents is an important aspect of human language. A social network refers to a social structure made up of agents that are linked to each other by some common shared trait. A social network can be represented by a graph, where nodes represent the agents and links between nodes define the relationships between the agents. The term social networking has become much more prevalent in common English usage because of the recent emergence of social networking Web sites such as Facebook, Myspace and Twitter.

Language is defined by the members of the social network comprising the language community. Within that community, there may be sub-networks for each of the dialects, sub-dialects and sociolects.

'Living' languages as opposed to 'dead' languages such as Latin are also dynamic, with new words and phrases constantly being coined to describe new concepts and ideas, and these new words and phrases are spread via the links in the social networks. The Language Change model provided in the Models Library in NetLogo illustrates how the structure of the social network combined with the variations of language users can affect language change. In this model, the nodes in the network represent the language users, and there are two variants of the language generated by two different grammars (0 and 1) in competition within the network.

Language users interact with each other each iteration of the simulation by first speaking either grammar 0 or 1 to the neighbours they are connected with in the social network. During the subsequent listening phase, they then adjust their grammar based on what was received during the speaking phase.

Download free eBooks at bookboon.com

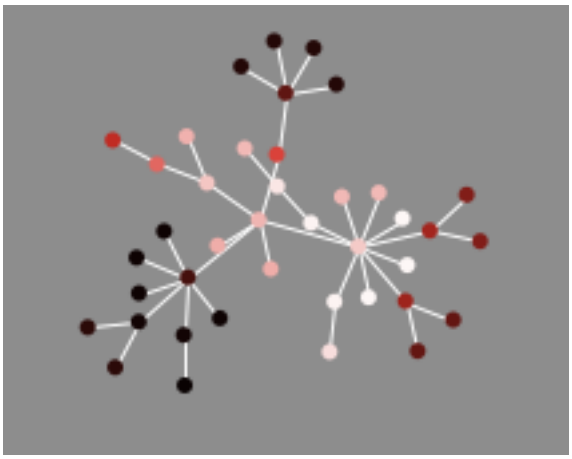


Figure 7.1 Screenshot of sample output produced by the Language Change model.

A screenshot of sample output produced by the model is shown in Figure 7.1. The output was produced using default settings and the reward update algorithm. The white nodes in the figure represent language users who access grammar 1, the black nodes those who access grammar 0. The red shaded nodes represent language users who have the ability to use both grammars. The colour represents the probability they will access either grammar 1 or 0, the higher the probability, then the darker the shading. A selection of the code in the model is shown in NetLogo Code 7.1.

Deloitte.

Discover the truth at www.deloitte.ca/careers

© Deloitte & Touche LLP and affiliated entities.

Click on the ad to read more

Download free eBooks at bookboon.com

```
breed [nodes node]

nodes-own [
   state ; current grammar state - [0,1]
   orig-state ; save for resetting the original states
   spoken-state ; stores output of agent's speech - 1 or 0
]

to communicate-via [ algorithm ] ; node procedure
   ;; *Probabilistic Grammar* ;;
      ;; speak and ask all neighbors to listen
      if (algorithm = "reward") [
      speak
      ask link-neighbors [
           listen [ spoken-state ] of myself
      ]
end

;;; listening uses a linear reward/punish algorithm
to listen [heard-state] ; node procedure
   let gamma 0.01 ; for now gamma is the same for all nodes
   ;; choose a grammar state to be in
   ifelse (random-float 1.0 <= state) [
      ;; if grammar 1 was heard
      ifelse (heard-state = 1) [
      set state (state + (gamma * (1 - state)))
      ][
      set state (1 - gamma) * state
      ]
   ][
      ;; if grammar 0 was heard
      ifelse (heard-state = 0) [
      set state (1 - gamma) * state
      ][
      set state gamma + ((1 - gamma) * state)
      ]
   ]
end

to update-color
   set color scale-color red state 0 1
end
```

NetLogo Code 7.1 Selected parts of the code for the Language Change model.

Download free eBooks at bookboon.com

The language user is defined as a `node` breed at the beginning, and each node owns a `state` variable, a number between `0` and 1 that is a weighting that reflects the user's probability of accessing grammar 0 or 1. The `communicate-via` procedure lists the code associated with just the reward update algorithm (there are two other update algorithms not included here – individual and threshold – that can be used to produce different network behaviour). With the reward update algorithm, the current node agent first speaks then listens to all its link neighbours in the network. The algorithm uses a linear reward/punishment weighting scheme as shown in the `listen` procedure to set the node agent's `state` variable. At the bottom of the code listing, the `update-color` procedure has been included to show how the colour of node agent is set depending on the current grammar weighting stored in the `state` variable.

Multiple runs of the model show that the language users in the simulation have a tendency to end up using a single grammar that dominates throughout the social network. However, often the other grammar still remains in pockets throughout the network, and this reflects the formation of dialects and sub-dialects in human social networks in real life.

7.4 Communicating Behaviour

Communicating behaviour comes in many forms. Table 7.2 lists some of the many different forms of communicating behaviour in humans. The table lists the behaviour (speaking, writing, signing and so on) and the medium in which the communication takes place (for example, for writing, the medium is usually paper). This is closely related to the human sense(s) and language being used to decode the information being communicated as listed in the table. Also listed is whether the communication is one-to-one or one-to-many, whether the receiver agent needs to decode the information immediately otherwise it will be lost or has the ability to delay the decoding, and whether the sender and receiver agents need to be close to each other in order to carry out the form of communication, or whether it can be performed remotely.

For much of history, human communication has been restricted to a few basic forms such as speaking and writing, but over the last century or so, human communicating behaviour has undergone rapid transformation due to technological advances enabling new forms of communication that were previously not possible, for example, broadcasting, and more recently texting and blogging. Technology is also blurring the nature of existing communication – whereas only one-to-one immediate delivery was possible in some forms, advances have allowed us to expand to include variations such as 1-to-many delayed delivery – for example, radio and TV is now being broadcast by streaming, and therefore it is now no longer necessary to have to listen or watch these media immediately as alternative means of communication are available where we can wait until a more convenient time. This is manifested in Table 7.2 with the columns with empty cells gradually being filled in with ✓'s as technology enables a previously unavailable option to become possible.

Download free eBooks at bookboon.com

Behaviour	Sense	Medium	Language	1-to-1	1-to-many	Immed.	Delayed	Close	Remote	Example(s)
Speaking by word of mouth	Hearing	Sound	Spoken	✓		✓		✓		Talking
Writing	Vision	Paper	Written		✓		✓		✓	Books, news papers
Signing	Vision	Light	Sign	✓		✓		✓		ASL
Writing using Braille	Touch	Braille	Braille		✓		✓		✓	Books for the blind
Phoning	Hearing	Sound	Spoken	✓		✓			✓	Phone
Texting	Vision	Mixed	Written	✓			✓		✓	Phone
Tweeting	Vision	Mixed	Written		✓		✓		✓	Phone, Web
Broad casting	Mixed	Mixed	Mixed		✓	✓	✓		✓	Radio, TV
Lecturing via 'black boards'	Vision	Black-board	Written		✓	✓	✓	✓		Black board, white board
Blogging	Mixed	Internet	Mixed		✓	✓	✓		✓	Web pages

Table 7.2 The many forms of human communicating behaviour.

In order to illustrate how communicating behaviour affects how language is spread throughout a social network, we can run some simulations using NetLogo models. The first, the Communication-T-T Example model that comes with the NetLogo Models Library, provides a simple illustration of how 'word of mouth' communication can be extremely effective at spreading a message throughout a network. A screenshot of the model in action is shown in Figure 7.2.

Download free eBooks at bookboon.com

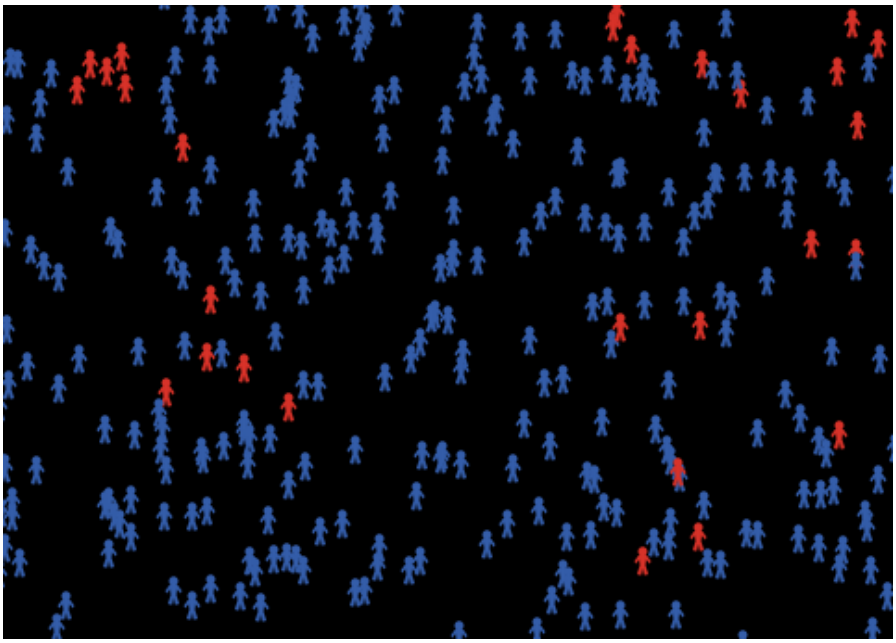


Figure 7.2 Screenshot of sample output produced by the Communication-T-T model.

The model works by first creating a number of agents at random positions in the environment, one of which is designated to be the agent who starts spreading the message. Then during the simulation, these agents move about randomly, and those who have the message will spread it via word-of-mouth to any agents who are nearby. The figure shows the simulation at an early stage – the message has been spread to only a small percentage of the agents (the ones coloured in red); eventually, the message will rapidly get spread to all the agents in the environment. However, even at this stage of the simulation, it is readily apparent the way the word-of-mouth method of communication works and why it is so effective. The message has been transmitted to clusters of agents who are near to each other. These clusters will rapidly grow and take over the whole network like the ripple effect of waves spreading out across a pond after a stone has been thrown into it.

The code for the model is shown in NetLogo Code 7.2.

Download free eBooks at bookboon.com

```
turtles-own [
message? ;; true or false: has this turtle gotten the message yet?
]

to setup
   clear-all
   crt 400 [
      set shape "Person"
      set size 2
      set message? false
      setxy random-xcor random-ycor
   ]
   ask one-of turtles
   [ set message? true ] ;; give the message to one of the turtles
   ask turtles [ recolor ]
end

to go
   ask turtles [ move ]
   ask turtles [ communicate ]
   ask turtles [ recolor ]
   tick
end

;; move randomly
to move ;; turtle procedure
   fd random 4
   ;; turn a random amount between -40 and 40 degrees,
   ;; keeping the average turn at 0
   rt random 40
   lt random 40
end

;; the core procedure!
to communicate ;; turtle procedure
   if any? other turtles-here with [message?]
      [ set message? true ]
end

; color turtles with message red, and those without message blue
to recolor ;; turtle procedure
   ifelse message?
      [ set color red ]
      [ set color blue ]
end
```

NetLogo Code 7.2 Code for the modified Communication-T-T model.

Download free eBooks at bookboon.com

The code has been slightly modified for visualisation purposes from that provided by the Models Library by reducing the number of agents created and by changing the size and shape of the agents. The `setup` procedure creates 400 turtle agents at random locations, with one of them designated as having the message at the start. During the `go` procedure, each agent first moves in a random direction, then each agent who has the message communicates the message to any other agents who are at the same patch location, and finally the colours of each agent are reset to reflect whether they have the message or not.

7.5 The Small World Phenomenon and Dijkstra's algorithm

Another NetLogo model, the Being Kevin Bacon model has been created to find out how different forms of communicating behaviour compare at spreading information throughout a social network. Before explaining the model, however, we need to learn about an important hallmark of networks called the 'small world phenomenon' and the related phrase 'six degrees of separation'. The small world phenomenon was first investigated by Stanley Milgram, a social psychologist at Yale University.

He conducted an experiment to show we are on average only six "steps" away from anybody else on Earth. In the experiment, letters were sent out to randomly selected people in Omaha and Wichita in the USA. These people were asked if they knew a person *X* living in Boston. If not, they were then asked to forward the letter onto someone who they thought might know *X*.

The results of the experiment were that 64 of the 256 letters sent did reach the target; some chains of successful letters were only 1 or 2 long while others were over 10 long. Significantly, the average path length was around 5.5 to 6 – this is where the phrase 'six degrees of separation' originated. The idea is that even in a very large network, a simple method of spreading a message via word of mouth can be extremely effective because of the ripple effect as the message is spread. Although the message in this experiment was written down rather than spoken, the method can still be considered analogous to word of mouth communication in humans as direct contact is still required to spread the message albeit via a third transmitting agent, the postman.

The degree of separation is a useful means for measuring the connectivity of a network, whether it is a social network or computer network. Many networks contain what are called 'super-nodes' – hubs that are super-connected to many other nodes in the network. Here the number of connections that the hub has with other nodes is substantially greater than the average number of connections per node. These hubs can aid the speed at which communication is spread throughout the network, since once the message has reached a hub, then the message can reach a greater part of the network immediately without having to go through further intermediate nodes.

Download free eBooks at bookboon.com

In any network we can identify a hub and then work out the degree of separation a particular node has from it – that is, how many nodes need to be visited along the shortest path from the node to the hub. In a social network, this might be useful to determine how 'close' one person is to a famous person – the closer one is to the celebrity, the greater the reflected glory. Mathematicians have in fact used a number called the Erdös Number in order to measure the collaboration distance from the well-known and prolific Hungarian mathematician, Paul Erdös, who had hundreds of collaborators and worked in many fields including combinatorics, graph theory, number theory, set theory and probability theory. The Erdös Number is calculated recursively in the following manner:

- If you have a paper co-authored by Erdös, you have an Erdös number of 1.
- Otherwise, if you have a paper published with a co-author who has an Erdös number of 1, you have an Erdös number of 2.
- Otherwise, if you have a paper published with a co-author who has an Erdös number of n, you have an Erdös number of $n + 1$.

This distance metric can readily be applied to other networks, not just networks of people who publish academic papers. A very different network is the network of actors who have starred in films – here the 'centre' of the film actors' universe has been chosen to be Kevin Bacon, an American film and theatre actor who has starred in many films such as *A Few Good Men*, *JFK*, *Apollo 13*, *and Footloose*. The Kevin Bacon number is calculated as follows:

SIMPLY CLEVER
ŠKODA

Do you like cars? Would you like to be a part of a successful brand?
We will appreciate and reward both your enthusiasm and talent.
Send us your CV. You will be surprised where it can take you.
Send us your CV on
www.employerforlife.com
Click on the ad to read more

Download free eBooks at bookboon.com

- If you have starred in a film with Kevin Bacon, you have a Kevin Bacon number of 1.
- Otherwise, if you have starred in a film with a co-star who has a Kevin Bacon number of 1, you have a Kevin Bacon number of 2.
- Otherwise, if you have starred in a film with a co-star who has a Kevin Bacon number of n, you have a Kevin Bacon number of $n + 1$.

The Erdös and Kevin Bacon Numbers provide rough measures of the degree of belonging an individual has to a particular community such as Mathematicians and/or film actors – the higher these numbers, the less the involvement with that community. In any community, there are individuals who are also members of other communities, and often, these individuals can help spread a message more quickly, especially between mostly disjoint communities where interaction is minimal – that is, where inter-community communication is much less than intra-community communication. Those individuals with multiple memberships serve as the means for the spread of ideas or messages that may be well known within one community to other communities where the same message may be less well known or even completely unknown. In academic research, often important breakthroughs have been achieved through interdisciplinary research, for example.

Just as we can measure the degree of separation from one well-known individual in a particular community, we can measure the combined separation from two or more communities. For the Mathematics and film actors' communities, we have the Erdös-Bacon Number that is the sum of the Erdös Number plus the Kevin Bacon Number. Not unexpectedly, not many people have both an Erdös Number and a Kevin Bacon Number, reflecting that there is little interaction between mathematicians and film actors. However, there are and have been some notable exceptions such as astronomer Carl Sagan and theoretical physicist Stephen Hawking, and also Natalie Portman, an actress famous for her role in the *Star Wars* movies. Natalie Portman has an Erdös Number of 5 due to authorship of psychology papers during her Harvard degree in psychology. She also has a Kevin Bacon Number of 1 (she starred in the movie *New York, I Love You* with Kevin Bacon). Her Erdös-Bacon Number is therefore 6 (5 + 1).

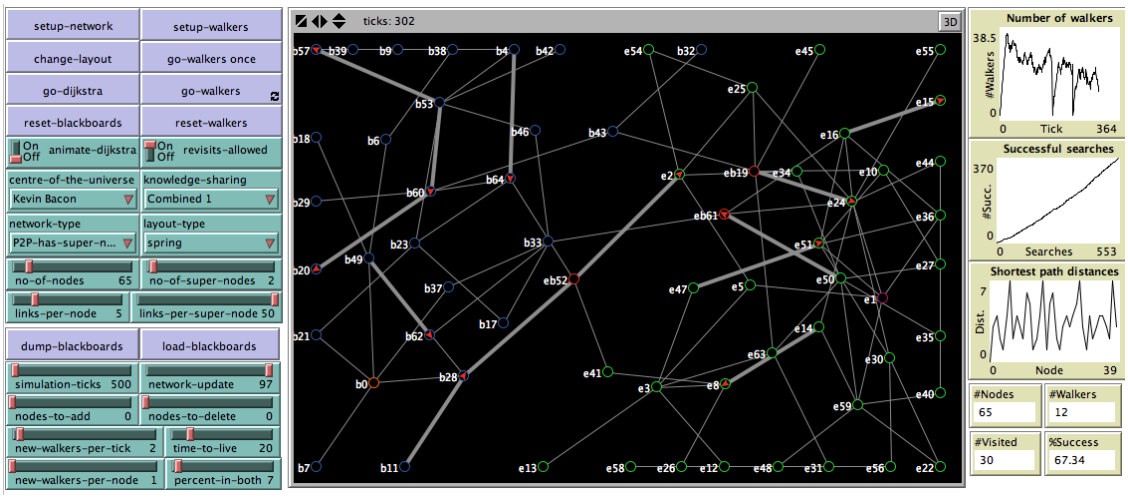


Figure 7.3 Screenshot of the Being Kevin Bacon model.

Download free eBooks at bookboon.com

We now have the background to look at the Being Kevin Bacon NetLogo model. A screenshot of the model is shown in Figure 7.3. Shown in the NetLogo environment is a randomly generated network. This is used to simulate the effectiveness of different communication methods at helping to spread a message throughout the network. The network in this case contains 65 nodes, two of them being super-nodes. Normal nodes have a random number of connections from one up to 5, and super-nodes have a random number of connections up to 50.

Code used to generate the network is shown in NetLogo Code 7.3. Three types of nodes are generated – nodes that belong to the Paul Erdös set (these are coloured lime), nodes that belong to the Kevin Bacon set (coloured blue) and those that belong to both (coloured red). The slider `percent-in-both` is used to determine the chance a node will be randomly generated to belong to both sets; otherwise a node will be randomly chosen to belong to either of the Paul Erdös set or the Kevin Bacon set. The labels of the nodes are shown next to the nodes in the environment with their agent who number prefixed by "e" if they are in the Paul Erdös set, "b" if they are in the Kevin Bacon set and "eb" if they are in both. The node associated with Paul Erdös is chosen to be the node from the Paul Erdös set with the minimum who number (this is coloured magenta and labelled "e1" in the middle right of the environment), and likewise for the Kevin Bacon node (coloured orange and labelled "b0" close to the bottom on the left).

Click on the ad to read more

Download free eBooks at bookboon.com

```
to create-network [number-of-nodes]
;; Creates number-of-nodes new nodes in the network.
   create-nodes number-of-nodes
   [
      set blackboard table:make ;; used by walkers if applying the blackboard
                                ;; knowledge sharing method
      ifelse (random 100 < percent-in-both)
      [ set link-type 0 ;; both erdos and bacon node
      set color red
      set label (word "eb" who " ") ]
      ;else
      [ ifelse (random 2 = 0)
           [ set link-type 1 ;; erdos node
             set color lime
             set label (word "e" who " ") ]
           [ set link-type 2 ;; bacon-node
             set color blue
             set label (word "b" who " ") ]]
   ]
   set erdos-set nodes with [link-type = 1 or link-type = 0] ;; erdos nodes
   set bacon-set nodes with [link-type = 2 or link-type = 0] ;; bacon nodes
end

to setup-network
   clear-all
   set-default-shape nodes "circle 2"
   set-default-shape dwalkers "person"
   ;; create a random network

   reset-counts

   set max-distance 999999
   ;; this number must be greater than maximum path length in the network

   if (network-update > 0)
   [ set network-update-count random network-update ]

   create-network no-of-nodes

   set paul-erdos min-one-of erdos-set [ who ]
   set kevin-bacon min-one-of bacon-set [ who ]
   ask paul-erdos [ set color magenta set paul-erdos-no who ]
   ask kevin-bacon [ set color orange set kevin-bacon-no who ]

   create-network-links

   reset-layout
end
```

NetLogo Code 7.3 Code to create the network for the Being Kevin Bacon model.

Download free eBooks at bookboon.com

Once we have created the network, we can get walker agents to 'crawl' through it performing various tasks. One task might be to find out the average degree of separation of nodes in the network from a particular node; in the model, this node is designated by the slider `centre-of-the-universe` (in the example shown in Figure 7.3, this is set to the Kevin Bacon node). A well-known algorithm for performing this calculation is Dijkstra's algorithm. The algorithm has been implemented in the Being Kevin Bacon model – the code is listed in NetLogo Code 7.4.

Click on the ad to read more

Download free eBooks at bookboon.com

```
to initialise-distances [initial-node]
;; initialises the shortest path distances between network nodes to 0
   let number-of-nodes max [who] of nodes + 1

   set dijkstra-distances array:from-list n-values number-of-nodes
      [number-of-nodes + 1] ;; "infinity"
   set dijkstra-directions array:from-list n-values number-of-nodes [nobody]
   array:set dijkstra-distances initial-node 0
   ;; (sets distance to 0 for initial node)
end

to perform-dijkstra [initial-node nodes-to-visit]
;; calculates the distance array for the Dijkstra algorithm using
;; the initial node as the focal point

   set nodes-visited 0

   initialise-distances [who] of initial-node

   let visited-set [] ;; this is the list of nodes that have been visited
   let unvisited-set nodes-to-visit
   ;; this is the list of nodes yet to have been visited

   let this-dwalker nobody
   if (animate-dijkstra)

      [ create-dwalkers 1
      [ set color white
      set size 2
      set this-dwalker who ]]

   let current-node initial-node
   while [count unvisited-set > 0]
   [
      if (animate-dijkstra)
      [ ask dwalker this-dwalker
           [ setxy [xcor] of current-node [ycor] of current-node
           display
           wait 0.1 ]]
      ask current-node
      [
      set nodes-visited nodes-visited + 1
      set visited-set fput who visited-set
      set unvisited-set other unvisited-set
      ask link-neighbors
      [
           let dist-thru-here
              (array:item dijkstra-distances [who] of current-node) + 1
           let dist-thru-to-there array:item dijkstra-distances who
           if (dist-thru-here < dist-thru-to-there)
              [ array:set dijkstra-distances who dist-thru-here
              array:set dijkstra-directions who [who] of current-node ]
      ]
```


Download free eBooks at bookboon.com

```
      ;; set the current-node to the remaining unvisited node that has
      ;; the smallest distance to the initial node
      set current-node min-one-of unvisited-set
              [array:item dijkstra-distances who]
      ]
   ]
   ;;
print array:to-list dijkstra-distances
   if (animate-dijkstra)
      [ wait 0.2 ask dwalker this-dwalker [ die ]]
end

to go-dijkstra
;; Run directly by the interface button - calculates the distance array
;; for the Dijkstra algorithm

   let initial-node nobody
   ifelse (centre-of-the-universe = "Paul Erdos")
      [ set initial-node paul-erdos ]
      ;; set the initial node to the node associated with Paul Erdos
      [ set initial-node kevin-bacon ]
      ;; set the initial node to the node associated with Kevin Bacon

   initialise-distances [who] of initial-node

   let nodes-to-visit []
   ;; (this is the list of nodes yet to have been visited)
   ifelse (centre-of-the-universe = "Paul Erdos")
      [ set nodes-to-visit erdos-set ]
      [ set nodes-to-visit bacon-set ]
   perform-dijkstra initial-node nodes-to-visit
   dump-dijkstra-directions

   plot-dijkstra
end
```

NetLogo Code 7.4 Code to perform the Dijkstra algorithm in the Being Kevin Bacon model.

The algorithm works by having walker agents crawl the network updating shortest path distances as they proceed. These distances are stored in an array data structure that is an extension to NetLogo. The procedure perform-dijkstra performs the calculations that are returned in the distances array dijkstra-distances. This array has a distance count associated with each node (indexed by node who number) that is incrementally updated by the walker agents as they crawl through the network. The algorithm maintains two sets – the set of nodes it has not yet visited, and the set of nodes it has. The initial node first visited is chosen to be the node associated with the target node (i.e. the node associated with the centre-of-the-universe or Kevin Bacon in this example). For each node being visited, it is first added to the already visited set so it will not be visited again, then the algorithm updates the distance counts for all its link neighbours where the distance to the target node would be shorter through the current node. The current node is then set to the node in the unvisited set that has the smallest distance to the initial target node.

Download free eBooks at bookboon.com

In the Being Kevin Bacon model, the Dijkstra algorithm is executed by clicking on the go-djijkstra button in the interface. Upon completion it will then pop-up a dialog that says what the average shortest path distance is (for the network shown in Figure 7.3 this was 3.41), then it will graph the shortest path distances for each node in the bottom plot shown in the bottom right of the figure.

7.6 Using communicating agents for searching networks

Another task we can get the walker agents to perform is to simulate network searching such as that which occurs in real-life human social networks or computer peer-to-peer networks. This is where multiple agents are searching the network simultaneously without any prior knowledge in order to find different goal nodes from different starting points in the network. For example, in peer-to-peer networks, a problem called 'resource discovery' occurs when a user needs to find a particular resource within the peer-to-peer network. The resource may be a particular file he or she wishes to download or a computer with the necessary compute power or memory that is required to perform execution of a program remotely.

Click on the ad to read more

Download free eBooks at bookboon.com

The simulation as implemented by the Being Kevin Bacon model works by creating walker agents who walk around the network with the task of trying to reach a particular goal node chosen at random. Knowledge concerning the location of the nodes in the network is relayed using different methods of communication – the model implements five: a method called `none` where the agents do not communicate anything when they cross each other; a method called `word-of-mouth`, which is analogous to the word of mouth communicating behaviour described above in Table 7.2; a method called `blackboard` which is analogous to the blackboard based communicating behaviour listed at the bottom of this table (which makes use of stigmergy as discussed in Section 6.5); and two methods (`combined 0` and `combined 1`) that combine the word of mouth and blackboard methods of communication.

The Blackboard method is based on an AI architecture of the same name where agents share and update a common knowledge base. The blackboard architecture is analogous to how blackboards are used in classrooms and during lectures. The blackboard serves as the medium through which the ideas are being communicated. In human communication, those in the classroom or lecture theatre have the choice to read at their leisure (or ignore) what has been written on the blackboard. The variation of the blackboard architecture used in the simulation is that rather than using a single blackboard, each node has its own blackboard. Any walker agent passing by can choose to read the information stored in the node's blackboard or they can choose to ignore it; however, in the model, the walker agents always choose to read and update the nodes' blackboards. The information that is stored is the distance it takes to get to another node in the network by choosing to follow a particular path from the current node. This is updated when a walker returns along the path to the node it started from. No other information is stored in the blackboard – for example, the full path of how to get to the goal node is not stored – just the information of how long it took if a particular path was chosen. If a later walker finds a shorter route via a different path, then the smaller distance value gets written into the blackboard. If there is no prior information in the blackboard, then the walker agents will make a random choice about where to go next.

The code for updating the blackboards is shown in NetLogo Code 7.5. The procedure `update-along-path` updates the blackboards along the path of the walker `this-walker` if the path to the goal is shorter.

Download free eBooks at bookboon.com

```
to update-along-path [this-walker value]
;; updates information along the path when the knowledge sharing method
;; is "Blackboard" or combined

   let p 0
   let val 0
   let val1 0
   let key-dist ""
   let key-path ""

   if knowledge-sharing = "Blackboard" or
      knowledge-sharing = "Combined 1" or
      knowledge-sharing = "Combined 2"
   [
      ask this-walker
   [
      ;;type "goal = " type goal type " path = " show path
      let this-goal goal
      let prev first path
      foreach but-first path
      [
      set key-dist (word this-goal " dist")
      set key-path (word this-goal " path")

      ifelse (value = "F")
           [ set val max-distance ] ;; failure - node was not found
           [ set val p + 1 ] ;; calculate the position in the path

      if is-turtle? ? ;; does the node exist? (it may have been deleted)
      [
           ask ?
           ;; update blackboards along the path if path to goal is shorter
           [ ifelse (not table:has-key? blackboard key-dist)
              [ set val1 max-distance + 1 ]
              ;; (no entry exists - make sure new entry is added)
              [ set val1 table:get blackboard key-dist ]
           if (val < val1)
              [ table:put blackboard key-dist val
                table:put blackboard key-path prev ]]
           ]
           set prev ?
           set p p + 1
      ]
      ]
   ]
end
```

NetLogo Code 7.5 The code to update the blackboards for the Being Kevin Bacon model.

Download free eBooks at bookboon.com

The code uses the table extension to NetLogo where values associated with lookup keys can be stored and retrieved in a hash table. There are two types of keys used for the blackboard, the key to store the path choice and the key to store its distance. The `dump-blackboards` button can be clicked in the Interface to dump out the contents of all the nodes' blackboards. A dump of a blackboard with entries for two destination nodes might look as follows:

```
[[(node 59) dist 8] [(node 59) path (node 11)]
[(node 52) dist 6] [(node 52) path (node 11)]]
```

The key string comprises first a list containing the string "node" followed by the who number of the destination node that can be found by following the link specified by the path key. In the example, the two destination nodes are nodes 59 and 52. The former can be reached along a path with distance 8 by following the link to node 11, the latter takes 6 steps if the same link to node 11 is followed.

The model's simulation is started by clicking on the `go-walkers` button; a single iteration of the simulation can be executed by clicking on the `go-walkers once` button. Repeated simulations show that the Blackboard method of communication as implemented by the simulation consistently outperforms the word-of-mouth method of communication. This reflects experimental results of simulations for the resource discovery problem in peer-to-peer networks (Al Dmour and Teahan, 2004).

During the NetLogo model's simulation, the path that the walkers are taking is shown by increasing the width of the link being crossed over as shown by the fatter white lines in Figure 7.3. On the right of the figure, there are some statistics that have been gathered while the simulation has been running. At the bottom right there are four monitors that display the number of nodes (65 in the figure), the current number of walker agents (12), the number of nodes the walkers have visited (30) and the percentage success the walkers have had at finding the goal node (67.34%). The plot in the top right of the figure graphs the number of walkers versus tick. The walker agents keep searching until either they find the goal node or they exceed the `time-to-live` variable (set at 20 ticks in this case). The graph reflects that initially the number of walkers grows until the `time-to-live` variable starts being exceeded by the walker agents and then continues to drop as more and more walker agents start finding the goal node more regularly as the knowledge contained in the blackboards and/or being transmitted via word of mouth becomes more effective at reducing the search for new agents. The middle plot graphs the percentage of successful searches versus total number of searches; a slight increase in the slope indicates the learning effect as more knowledge is stored or transmitted. The bottom plot is output produced by the Dijkstra algorithm as discussed above.

The code that gets executed when the `go-walkers` button is clicked is shown in NetLogo Code 7.6.

Download free eBooks at bookboon.com

```
to go-walkers
;; make the walkers continue the search
   if (simulation-ticks > 0) and (ticks > simulation-ticks)
      [ stop ]

   update-network

   setup-walkers
   ask links [ set thickness 0 ]
   ask walkers
   [
      set nodes-visited nodes-visited + 1
      ifelse (ticks-alive >= time-to-live)
      ;; lived too long without success - kill it off
      [ set unsuccessful-searches unsuccessful-searches + 1
      update-along-path self "F"
      ;; update information along path if necessary; "F" means failure
      ]
      [ ;; create new walkers to move to new location(s)
      expand-walkers (next-locations location)
      ]
      die ;; always die - the minions have already been sent out to
             ;; continue the search for me
      ]
      plot-walkers
      tick
end
to expand-walkers [new-locations-list]
   let new-walkers-count length new-locations-list
   let new-location 0

   hatch-walkers new-walkers-count
   [ ;; create a new walker at location to continue the search
      ;; and move to new location
      set ticks-alive ticks-alive + 1
      ;; pop off the next location to visit:
      set new-location first new-locations-list
      set new-locations-list but-first new-locations-list
      ifelse (not revisits-allowed and member? new-location path)
      [ die ] ;; don't revisit nodes already visited unless allowed to
      [
      if (new-location != nobody) and
      ([link-with new-location] of location != nobody)
      [
      ;; highlight link I am crossing
      ask [link-with new-location] of location [ set thickness 0.4 ]
      face new-location
      move-to new-location
      set location new-location
      set path fput new-location path
```


Download free eBooks at bookboon.com

```
      if new-location = goal ;; success-no need to hang around any longer
      [ set successful-searches successful-searches + 1
           ;; update information along path if necessary; "S" means success
           update-along-path self "S"
           die ] ;; die happy
      ]
      ]
   ]
end
```

NetLogo Code 7.6 A selection of the code executed by the walker agents each tick for the Being Kevin Bacon model.

The `go-walkers` procedure first updates the network by adding or deleting random nodes in the network if the network is dynamic. This is to simulate networks where the nodes are continually changing; the rate of change is controllable by the `network-update`, `nodes-to-add` and `nodes-to-delete` sliders in the model's interface. If `network-update` is set to 0, then the network becomes static with no new nodes added or deleted each tick.

Click on the ad to read more

Download free eBooks at bookboon.com

The `go-walkers` procedure then resets all links' line thickness to 0, and then asks each currently active walker agent whether they have been taking too long (i.e. `time-to-live` has been exceeded). If it has, the blackboards along the path the walker has visited are updated by writing a large number `max-distance` to reflect that the path has been a failure (this number is initialised in NetLogo Code 7.3, and must be a number much greater than the maximum length of any path in the network). Otherwise the search is expanded by calling the `expand-walkers` procedure that hatches new walkers to move to new locations in the network, and updates the blackboards if the search path has been successful. Then the current walker is allowed to die since it has already delegated the remainder of the search to the walkers created by the `expand-walkers` procedure.

We will now look at a probabilistic approach to characterising communication based on information theory, to help us further explore the nature of communication and language.

7.7 Entropy and Information

In information theory, *entropy* provides a means for measuring the information transmitted between communicating agents down a communication channel. Entropy is the average number of bits required to communicate a message between two agents down a communication channel.

The fundamental coding theorem (Shannon, 1948) states that the lower bound to the average number of bits per symbol needed to encode a message (i.e. a sequence of symbols such as text) sent down a communication channel is given by its *entropy*:

$$H(P) = -\sum_{i=1}^{k} p(x_i)\log(x_i)$$

where there are k possible symbols with probability distribution and where the probabilities are independent and sum to 1. If the base of the logarithm is 2, then the unit of measurement is given in bits. For example, suppose that an alphabet contains five symbols *White*, *Black*, *Red*, *Blue* and *Pink* with probabilities $\frac{2}{7}, \frac{2}{7}, \frac{1}{7}, \frac{1}{7}$ and $\frac{1}{7}$. Then the average number of bits required to encode each symbol is given by:

$$\begin{aligned} H &= -p(White) - p(Black) - p(Red) - p(Blue) - p(Pink) \\ &= -\tfrac{2}{7}\log_2\left(\tfrac{2}{7}\right) - \tfrac{2}{7}\log_2\left(\tfrac{2}{7}\right) - \tfrac{1}{7}\log_2\left(\tfrac{1}{7}\right) - \tfrac{1}{7}\log_2\left(\tfrac{1}{7}\right) - \tfrac{1}{7}\log_2\left(\tfrac{1}{7}\right) \\ &= 2.24 \text{ bits.} \end{aligned}$$

The entropy is a measure of how much uncertainty is involved in the selection of a symbol – the greater the entropy, the greater the uncertainty. It can also be considered a measure of the "information content" of the message – more probable messages convey less information than less probable ones.

Download free eBooks at bookboon.com

Entropy can be directly related to compression. The entropy is a lower bound on the average number of bits per symbol required to encode a long string of text drawn from a particular source language (Brown *et al*. 1992, page 32). Bell, Cleary & Witten (1990) show that arithmetic coding, a method of assigning codes to symbols with a known probability distribution, achieves an average code length arbitrarily close to the entropy. Hence, compression can be used directly to estimate an upper bound to the entropy in the following manner. Given a sequence of *n* symbols $x_1, x_2 \ldots x_n$, the entropy can be estimated by summing the *code lengths* required to encode each symbol:

$$H = \sum_{i=1}^{n} -\log_2 p(x_i).$$

Here the code length for each symbol x_i is calculated by using the formula $-\log_2 p(x_i)$.

The entropy calculated in this manner is relevant as it provides a measure of how well the statistical model is doing compared to other models. This is done by computing the entropy for each statistical model, and the model with the smallest entropy is inferred to be the "best".

7.8 Calculating Entropy in NetLogo

A Cars Guessing Game model in NetLogo has been created to illustrate how entropy and compression code lengths are calculated. A screenshot of the model's interface is shown in Figure 7.4. The model represents a variation of a game that people can play while travelling in a car. People select a colour in advance – usually white, black, red or blue – and then they count how many cars with the selected colour drive past them in the opposite direction. The person who has the highest count at the end of the game wins. The variation of the game covered in the model is that instead of each person selecting a single car, they specify counts of each colour in advance in order to define a probability distribution. Their performance is measured by encoding the arrival of the cars (seen in the animation of Figure 7.4) against their own distribution. The person with the smallest total code length (the sum of the log of the probabilities of the colours of the cars that appear) is the one who wins the game.

In the figure, there are probability distributions for three agents to simulate three different people playing the game. These are defined using the sliders on the left that are used to alter the counts for each colour in the distribution, and the counts are used to calculate the probabilities by dividing by the total count. Additionally, there are counts that define the source distribution that is used to generate the colour of the cars that appear in the animation. A fourth computer agent is also playing the game by adaptively updating the counts of each colour as the cars appear. These counts are shown in the white boxes under the note "Adaptive Agent's distribution" in the middle of the figure. As the source distribution is equiprobable in this case – each of the colours have the same count of 1 – then this is reflected in the adaptive counts which are very similar, either 214 or 215. That is, there have been either 214 or 215 cars with the observed colour that have been generated by the model since the animation was started.

Download free eBooks at bookboon.com

The entropy and code length calculations are shown in the right boxes in Figure 7.4. The entropy for all the distributions remains the same except for the adaptive distribution until a slider is used to change a count on the left, and then the entropy is immediately recalculated to reflect the change in the probability distribution. The entropy for the adaptive distribution changes constantly to reflect that the adaptive counts are continually being updated. The code lengths on the right of the figure are the cumulative sums of the cost of encoding the information of the colour of the actual car according to each agent's distribution.

Usually the source code length total is smaller than the rest as it reflects the actual distribution that is being used to generate the appearance of the cars. The adaptive code length is also usually close to but slightly greater than the source code length, which is indicative of the usefulness of the strategy of matching your distribution to the actual observations, rather than using a pre-defined set of probabilities in advance. In some circumstances, however, due to randomness used to generate the sequence of cars that appear, at various times the actual distribution does not exactly match the source distribution, in which case the adaptive strategy can outperform the source predictions at guessing the colour of the cars.

Click on the ad to read more

Download free eBooks at bookboon.com

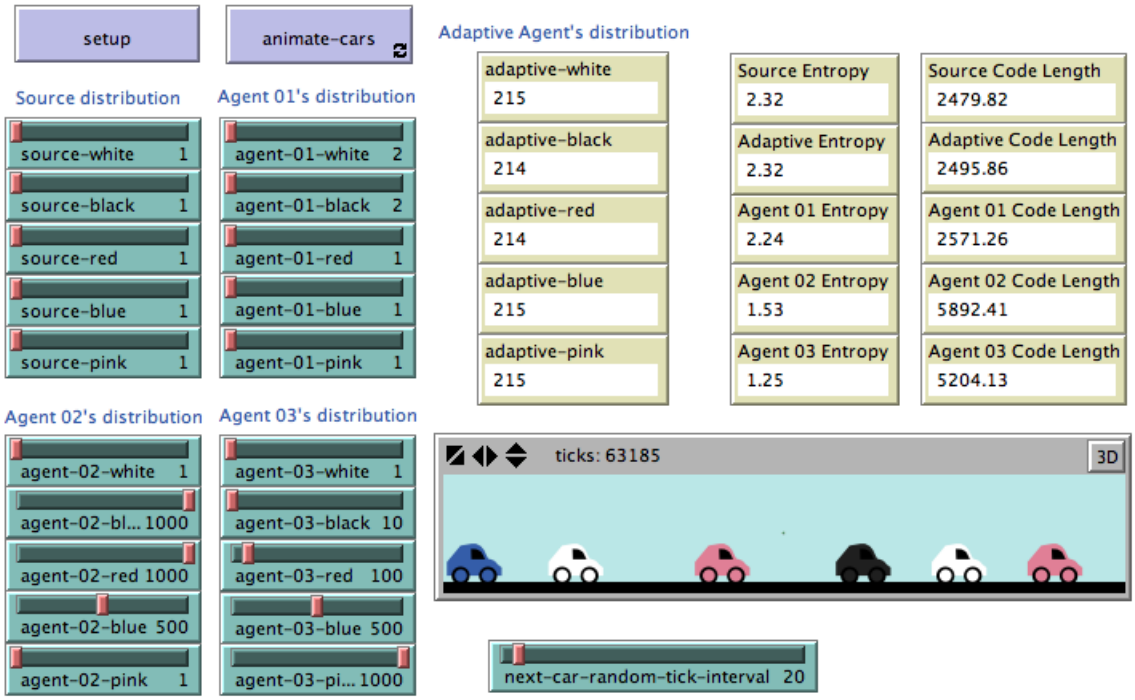


Figure 7.4 A screenshot of the Cars Guessing Game model.

For the three agents that use a semi-fixed strategy (i.e. the user can use the sliders to change the counts on the fly, so the distribution is fixed only until a count is next changed), the code lengths reflect how similar the counts are to the source counts. If there is a variation, then this will result in a larger encoding cost. For example, in the figure, Agent 01's counts are closest to the source counts, with only the white and black counts being different. This results in a slight decrease in entropy – from 2.32 down to 2.24 as the entropy indicates the average cost of encoding the colour of a single car, and some colours are now expected to occur more frequently than others. However, this is not reflected in the observations as the probability distributions now do not match, and therefore there is an increase in the overall code length as a result (from 2479.82 to 2571.26). When the counts are significantly mismatched as for agent 02 and 03, then the code length totals are much greater than the source code length as a result.

Selected parts of the code for the model have been listed in NetLogo Code 7.7 and NetLogo Code 7.8. (The full code can be downloaded by using the URL link shown at the bottom of this chapter). The first code listing includes the `setup-distribution` procedure where the distributions are set based on the slider counts. The distributions are represented as a list of five numbers – the white, black, red, blue and pink counts, in that order. The adaptive distribution is initialised so that all counts are 1 at the beginning. At the end of the setup procedure, the total count `dist-total` and entropy `dist-entropy` is calculated for the distribution. The procedure `neg-log-prob` calculates the negative of the log of the probability for each colour using the count and total counts, and this is used to calculate the entropy for the distribution.

Download free eBooks at bookboon.com

```
to setup-distributions
;; sets the distributions for each agent
   ask agents with [agent-id = 0]
   [ set distribution
      (list source-white source-black source-red source-blue source-pink) ]
   ask agents with [agent-id = 1]
   [ set distribution
      (list agent-01-white agent-01-black agent-01-red agent-01-blue
            agent-01-pink) ]
   ask agents with [agent-id = 2]
   [ set distribution
      (list agent-02-white agent-02-black agent-02-red agent-02-blue
               agent-02-pink) ]
ask agents with [agent-id = 3]
[ set distribution
   (list agent-03-white agent-03-black agent-03-red agent-03-blue
      agent-03-pink) ]
if (ticks = 0)
[ ask agents with [agent-id = 4] ; initialise the adaptive distribution
   [ set distribution (list 1 1 1 1 1) ]]
;; all colours are equiprobable at the beginning for the adaptive agent
;; the adaptive agent's distribution counts are updated subsequently
;; elsewhere in update-adaptive-count

ask agents
[ set dist-total 0
   set dist-entropy 0
   foreach distribution
   [ set dist-total dist-total + ?] ; calculate total first
   foreach distribution
   [ set dist-entropy dist-entropy + neg-log-prob ? dist-total]]
end

to-report neg-log-prob [p q]
;; returns the negative of the log to base 2 of the probability p/q.
   report (- log (p / q) 2)
end
```

NetLogo Code 7.7: Code to set up each agent's distribution and calculate its entropy.

The code length calculation is performed by the `encode-this`-car procedure in NetLogo Code 7.8. The total code length to date, `codelength-total`, is updated by adding the negative log of the probability for the colour of the car being encoded. The probability is the count of the colour stored in the agent's `distribution` list divided by the total count, `dist-total`.

Download free eBooks at bookboon.com

```
to encode-this-car [colour]
;; returns the cost of encoding this car's colour according to the
;; agent's distribution
   let codelength 0
   ask agents
   [
      set codelength 0
      ifelse colour = white
      [set codelength neg-log-prob (item 0 distribution) dist-total]
      [ifelse colour = 1 ;almost black to make the car doors & windows visible
      [set codelength neg-log-prob (item 1 distribution) dist-total]
      [ifelse colour = red
      [set codelength neg-log-prob (item 2 distribution) dist-total]
      [ifelse colour = blue
      [set codelength neg-log-prob (item 3 distribution) dist-total]
      [set codelength neg-log-prob (item 4 distribution) dist-total]]]]
      set codelength-total codelength-total + codelength
   ]
end
```

NetLogo Code 7.8: Code to encode the colour of a car according to an agent's distribution.

Click on the ad to read more

Download free eBooks at bookboon.com

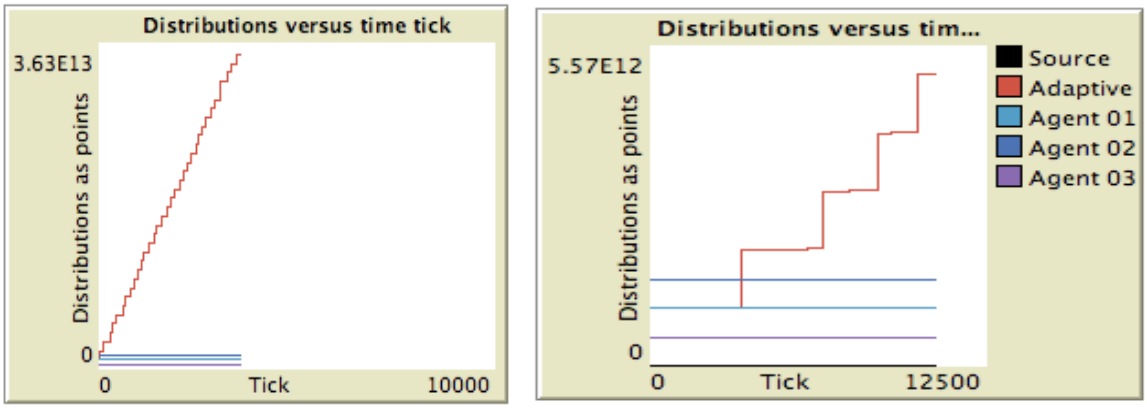


Figure 7.5 Screenshots of the plot produced by the Cars Guessing Game model showing how the distributions vary with each tick.

We can represent the internal state of the agents in this simulation by their distributions. If we recast the list of distribution counts as a single number by using the reporter procedure listed in NetLogo Code 7.9, then we can plot how each distribution varies versus time. The purpose of this is to illustrate how probability distributions can be depicted as single points in an *n*-dimensional space, and how their change over time represents movement in that space (note that this uses a similar parallel co-ordinates approach to that shown in the right plot of Figure 2.5 mentioned in Chapter 2). Figure 7.5 shows two screenshots of the plots that the model produced for two configurations – one where the source counts were all 1's (the left plot) and one where the source counts were all 1000's (the right plot). Only the adaptive distribution is varying and the red lines in the plots reflect this. As the adaptive counts are continually increasing, then we have to make sure that a maximum number is never exceeded, so this is done using the `remainder` reporter in the code. The other distributions are static, and their lines are horizontal as a result. If the user were to change a count for one of these distributions, the plots would reflect the change by an alteration in the position of the relevant line.

```
to-report distribution-as-a-point [agent]
; return the agent's distribution represented as a single point

   report ;; note that the maximum count for non-adaptive distributions is
          ;; 1000 so make sure that max. count does not exceed 1000 for
          ;; the adaptive distribution
   (remainder (item 0 [distribution] of agent) 1000) +
   (remainder (item 1 [distribution] of agent) 1000) * 1000 +
   (remainder (item 2 [distribution] of agent) 1000) * 1000 * 1000 +
   (remainder (item 3 [distribution] of agent) 1000) * 1000 * 1000 * 1000 +
   (remainder (item 4 [distribution] of agent) 1000) * 1000 * 1000 * 1000 *
             1000
end
```

NetLogo Code 7.9 Representing the agent's distributions as single numbers.

Download free eBooks at bookboon.com

This model shows how agents can compare the quality of different statistical models. These models can help to predict likely future events in order to make informed decisions. It is important to note that unless you have some knowledge about the process producing future occurrences, then you can never be certain which statistical model is the 'best' one at predicting what might happen next, especially in a dynamic environment where forces that generate events can change from one moment to the next.

7.9 Language Modelling

The method of making decisions based on which statistical model has been best at predicting the past sequences of events (for example, using compression code length calculations as above) has a wide range of applications for agent decision making. In this chapter, we have been focusing on communication and language, and we can examine some further examples that illustrate this, specifically in the area of language modelling.

In a statistical (or probabilistic) model of language, the assumption is made that the symbols (e.g. words or characters) can be characterized by a set of conditional probabilities. A *language model* is a computer mechanism for determining these conditional probabilities. It assigns a probability to all possible sequences of symbols.

The most successful types of language models for sequences of symbols that occur in natural languages such as English and Welsh are word *n*-gram models (which base the probability of a word on the preceding *n* words) and part-of-speech *n*-gram models (which base the probability on the preceding words and parts of speech; these are also called *n*-pos models). Character *n*-gram models (models based on characters) have also been tried, although they do not feature as prominently in the literature as the other two classes of model. The probabilities for the models are estimated by collecting frequency statistics from a large corpus of text, called the *training text*, in a process called *training*. The size of the training text is usually very large containing many millions (and in some cases billions) of words.

These models are often referred to as "Markov models" because they are based on the assumption that language is a Markov source. An *n*-gram model is called an order $n - 1$ Markov model (for example, a trigram model is an order 2 Markov model). In an *n*-gram model, the probabilities are conditioned on the previous words in the text. Formally, the probability of a sequence *S*, of *n* words, $w_1, w_2 \ldots w_n$ is given by:

$$\begin{aligned} p(S) &= p(w_1)p(w_2|w_1)p(w_3|w_1, w_2)\ldots p(w_n|w_1, w_2 \ldots w_n) \\ &= \prod_{i=1}^{n} p(w_i|w_1, w_2 \ldots w_{i-1}) \end{aligned}$$

Download free eBooks at bookboon.com

Here, w_i is called the *prediction* and $w_1, w_2, \ldots w_{i-1}$ the *history*. Building a language model that uses the full history can be computationally expensive. *n*-gram language models make the assumption that the history is equivalent to the previous $n - 1$ words (called the conditioning context). For example, bigram models make the following approximation:

$$p(S) = \prod_{i=1}^{n} p(w_i | w_{i-1}).$$

In other words, only the previous word is used to condition the probability. Trigram models condition the probability on the two previous words:

$$p(S) = \prod_{i=1}^{n} p(w_i | w_{i-1}, w_{i-2}).$$

The assumptions which these approximations are based upon are called "Markov assumptions." For example, the bigram model makes the following Markov assumption:

$$p(w_i | w_1, w_2 \ldots w_{i-1}) = p(w_i | w_{i-1}).$$

American online
LIGS University
is currently enrolling in the
Interactive Online BBA, MBA, MSc,
DBA and PhD programs:

- enroll by September 30th, 2014 and
- save up to 16% on the tuition!
- pay in 10 installments / 2 years
- Interactive Online education
- visit www.ligsuniversity.com to find out more!

Note: LIGS University is not accredited by any nationally recognized accrediting agency listed by the US Secretary of Education.
More info here.

Click on the ad to read more

Download free eBooks at bookboon.com

It might seem that such drastic assumptions would adversely affect the performance of the statistical models, but in practice, bigram and trigram models have been applied successfully to a wide range of domains especially machine translation and speech recognition. Kuhn and De Mori (1990, page 572) point out why this approach is so effective: "The novelty of [this] approach is that it considers it more important to keep the information contained by the last few words than to concentrate on syntax, which by definition involves the whole sentence. A high percentage of English speech and writing consists of stock phrases that reappear again and again; if someone is halfway through one of them, we know with near-certainty what his next few words will be". The decision as to which assumptions yield better performance is an empirical issue rather than a theoretical one.

7.10 Entropy of a Language

The amount of information per word in some corpus of n words is given by:

$$H(S,M) = -\frac{1}{n}\log p_M(S).$$

This can be used to estimate the entropy of a language where a language model M as described in the previous section is used to estimate the probabilities, and where the larger the number of words n in the corpus, the better the estimate is. (Technically, $H(S, M)$ is referred to as the *cross-entropy* of the language as it is an estimate based on a model M rather than the true entropy.)

In a classic paper titled "Prediction and entropy of printed English" published in 1951, Shannon estimated the entropy of English to be about 1 bit per character (Shannon, 1951). Shannon's method of estimating the entropy was to have human subjects guess upcoming characters based on the immediately preceding text. He chose 100 random samples taken from Dumas Malone's *Jefferson the Virginian* (Malone 1948). From the number of guesses made by each subject, Shannon derived upper and lower bound estimates of 1.3 and 0.6 bits per character ("bpc" for short). Cover & King (1978) noted that Shannon's guessing procedure gave only partial information about the probabilities for the upcoming symbol. A correct guess only tells us which symbol a subject believes is the most probable, and not how much more probable it is than other symbols. They developed a gambling approach where each subject gambled a proportion of their current capital on the next symbol. Using a weighted average over all the subjects' betting schemes they were able to derive an upper bound of 1.25 bpc. The performance of individual subjects ranged from 1.29 bpc to 1.90 bpc.

Cover and King discuss the meaning of the phrase 'the entropy of English': "It should be realized that English is generated by many sources, and each source has its own characteristic entropy. The operational meaning of entropy is clear. It is the minimum expected number of bits/symbol necessary for the characterization of the text." Both Shannon's and Cover and King's approaches were based on human subjects guessing a text. Other approaches have been based on the statistical analysis of character and word frequencies derived from text, and on the performance of computer models using compression algorithms.

Download free eBooks at bookboon.com

The Shannon's Guessing Game model has been created in NetLogo to illustrate how this works. The model allows the user to first load in a training text to prime its statistical model, and then choose a testing text to play Shannon's Guessing Game on. This is done by first selecting the max depth of the tree and maximum size of text to train on using the `max-tree-depth` and `max-text-size` sliders, then using the `which-text` chooser on the mid-left of the NetLogo application to select the training or testing text, then clicking on the `load-training-text` or `load-testing-text` buttons as appropriate. The user then makes the application start playing Shannon's Guessing Game by clicking on the `predict-text` button. The application then repeatedly predicts each upcoming character one at a time.

For example, we can first choose to load Jane Austen's *Pride and Prejudice* as the training text. In this case, the training text is the following characters taken from the beginning of Jane Austen's book: "*It is a truth universally acknowledged, that a single man in possession of a good fortune, must be in want of a wife. However little known the feelings or views of such a man may be on his first entering a neighbourhood, this truth is so well fixed in the minds of the surrounding families, that he is considered the rightful property of some one or other of their daughters.*"

If we then choose Jane Austen's *Sense and Sensibility* as the testing text, the NetLogo application will predict the upcoming characters one by one using the following sequence taken from the beginning of this book: "*The family of Dashwood had long been settled in Sussex. Their estate was large, and their residence was at Norland Park, in the centre of their property, where, for many generations, they had lived in so respectable a manner as to engage the general good opinion of their surrounding acquaintance. The late owner of this estate was a single man, who lived to a very advanced age, and who for many years of his life, had a constant companion and housekeeper in his sister.*"

The application lists the predictions for all the characters it has seen before, including their frequency counts and estimated probabilities, according to the current context (i.e. immediate prior text) and consequent depth in the training text's tree model. For example, at the beginning, there is no prior context, so only the predictions at the top of the tree (depth 0) are listed as shown in Figure 7.6. In this example, the maximum depth of the tree has been set at 3. The space character has been encoded first, and then the next character is encoded, the character "T", followed by the character "h", then "e" and so on. The situation after the first character (a space) is processed is shown in the figure. All the predictions are listed at depth 1 and 0 but no deeper than that as we have no further context to base our predictions on. The depth 1 predictions list all the characters that have occurred after the single character context consisting of a space. As space is a frequently occurring character in English text, then there are many predictions – for example, after a space the character "o" has occurred 11 times after a space in the training text, the character "t" has occurred 10 times, the character "a" has occurred 7 times and so on. These predictions have been listed in decreasing frequency order.

Download free eBooks at bookboon.com

The situation that arises in this example is that we need to encode the character "T", but this has never occurred before in the training text. (Look in the training paragraph above shown in italics – the character "T" never follows the space character. This is just a result of the shortness of the training text used in this example; in reality, in a larger training text, there may be many occurrences when "T" follows a space such as at the beginning of a sentence beginning with the word "The".) The problem in this example is that without a prediction in our model we have to assign a zero probability that will result in an *infinite* code length since $\log(0) = \infty$. This problem is called the *zero frequency problem* and in order to overcome this, we need to assign some probability to events that have never occurred before. One method of doing this is called 'escaping' or 'backing off' – we back off to a shorter context where there are usually more predictions. To do this, we assign a small probability for the occasions when we will need to escape – i.e. when we encounter a sequence that we have never seen before. This process is also called 'smoothing' as this tends to make the probability distributions more uniform (hence the reason for its name), with low (including zero) probabilities typically adjusted upwards, and high probabilities adjusted downwards. The purpose of smoothing is twofold: firstly, to ensure that the probability is never zero; and secondly, to hopefully improve the accuracy of the statistical model.

Click on the ad to read more

Download free eBooks at bookboon.com

```
Text so far:
" "

Predictions (at various depths in the training model's tree):
(Each character is listed followed by its frequency of occurrence and
estimated probability)
depth 1:  (types = 22 tokens = 69 escape probability = 22/91)
  "o" [11] (11/91)  "t" [10] (10/91)  "a" [7] (7/91)  "i" [6] (6/91)  "s" [5] (5/91)
  "m" [5] (5/91)  "f" [5] (5/91)  "w" [3] (3/91)  "b" [2] (2/91)  "h" [2] (2/91)
  "p" [2] (2/91)  "e" [1] (1/91)  "u" [1] (1/91)  "l" [1] (1/91)  "H" [1] (1/91)
  "c" [1] (1/91)  "g" [1] (1/91)  "d" [1] (1/91)  "k" [1] (1/91)  "n" [1] (1/91)
  "v" [1] (1/91)  "r" [1] (1/91)
depth 0:  (types = 28 tokens = 372 escape probability = 28/400)
  " " [69] (69/400)  "e" [34] (34/400)  "o" [27] (27/400)  "i" [26] (26/400)  "t" [26] (26/400)
  "n" [23] (23/400)  "s" [22] (22/400)  "h" [18] (18/400)  "r" [18] (18/400)  "a" [16] (16/400)
  "f" [13] (13/400)  "u" [11] (11/400)  "l" [11] (11/400)  "d" [10] (10/400)  "g" [9] (9/400)
  "m" [7] (7/400)  "w" [7] (7/400)  "," [4] (4/400)  "v" [3] (3/400)  "b" [3] (3/400)
  "p" [3] (3/400)  "y" [3] (3/400)  "c" [3] (3/400)  "k" [2] (2/400)  "H" [1] (1/400)
  "x" [1] (1/400)  "I" [1] (1/400)  "." [1] (1/400)

Encodings and entropy for next character in sequence "T":
 22/91 X 28/400 X 1/256
Cross-Entropy for this character = 13.88486428927852 bits
Total Cross-Entropy for all characters so far = 101.4501216723048 bits
```

Figure 7.6 Screenshot of the output produced by the Shannon's Guessing Game model at the beginning of the test sequence.

The method used to estimate the escape probability in the example (as coded in the Shannon's Guessing Game NetLogo model) is to base it on the number of *types* that have occurred in the context. For a given sequence of symbols, the number of types is the number of unique symbols that have occurred, whereas the number of *tokens* is the total number of symbols. For example, in the character sequence "aaaa bbb cc d" there are 5 character types – 'a, 'b', 'c', 'd' and space – as opposed to 13 character tokens; for the palindromic word sequence "a man a plan a canal panama" there are 5 word types – "a", "man", "plan", "canal" and "panama" – and 7 word tokens in total.

In Figure 7.6, for the context consisting of a single space, we can see there are 22 types and 69 tokens. Therefore, the escape probability is estimated to be 22/91 and the probabilities for the other characters have been adjusted accordingly to equal their count divided by 91 rather than 69. Formally, the probability for a symbol s in a particular context C is calculated as follows:

$$p(s|C) = \frac{count(s)}{types(C) + tokens(C)}.$$

This method of smoothing, invented by Alistair Moffat for the PPMC compression scheme, is called Witten-Bell smoothing in the language modelling literature.

Download free eBooks at bookboon.com

In the example, the statistical model has to back off to a default model where all characters are equiprobable in order to be able to encode the character 'T' because in this case, it has never been seen anywhere in the training sequence. Therefore the cost of encoding the character 'T' is $22/91 \times 28/400 \times 1/256$ with the factor 28/400 being for encoding the escape for the null context (i.e. just using the character frequency counts at depth 0 in the tree) and 1/256 for the default model (where 256 is the size of the alphabet). This requires 13.88 bits to encode as $-\log_2(22/91 \times 28/400 \times 1/256) = 13.88$ bits.

Figure 7.7 shows the output after we have encoded the first ten characters in the testing sequence (" The famil"). The next character we need to encode is 'y', and because it has not been seen in the depth 3 context in the training sequence, we need to escape from the context which is at depth 3 in the tree (where there has only been a single previous occurrence so the probability is 1/2), then through depth 2 (where again there has only been a single previous occurrence so the probability is 1/2) to depth 1 (where we have seen the character 'y' before, and its probability is 1/16). This costs 6 bits to encode since $-\log_2(1/2 \times 1/2 \times 1/16) = 6$.

```
Text so far:
" The famil"

Predictions (at various depths in the training model's tree):
(Each character is listed followed by its frequency of occurrence and
estimated probability)
depth 3:  (types = 1 tokens = 1 escape probability = 1/2)
  "i" [1] (1/2)
depth 2:  (types = 1 tokens = 1 escape probability = 1/2)
  "i" [1] (1/2)
depth 1:  (types = 5 tokens = 11 escape probability = 5/16)
  "i" [3] (3/16)  "e" [3] (3/16)  " " [2] (2/16)  "l" [2] (2/16)  "y" [1] (1/16)
depth 0:  (types = 28 tokens = 372 escape probability = 28/400)
  " " [69] (69/400)  "e" [34] (34/400)  "o" [27] (27/400)  "t" [26] (26/400)  "i" [26] (26/400)
  "n" [23] (23/400)  "s" [22] (22/400)  "h" [18] (18/400)  "r" [18] (18/400)  "a" [16] (16/400)
  "f" [13] (13/400)  "u" [11] (11/400)  "l" [11] (11/400)  "d" [10] (10/400)  "g" [9] (9/400)
  "m" [7] (7/400)  "w" [7] (7/400)  "," [4] (4/400)  "y" [3] (3/400)  "p" [3] (3/400)
  "b" [3] (3/400)  "v" [3] (3/400)  "c" [3] (3/400)  "k" [2] (2/400)  "." [1] (1/400)
  "x" [1] (1/400)  "H" [1] (1/400)  "I" [1] (1/400)

Encodings and entropy for next character in sequence "y":
 1/2 X 1/2 X 1/16
Cross-Entropy for this character = 6 bits
Total Cross-Entropy for all characters so far = 139.13727151396256 bits
```

Figure 7.7 Screenshot of the output produced by the Shannon's Guessing Game model after the sequence " The famil" has been processed.

Another Netlogo model called Language Modelling has been developed to help visualise the tree models. Figure 7.8 shows a screenshot of a tree model that has been trained on the 32 characters at the beginning of Jane Austen's *Sense and Sensibility* shown above. For this example, the maximum tree depth was set at 2. The figure shows that most two-character sequences are unique (i.e. outermost counts are all 1's), which is to be expected because of the shortness of the training text, apart from the sequence "d" which occurs twice (this can be found upper left or at direction NNW in the figure). Some of the single characters occur more than once, resulting in a fan-out affect at depth 2 – for example, " " occurs 5 times (see middle right of figure) and is followed by 5 different characters ("f", "o", "D", "h" and "l").

Download free eBooks at bookboon.com

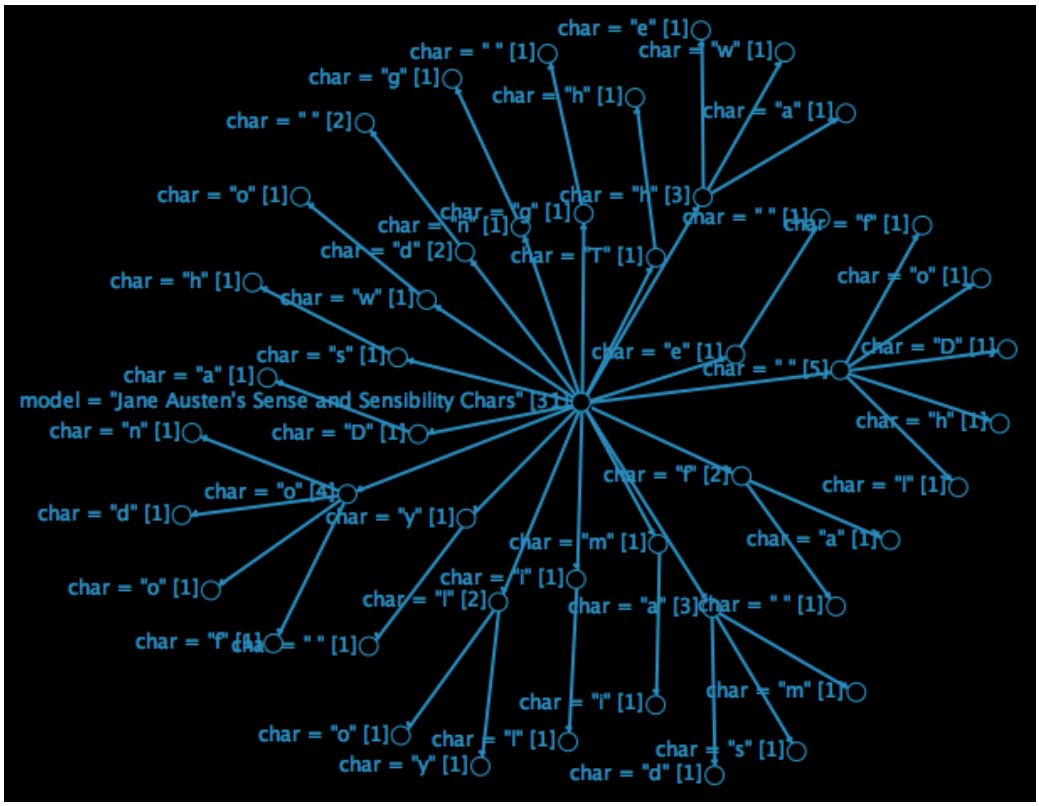


Figure 7.8 Screenshot of the output produced by the Language Modelling model with maximum tree depth of 2 after processing the first 32 characters of Jane Austen's Sense and Sensibility.

Maastricht University Leading in Learning!

Join the best at the Maastricht University School of Business and Economics!

Top master's programmes

- 33rd place Financial Times worldwide ranking: MSc International Business
- 1st place: MSc International Business
- 1st place: MSc Financial Economics
- 2nd place: MSc Management of Learning
- 2nd place: MSc Economics
- 2nd place: MSc Econometrics and Operations Research
- 2nd place: MSc Global Supply Chain Management and Change

Sources: Keuzegids Master ranking 2013; Elsevier 'Beste Studies' ranking 2012; Financial Times Global Masters in Management ranking 2012

Maastricht University is the best specialist university in the Netherlands (Elsevier)

Visit us and find out why we are the best!
Master's Open Day: 22 February 2014

www.mastersopenday.nl

Click on the ad to read more

Download free eBooks at bookboon.com

The Language Modelling application has been developed to handle the modelling of events on different input streams using a similar approach to that used in the Wall Following Events model described in Section 6.6. In the example depicted in the figure, only character events are being modelled. This explains the re-appearance of the string "char" throughout the figure. The model also allows loading of word events as well as character events, a longer size of text to train on and the setting of the maximum tree depth up to 8, although the visualisation of these trees becomes much more problematic due to the constraints of NetLogo's environment and the complexity of the trees that are produced.

7.11 Communicating Meaning

This chapter has focused on agent communication, the different types of communicating behaviour and the important role that agent communication has to play in agent-to-agent interaction. This chapter has had less to say about the content of the messages being communicated between agents. Language is used to define the symbols that convey the meaning behind these messages. The purpose of the communication for the sender is to convey the meaning of the message in such a way that the receiver can decode it and respond accordingly. If the receiver is not able to do this, then the communication has failed and the message has been wasted.

Meaning can be defined as follows. Meaning is the sense or significance behind the message expressed by some language that an agent wishes to convey to another agent. As with all definitions, a closer inspection of this working definition will reveal some inadequacies. One can ask further questions concerning the definition such as "What is the meaning of the word 'message'?" and "What is the meaning of 'sense or significance'?" Continuing this line of questioning results in an infinite regress of definitions where the meaning of anything never seems to be pinned down precisely in the way that we can pin down the meaning of mathematical symbols, for example.

The nature of meaning (and the related study of semantics in linguistics) has been discussed and debated for centuries, and is the subject of many reams of written works and scholarly research. As humans, we can readily understand the meanings of these words, so we could take the pragmatic viewpoint, like Turing did when suggesting the Turing Test for intelligence, that although obtaining precise definitions of the meaning of words and language is an interesting philosophical pursuit, it is not necessarily required as humans can achieve successful communication without it. This is despite humans never being in complete agreement about the precise meanings of the symbols they use to communicate with.

But what if we needed to convey the meaning of a message to someone or something that was not human? This is the subject of Thought Experiment 7.1.

Download free eBooks at bookboon.com

Thought Experiment 7.1 Communicating with aliens.

Let us consider the problem of how to communicate a message with an alien. In this setting, let us assume that there is no common basis for understanding the meaning of the message. That is, we need to communicate the meaning of the symbols in the message as well as the symbols themselves. For example, we might wish to communicate the following message: "*Do not open*". Just such a scenario occurred in the movie *Alien Hunter* but with a slight twist – we (the humans) were the ones that needed to decode the message, whereas the aliens were the ones that wished to communicate it. A group of scientists had discovered a large metal object of alien origin frozen in the ice in Antarctica that was sending out some kind of radio message. One of the scientists eventually decoded the message, but only after some of the other scientists had already just opened the object – the phrase "*Do not open*" was being repeated over and over again in the radio message.

The question is the following. How can we transmit even such a simple message as "*Do not open*" when we do not know what language the creatures receiving the message will be capable of understanding? We may not know the cognitive abilities of the receivers, their cultural background or even their sensory capabilities, and therefore have very little common frames of reference with which to communicate with. Language involves the communication of concepts – that is, the meaning behind the symbols being transmitted in the message. Specifically, how do we communicate the concept of "*open*", for example, or the concept "*Do not*"?

Astronomers Carl Sagan and Frank Drake were faced with this problem in the early 1970s when deciding what messages to put on the Pioneer spacecraft, the first objects of human manufacture that have left the solar system. They chose to use a pictorial message that included nude figures of human beings along with several symbols to convey information about where the spacecraft came from. The messages have been criticised by other people for being too human-centric and hard to understand, even for humans – see Gombrich (1972) in *Scientific American*, for example. Even parts of the pictures that are easiest for humans to understand – an arrow showing the spacecraft trajectory – might be indecipherable to an alien culture, since the arrow symbol is a product of hunter-gatherer societies that are common on Earth.

Communicating with aliens is clearly a hypothetical situation. It is uncertain whether any extra-terrestrials exist, or that if they do exist, whether they have the ability to communicate with us – after all, we have found very little evidence that supports existence of extra-terrestrials elsewhere in the universe even after decades of searching with powerful computers. Seti@home is a distributed computing project that makes use of idle computers on the Internet to search for evidence of extra-terrestrial intelligence. Since its launch date on May 17, 1999, it has logged over ten million years of computer time and currently makes use of over 300,000 computers in over 200 countries. However, the project has not produced any evidence despite running for over ten years.

However, all communication between human agents may be considered to have some degree of 'alienness' in the following sense. Whenever we have a conversation with another human, we are often required to explain the meanings of concepts that are unknown or foreign to us, or explain concepts that are being understood in different ways. If the languages of the sender and receiver are different – for example, a conversation between someone from Wales who mainly speaks Welsh and someone from Africa who mainly speaks Swahili, or a deaf person who knows sign language and a person who does not – then longer explanations are required more often throughout the conversation. The difficulty with aliens, however, is that we might be faced with the problem of having to explain the meanings of everything that we wish to communicate.

The problem of how to communicate with an alien is similar to the problem of how to communicate with a newly discovered tribe, for example in Papua New Guinea, that has been isolated from the rest of humanity. The problem was considered by the twentieth century philosopher W. Quine who called it *radical translation*. The difficulty in communication arises, whether it be with an isolated tribe or an alien species, when perhaps not even a single word of the language the sender is using is understood in advance.

Essentially that is the problem faced by designers of text and speech understanding systems, because we can consider the 'alien' agent to be the computer system, and the human agent the one who produced the text or speech that needs to be understood.

Download free eBooks at bookboon.com

As stated in Section 1.4, much of language is made up of conceptual metaphor and analogy. Lakoff and Johnson (1980) in their book *Metaphors we live by* highlight the important role that conceptual metaphor plays in natural language and how it is linked with our physical experiences. They argue that metaphor is not just a device of the poetic imagination but "*is pervasive not just in everyday language, but in our thoughts and action*", being a fundamental feature of the human conceptual system. They state that "*The essence of metaphor is understanding and experiencing one kind of thing in terms of another*." They claim that most of our thought processing system is conceptual in nature and that people use metaphorical concepts much more frequently than they think. The metaphors are related to our physical embodiment in the real world. The purpose of the metaphorical device is to highlight similarities and hide dissimilarities between related concepts.

Some examples of conceptual metaphor have already been provided in Tables 4.1 and 5.1. Further examples are provided in Tables 7.3 and 7.4 below.

Click on the ad to read more

Download free eBooks at bookboon.com

Conceptual Metaphor	Sample Phrases
ARGUMENT is WAR.	Your claims are *indefensible*. He *attacked every weak point* in my argument. His criticisms were *right on target*. I *demolished* his argument.
TIME is MONEY.	You're *wasting* my time. This gadget will *save* you hours. I don't *have* the time to *give* you. How do you *spend* your time these days?
HAPPY is UP. SAD is DOWN.	I'm feeling *down*. That *boosted* my spirits. My spirits *rose*.
CONSCIOUS is UP. UNCONSCIOUS is DOWN.	Get *up*. He *rises* early in the morning. He *sank* into a coma.
HIGH STATUS is UP. LOW STATUS is DOWN.	He has a *lofty* position. She'll *rise* to the *top*. She's at the *bottom* of the social *hierarchy*. I look *down* at him.

Table 7.3 Some examples of conceptual metaphor and their use as identified by Lakoff and Johnson (1980).

We can examine the first example in Table 7.3 – ARGUMENT is WAR. Lakoff and Johnson devised a simple thought experiment related to this metaphor to illustrate that our own personal points of view are to some extent prescribed by the metaphors in the language we commonly use. Imagine an alien culture quite different from our own. In this culture, an alternative metaphor ARGUMENT is DANCE is predominantly used instead.

To these people, the concept that argument is related to the act of war is alien – instead, the emphasis in any confrontation or argument is placed on an active co-operation of the participants. The goal is not to destroy each other's arguments but to actively enhance each other's points of view and/or their own knowledge and understanding. This is in a similar way that the result of two people dancing together is artistically more appealing than two people dancing separately and independently. In conversation, these people might say things such as "*I combined with his argument*", "*Our arguments were out of step*" and "*I picked up his criticisms and swept them along*". For us, these would seem initially strange until we understood the underlying conceptual metaphor.

Table 7.4 further illustrates the pervasive use of conceptual metaphor in language. The first fifteen lines of the song "*The Winner Takes It All*" by ABBA are shown along with the conceptual metaphors that underlay their meaning.

Download free eBooks at bookboon.com

Line from the song	Conceptual Metaphor(s)
I don't wanna talk	Not 'INTERACTING' is Not TALKING
About the things we've gone through	'LIFE' is a 'JOURNEY'
Though it's hurting me	Not 'INTERACTING' is 'PAIN'
Now it's history	'LIFE' is a 'JOURNEY'
I've played all my cards	'LIFE' is a 'GAME'
And that's what you've done too	'RELATIONSHIP' is 'TWO-SIDED'
Nothing more to say	'INTERACTING' is 'TALKING'
No more ace to play	'LIFE' is a 'GAME'
The winner takes it all	'LIFE' is a 'GAME'
The loser standing small	'LOW STATUS' is 'BEING DOWN'
Beside the victory	'LIFE' is a 'GAME'
That's her destiny	'LIFE' is 'FIXED' 'FATE' is an 'IMMOVABLE OBJECT'
I was in your arms	'BELONGING' is 'HOLDING'
Thinking I belonged there	'BELONGING' is 'BEING THERE'
I figured it made sense	'SENSING' is `BUILDING SOMETHING' 'THINKING' is 'VISUALISING' 'THINKING' is 'BUILDING SOMETHING'

Table 7.4 Conceptual metaphor in the ABBA song "The Winner Takes It All".

Click on the ad to read more

Download free eBooks at bookboon.com

Much of the meaning in the ABBA song is revealed by the stream of conceptual metaphors shown in the table. However, the meaning of the song is richer than just the meaning of the individual words and sentences. Context plays an important part – for example, the meaning of the song is richer if we take into account the context of the group splitting up when the song was written and the powerful performances of the lead singer as she sang the song reveals this.

Human language has the ability to convey multiple meanings at several levels. Each of these interacts to provide something that is greater than the sum of its parts. Sometimes the reader or listener finds meaning that not even the author(s) intended. Language also reveals a great deal about the thoughts going on in the minds of the author(s) – what they know and what they don't know; their beliefs; their likes or dislikes; their opinions; their hopes and fears; what they respect and have high regard for or what they disrespect and have contempt for; what they support or are opposed to; their desires/wants/needs; and their intentions. The following excerpt (on page 418) from the autobiography *My Life* written by the former president of the United States, Bill Clinton, illustrates this:

> *"[Mandela's] visit was the beginning of a **great friendship** for all of us. Mandela **plainly liked** Hillary, and I was **really struck by the attention** he paid to Chelsea. In the eight years I was in the White House, he never talked to me without asking about her. Once, during a phone conversation, he asked to speak to her too. I've seen him show the **same sensitivity to children**, black and white, who crossed his path in South Africa. It speaks to his **fundamental greatness**."*

In the excerpt, Bill Clinton is revealing his thoughts about Nelson Mandela, the former president of the Republic of South Africa. The words underlined above show that Clinton classes Mandela as his friend, that he believes Mandela likes his wife, that he feels that Mandela is sensitive to children, and the overall impression is that he believes Mandela to be a great person.

The purpose of these examples is to highlight the complexities that occur in human language communication. Human language provides perhaps one of the greatest challenges in Artificial Intelligence research, where there are many frontiers in research still being explored or still left to be explored. Some of these will be discussed in Volume 2 of this book series.

Download free eBooks at bookboon.com

7.12 Summary

Agents exhibit communicating behaviour when they attempt to convey information to other agents. Language is the set of symbols agents use to communicate the information. Humans use many forms of communicating behaviour, and their language is diverse and rich in meaning. The communication between humans is spread via social networking.

A summary of important concepts to be learned from this chapter is shown below:

- Entropy is defined to be the sum of the probabilities multiplied by the log of the probabilities for a probability distribution. It can be used as a means for measuring the information content in messages.
- Language is defined by a set of socially shared rules that define the commonly accepted symbols, their meaning and their structural relationships specified by rules of grammar.
- Human language comes in many forms and is extremely diverse.
- Language modelling is a mechanism for determining the conditional probabilities for symbols in a language.
- The entropy of a language can be estimated from a large corpus of text representative of the language.
- Social networking plays a crucial role in how communication is spread between humans.
- The small world phenomenon is an important hallmark of social and computer networks.
- The purpose of communication is to convey the meaning of the message being communicated. The meaning conveyed in human messages is complex and multi-layered, and an area that presents a continuing challenge for Artificial Intelligence research.

The code for the NetLogo models described in this chapter can be found as follows:

Model	URL
Being Kevin Bacon	http://files.bookboon.com/ai/Being-Kevin-Bacon.nlogo
Cars Guessing Game	http://files.bookboon.com/ai/Cars-Guessing-Game.nlogo
Language Modelling	http://files.bookboon.com/ai/Language-Modelling.nlogo
Shannon Guessing Game	http://files.bookboon.com/ai/Shannon-Guessing-Game.nlogo

Model	NetLogo Models Library (Wilensky, 1999) and URL
Communication-T-T Example	Code Examples > Communication-T-T Example; for modified model used here: http://files.bookboon.com/ai/Communication-T-T.nlogo
Language Change	Social Science > Language Change http://ccl.northwestern.edu/netlogo/models/LanguageChange

Download free eBooks at bookboon.com

8 Search

Lewis and Clark expedition by Charles Marion Russell.

Search is fundamental for intelligent behavior. It is not just another method or cognitive mechanism, but a fundamental process. If there is anything that AI has contributed to our understanding of intelligence, it is discovering that search is not just one method among many that might be used to attain ends but is the most fundamental method of all.

Allen Newell (1994).

Need help with your dissertation?

Get in-depth feedback & advice from experts in your topic area. Find out what you can do to improve the quality of your dissertation!

Get Help Now

Go to www.helpmyassignment.co.uk for more info

Helpmyassignment

Click on the ad to read more

Download free eBooks at bookboon.com

This chapter looks at searching behaviour. The chapter is organised as follows. Section 8.1 provides a definition of searching behaviour. Section 8.2 describes some search problems that we use throughout this chapter to test out different search behaviours on. Section 8.3 describes several uninformed (or blind) search behaviours and Section 8.4 shows how they can be implemented in NetLogo. Section 8.5 discusses how search from an embodied, situated perspective can be thought of as a search of alternative behaviours rather than a search of alternative paths. Section 8.6 describes several informed search behaviours and how they can be implemented in NetLogo. Section 8.7 describes local search and optimisation behaviours and their implementation. The different search behaviours are compared in Section 8.8.

8.1 Search Behaviour

It is generally acknowledged in Artificial Intelligence research that search is crucial to building intelligent systems and for the design of intelligent agents. For example, Newell (1994) has stated that search is fundamental for intelligent behaviour (see the quote at the beginning of this chapter). From a behavioural perspective, search can be considered to be a meta-behaviour where the agent is making a decision on which behaviour amongst a set of possible behaviours to execute in a given situation. In other words, it can be defined as the behavioural process that the agent employs in order to make decisions about which choice of actions it should perform in order to carry out a specific task. The task may include higher-level cognitive behaviours such as learning, strategy, goal-setting, planning, and modelling (these were called Action Selection in Reynold's Boids model). If there are no decisions to be made, then searching is not required. If the agent already knows that a particular set of actions, or behaviour, is appropriate for a given situation, then there is no need to search for the appropriate behaviour, and therefore the actions can be applied without coming to any decision.

Searching can be considered to be a behaviour that an agent exhibits when it has insufficient knowledge in order to solve a problem. The problem is defined by the current situation of the agent, which is determined by the environmental conditions, its own circumstances and the knowledge the agent currently has available to it. If the agent has insufficient knowledge in order to solve a given problem, then it may choose to search for further knowledge about the problem. If it already has sufficient knowledge, it will not need to employ searching behaviour in order to solve the problem. An agent uses search behaviour in order to answer a question it does not already know the answer to or complete a task it does not know how to complete. The agent must explore an environment by following more than one path in order to obtain that knowledge.

Download free eBooks at bookboon.com

The exploration of the environment is carried out in a manner analogous to early explorers of the American or Australian continents, or people exploring a garden maze such as the Hampton Court Palace Maze or the Chevening House Maze. For the two garden mazes, the people trying to get to the centre of the maze must employ search behaviour if they do not have the knowledge about where it is. If they take a map with them, however, they no longer have to use search behaviour as the map provides them with the knowledge of which paths to take to get to the centre. The early American explorers Lewis and Clark were instructed by Thomas Jefferson to explore the Missouri and find a water route across the continent to the Pacific. They did not already know how large the continent was or what was out there and needed to physically explore the land. They chose to head in a westerly then north-westerly direction following a path along the Missouri river. Similarly, the Australian explorers Burke and Wills led an expedition starting from Melbourne with the goal of reaching the Gulf of Carpentaria. The land at the time had yet to be explored by European settlers. The expedition were prevented from reaching their ultimate goal just three miles short of the northern coastline due to mangrove swamps and, worse still, the expedition leaders died on the return journey.

When performing a search, an agent can adopt different behaviours that determine the way the search is performed. The search behaviour adopted can have a significant impact on the effectiveness of the search. For example, poor leadership was blamed for the unsuccessful Burke and Wills expedition. The behaviour can be thought of as a strategy the agent adopts when performing the search. We can consider these search behaviours from an embodied agent perspective. The type of search behaviour is determined both by the embodiment of the agent, and whether the agent employs a reactive or more cognitive behaviour when executing the search.

We have already seen many examples of how an agent can perform a search of an environment as a side effect of employing purely reactive behaviour (see Section 5.4 and Figures 5.2, 5.7 to 5.10 and 6.9). We can term these types of search behaviours as *reactive search*. We can also use the term *cognitive search* for cases when an agent has an ability to recognize that there is a choice to be made when a choice presents itself in the environment (such as when the agent reaches a junction in the maze). As stated in section 5.5, this act of recognition is a fundamental part of cognitive-based searching behaviour, and it is related to the situation that the agent finds itself in, the way its body is moving and interacting with the environment. It is also related to what is happening in the environment externally, and/or what is happening with other agents in the same environment if there are any.

In addition, a single agent can be used to perform any given search, but there is nothing stopping us from using more than one agent to perform the search. We can adopt a multi-agent perspective to describe how each particular search can be performed in order to clarify how the searches differ. In this case, the effectiveness of a particular search can be evaluated by comparing the number of agents that are needed to perform the search and the amount of information that is communicated between them.

Download free eBooks at bookboon.com

8.2 Search Problems

There are many problems that require search with varying degrees of difficulty. Simplified or 'toy' problems are problems that are to the most part synthetic and unrelated to real life. However, these problems can be useful to designers in gaining insight into the problem of search. Once the properties of the different search behaviours have been investigated on toy problems, then they can be adapted and/or scaled up to real life problems. To illustrate different search problems, this section describes three problems in particular that have been simulated in NetLogo – these are the Searching Mazes model, the Missionaries and Cannibals model, and the Searching for Kevin Bacon model.

The garden maze problems introduced in Chapter 3 and depicted in Figures 3.6 and 3.7 can be thought of as toy problems that have a corresponding problem in real-life. These are toy problems not because they are artificial problems, but because of their relative simplicity. However, they provide useful test beds when developing and evaluating the effectiveness of various search behaviours as we can use them to compare how well each behaviour does on them. Then with this information we can go on to tackle much larger or more difficult search problems.

A classic toy problem often used to demonstrate aspects of searching is the missionaries and cannibals problem. In this dilemma, there are three missionaries and three cannibals. They wish to all get across a river using a canoe that only holds two people, but the problem is that if the cannibals ever outnumber the missionaries at any stage, then the cannibals will eat the missionaries.

Click on the ad to read more

Download free eBooks at bookboon.com

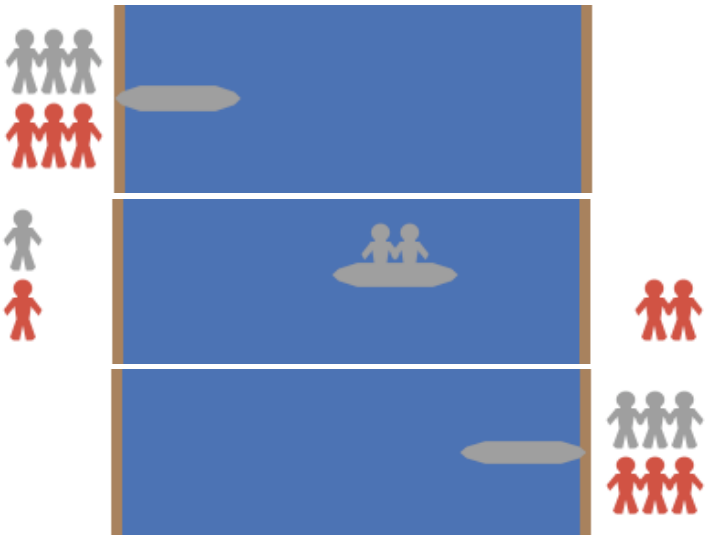


Figure 8.1 Screenshots of the animated part of the Missionaries and Cannibals model.

Figure 8.1 shows some screenshots of an animation provided by the Missionaries and Cannibals NetLogo model that demonstrates a solution to the problem. The top image represents the start state, with three missionaries, three cannibals and the canoe on the left side of the river. The gray coloured persons represent the missionaries and the red coloured persons represent the cannibals. The middle image depicts an intermediate state where there is one missionary and one cannibal on the left side, there are two missionaries in the canoe half way across the river, and there are two cannibals on the right of the river. The bottom image represents the final desired or goal state where everybody ends up on the right side.

Searching for people is a problem that often occurs in real life. It also has a corollary to many computer network problems, for example resource discovery in peer-to-peer networks where a particular resource such as a file or computer with specific CPU requirements needs to be located. The Searching for Kevin Bacon model simulates this problem by creating random networks as shown in Figure 8.2.

Download free eBooks at bookboon.com

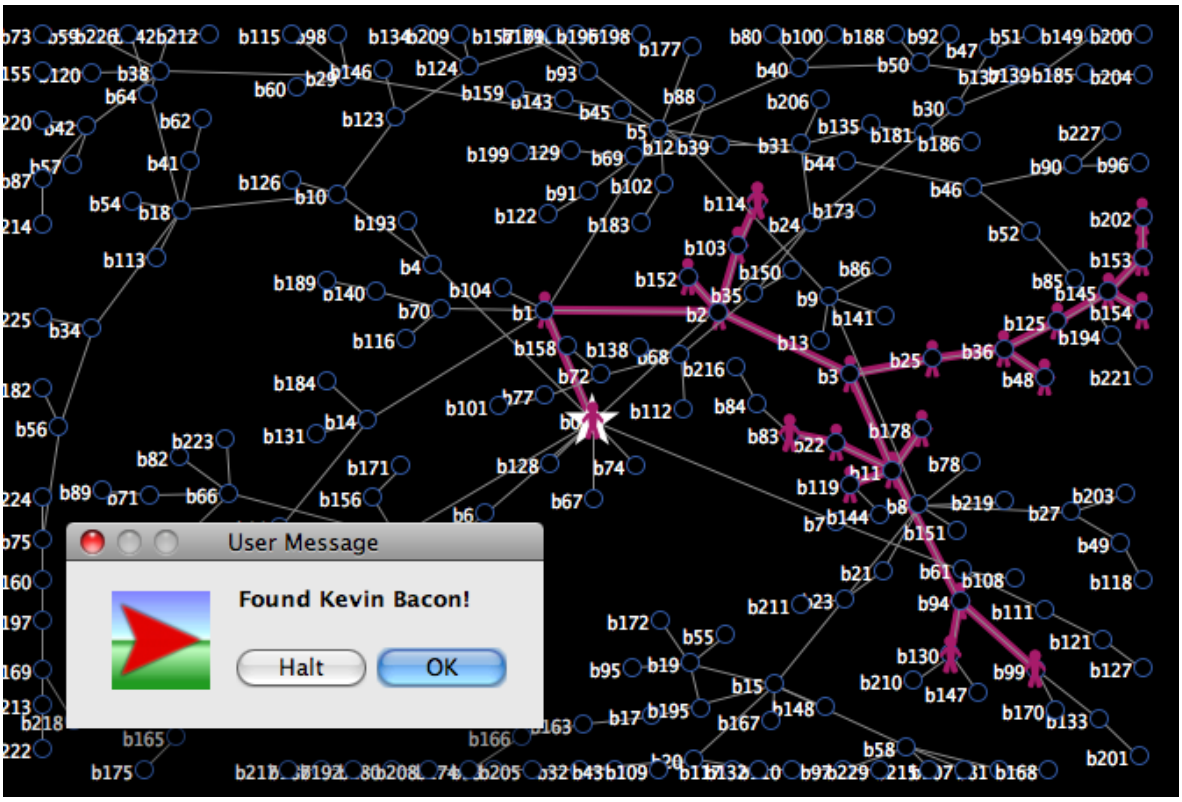


Figure 8.2 A screenshot of the Searching for Kevin Bacon model.

A particular node in the network is designated to be the desired or goal state – this is shown by a node drawn as a star in the middle of the image. The search starts at a random node in the network – for the image shown, the start node was node b154 in the middle right of the image. In this scenario, an analogy with a real life problem is a classroom full of students where the objective is to pass a specific object such as a note from person to person until it reaches the intended recipient. The divergence from the real life problem occurs when the agents in the simulation (represented as nodes in the network) have specific constraints placed on their sensory capabilities that make the simulation more analogous to searching in computer networks. In this case, the agents are 'blind' in the sense they have no idea about the global configuration of the network and only have local knowledge of what is immediately around them (i.e. they only know the links to neighbouring nodes). In addition, the agents do not know where the star node is.

A search problem can be characterised as an agent exploring a finite state automaton in an n-dimensional space. One or more states are designated as the start states – that is, the location where the agents performing the search first head out on their search. These start states are connected to other intermediate states which are in turn connected to further states and so on until eventually the connections will lead to desired or goal states when the search completes, not unlike the situation modelled in the Searching for Kevin Bacon model. Search is required when the agents do not know how to get to the desired states, or wish to find the shortest path to the desired states. If they already know the path, then no search is required. Otherwise, they must explore the finite state automaton until they find a desired state or until they have found the shortest path.

Download free eBooks at bookboon.com

8.3 Uninformed (blind) search

Uninformed or blind search occurs when the agent has no information about the environment it is searching. A real-life analogy to this type of search is a blind person searching a maze he has never been inside before, with no prior knowledge of its dimensions or the whereabouts of the centre or exit of the maze. One approach a 'blind' search agent can take is to keep on making one particular choice at each junction he encounters, and continue until he either reaches the exit or centre (the goal) or until he reaches a dead end. If the latter, he can then backtracks to the last junction he has visited that has paths he hasn't already searched, and then chooses one of those. He repeatedly applies this behaviour until the goal is reached. This agent still has one further choice to make – the order that he chooses to follow the paths. He could choose the paths at random, for example, or he could choose to always follow the first branch on his right first, and then follow the subsequent branches in clockwise order.

Click on the ad to read more

Download free eBooks at bookboon.com

This type of search is called *depth first* search. In order to simplify the discussion that follows, and in order to explain the multi-agent based solutions that have been developed for the NetLogo models described below, we will depart from the real-life analogy of a single agent performing the searching in the following way. At each junction or decision point in the search, instead of the agent itself continuing the search along a single path, we will provide the agent with the ability to send out clones of itself along all the paths it finds. In the process of cloning itself, it will then die. For depth first search, one of the newly cloned agents will then be chosen to continue the search while the others sit and wait, just in case they are needed at a latter time. This process of clone creation followed by picking one of the clones to further the search continues until a goal or a dead end is reached. If the latter, the clone will die without creating any further clones of itself, and the search will continue with the clone that has been waiting the shortest time.

This is illustrated in Figure 8.3, which shows depth first search for the Searching for Kevin Bacon model. In the top right image, a single agent starts from the bottom left node labelled `b26` and is trying to search for the goal node drawn as a white star. This agent then moves to node `b14` after cloning itself once, then creates two clones of itself because there are a further two possible paths to follow (to nodes `b5` and `b23`) as shown in the top right image. It then dies and the search is continued by one of the agents, in this case the agent at node `b5`. The search continues along the `b5` branch heading by luck in the direction of the goal node until it reaches node `b3` as shown in the bottom left image. Then one clone is chosen to search the side path that heads north at `b8`, but once this path has been completely searched, the clone at `b1` takes over and finally the goal node is reached as shown in the bottom right image.

Download free eBooks at bookboon.com

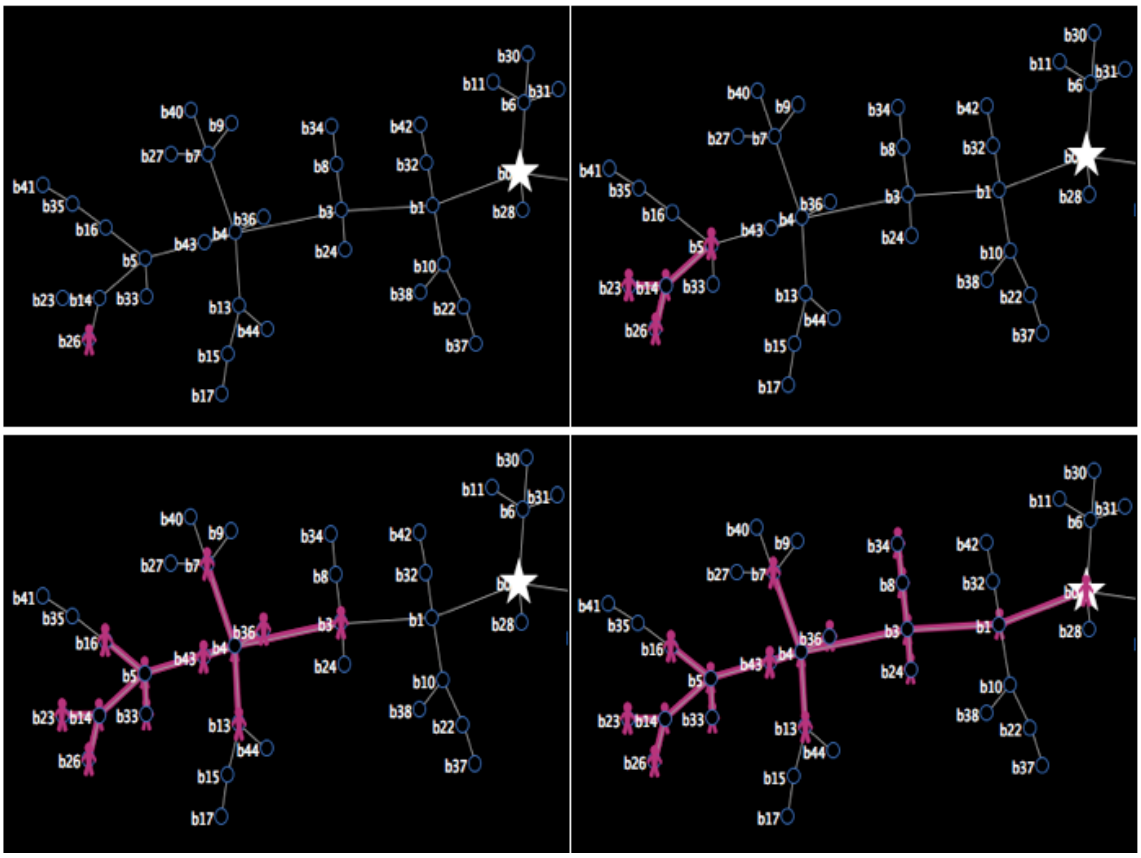


Figure 8.3 Screenshots of the depth first search for the Searching for Kevin Bacon model.

An obvious alteration to this type of search is to allow more than one agent clone to take on the search simultaneously. In other words, instead of a single clone doing the search by itself, we can send out all the clones to do the searching together in parallel. This type of search is called *breadth first* search. The way the network is searched in the Searching for Kevin Bacon model using breadth first search is shown in Figure 8.4. The left image is a screenshot during an intermediate part of the search. The right image shows the search when it is complete.

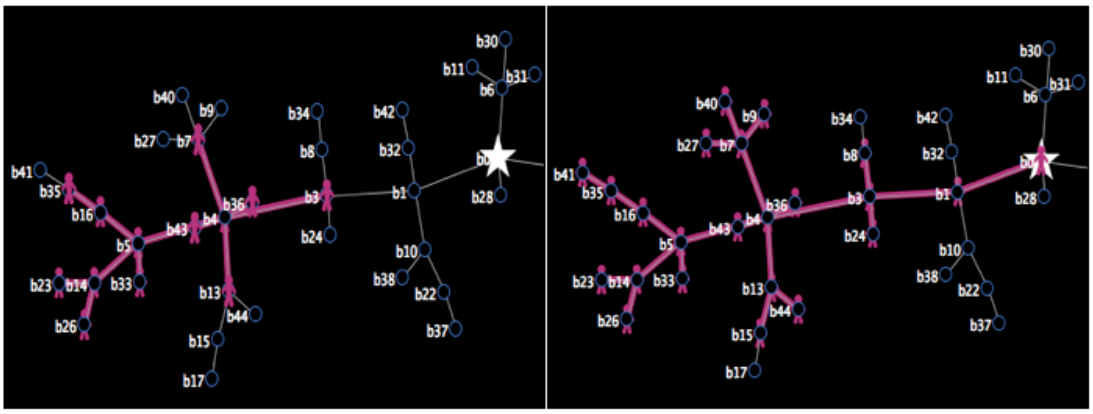


Figure 8.4 Screenshots of the breadth first search for the Searching for Kevin Bacon model.

Download free eBooks at bookboon.com

To understand the fundamental differences between these two different types of search, we can think of the analogy of the Lewis and Clark expedition exploring the American continent following the Missouri river and its tributaries. As they progressed, the 'frontier' of their knowledge would 'expand'. If this expedition adopted a depth first search behaviour, the frontier would expand by following one particular branch of the river – they would keep on getting deeper and deeper down a single branch. If the tributary stopped, then the expedition would return to the last junction and start searching the alternative path. Referring to the analogy above, the clones can be thought of as locations of past selves that are returned to when one path has been unsuccessful. For the breadth-first behaviour, on the other hand, the expedition adopts a policy of splitting the team up to go searching the alternative branches at the same time. In this case, the frontier expands outwards at the same rate not unlike a ripple expands evenly outwards when a pebble is thrown into a pond.

Using another analogy, the expanding frontier can be thought of as a 'tree' with roots growing deeper into the soil. For depth-first search, one root is grown at a time, and this keeps on getting deeper and deeper (hence why this particular type of search is called depth-first search) until bedrock blocks it. For breadth-first search, the roots all grow at a similar rate, and expand evenly downwards together.

We can look at the Searching Mazes model to further illustrate the differences between these two searches. We can select the empty maze to search, then set the `move-forward-behaviour` and `move-forward-behaviour-after-turning` choosers to `Move-forward-n-steps-unless-hit-wall,` as well as setting the slider `move-forward-step-amount` to 2. This will ensure that the search expands into the empty space only 2 steps at a time as shown in the screenshots of Figure 8.5. These graphically illustrate the differences between the two searches. The depth-first search is opportunistic, heading in whatever direction it randomly chooses, whereas the breadth-first search is more thorough, with every possibility searched before moving on.

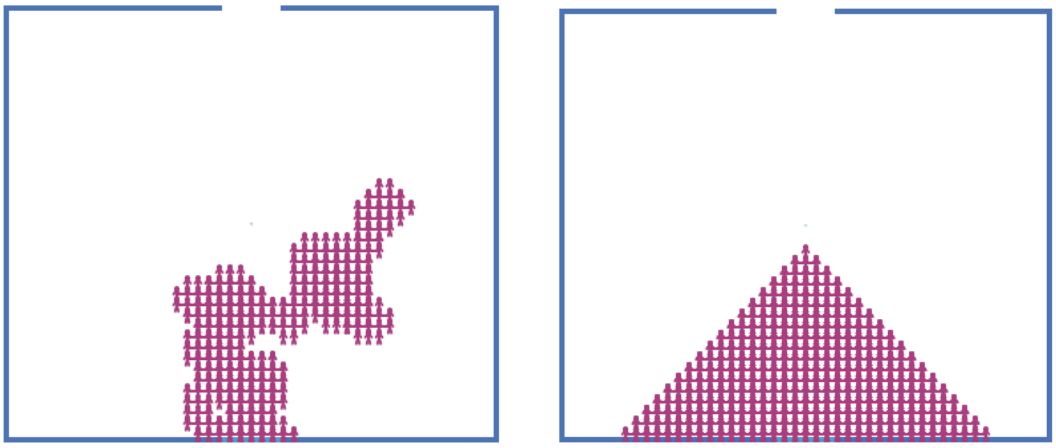


Figure 8.5 Screenshots of the depth-first search and the breadth-first search being performed on the empty maze for the Searching Mazes model.

Download free eBooks at bookboon.com

The terms 'frontier', 'tree' and 'expanding the search' are often used to describe searching. The possible paths in the search can be thought of as a search tree using the analogy of branches and leaves – that is, the analogy is of a tree growing above the ground rather than roots growing into the soil as above. In this case, the frontier of the search can be thought of as the leaf nodes of the tree. These are the nodes of the tree that need to be searched next. For each leaf node, there is a defined set of possible nodes that can be expanded.

For example, for the left top image in Figure 8.3, the set of expanded nodes for node `b14` is `{ b5, b23, b26 }`. Even though it has just come from node `b26`, there is nothing stopping the search returning to where it came from unless the search specifically disallows revisiting of nodes. The advantage of not revisiting nodes is that the search will not waste search time on something it has already searched; however, the disadvantage is that it must record where it has been. In some applications where there are a very large number of nodes to visit (for example, Web crawlers need to visit billions of nodes) then storing the set of already visited nodes can be expensive and pose major engineering problems. The three NetLogo models all provide an Interface switch that sets the variable `allow-revisited-states` to turn on or off the revisiting of nodes. The models also maintain a global variable called `visited-nodes` which is a list of all the nodes already visited.

Click on the ad to read more

Download free eBooks at bookboon.com

Whether revisiting of nodes is allowed or not is an important factor affecting how well the various searches perform. This is apparent with the Missionaries and Cannibals model, for example, with some of the searches requiring substantially more searcher agents to complete the search, or in some cases, the search not completing at all. Figure 8.6 shows a screenshot of the Missionaries and Cannibals model for the breadth-first search with revisiting allowed. The model shows the execution of the search as it proceeds – each iteration is displayed by pressing the `go-once` button in the Interface. For this problem, we can represent the state that the search has reached using a tuple containing three numbers – the number of missionaries on the left side of the river (0 to 3), the number of cannibals on the left side (0 to 3), and the number of canoes on the left side (1 or 0). For example, the start state is represented by the tuple (3, 3, 1) as there are 3 missionaries, 3 cannibals and 1 canoe on the left side of the river at the start. The desired or goal state is represented by the tuple (0, 0, 0) with everybody and the canoe now over on the right side. There are 32 possible states the search can reach but not all of them will be allowable – for example, the tuple (1, 3, 0) is not an allowable state as this represents the case when the number of cannibals is greater than the number of missionaries by 2.

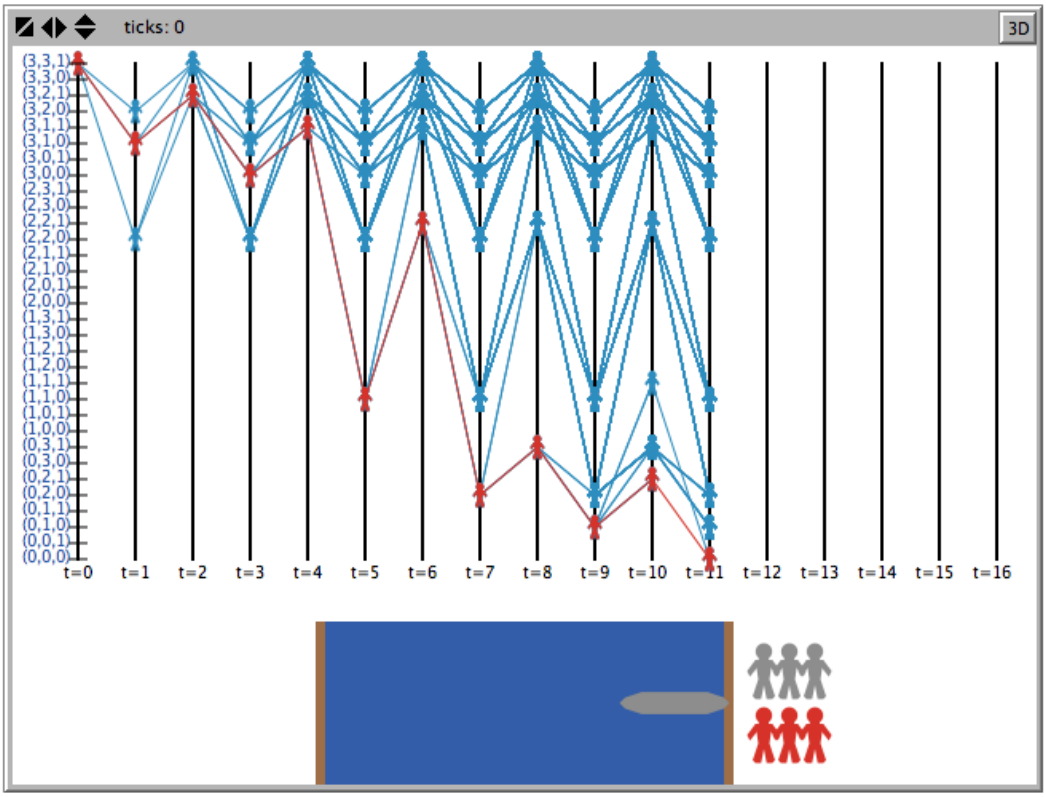


Figure 8.6 Breadth-first search with revisiting allowed for the Missionaries and Cannibals model.

Download free eBooks at bookboon.com

This representation allows us to visualise the search using a parallel co-ordinate system as shown in Figure 8.6. The iteration of the search each time step is shown on the parallel axes, with the start of the search shown at tick t=0 on the left. Then the search moves to three possible states at tick t=1: to (3, 2, 0), after one cannibal has rowed across to the right in the canoe; to (3, 1, 0), after two cannibals have rowed across in the canoe; and to (2, 2, 0), after one missionary and one cannibal have rowed across. The figure shows the paths the breadth-first search takes through the state space environment. The number of agents significantly grows with each tick. The screenshot also shows one possible solution drawn in red. This is drawn from left to right during the animation shown at the bottom once the go-animation button is pressed in the Interface.

We can stop the search from revisiting states by setting the `allowed-revisited-states` switch in the Interface to `Off`. This has a dramatic effect by eliminating most of the paths from the search as shown in Figure 8.7.

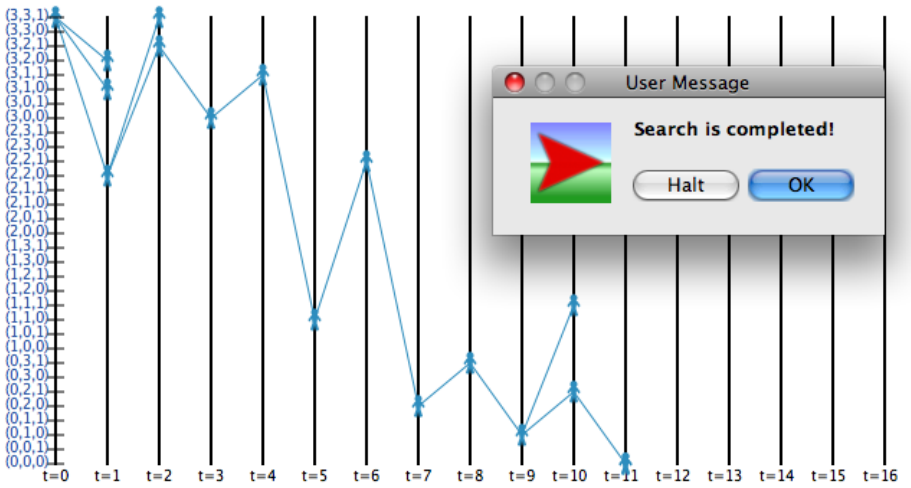


Figure 8.7 Breadth-first search with revisiting not allowed for the Missionaries and Cannibals model.

A similar effect is seen with depth-first search. Figure 8.8 shows what happens with depth first search in the Missionaries and Cannibals model with revisiting allowed. The screenshot shows the search repeatedly choosing the same wrong paths that never lead to a solution. Eventually, the search is aborted since it is taking too long.

Download free eBooks at bookboon.com

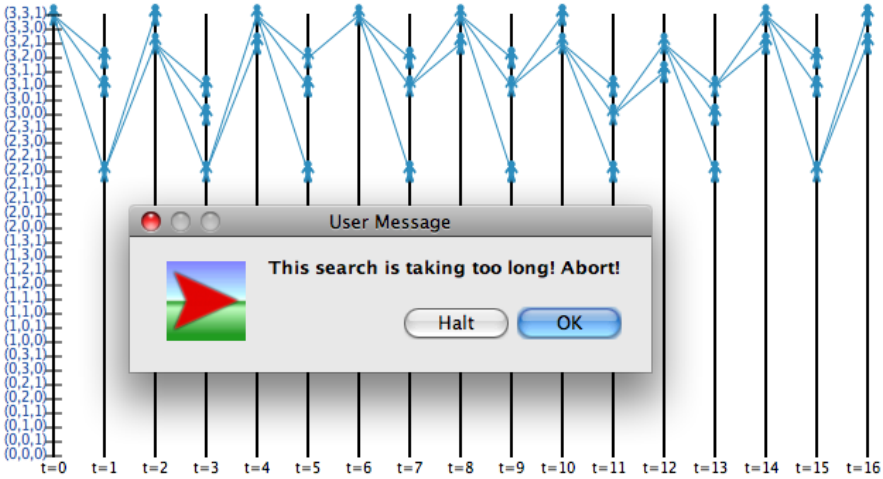


Figure 8.8 Depth-first search with revisiting allowed for the Missionaries and Cannibals model.

Figure 8.9 shows what happens with depth first search when revisiting is not allowed. The search almost immediately finds the correct path as the alternate wrong paths are eliminated as possibilities.

Click on the ad to read more

Download free eBooks at bookboon.com

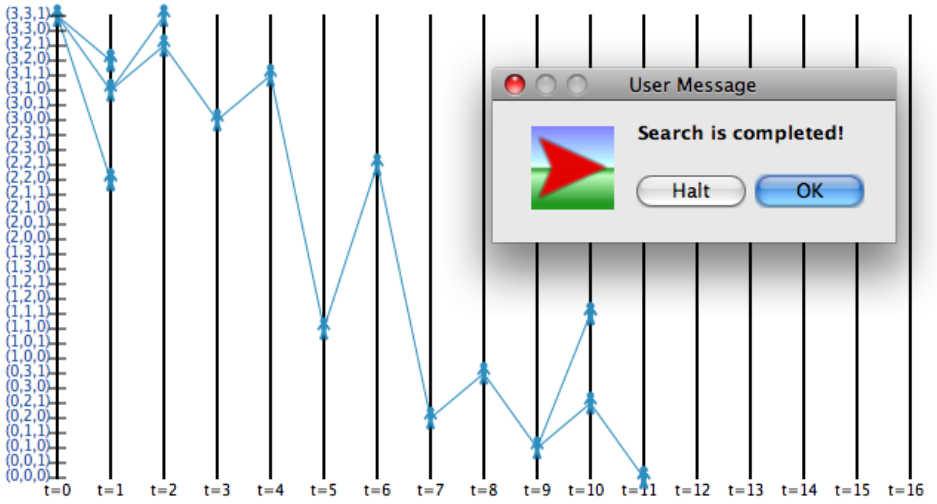


Figure 8.9 Depth-first search with revisiting not allowed for the Missionaries and Cannibals model.

8.4 Implementing uninformed search in NetLogo

The code for expanding the search frontier for the Searching for Kevin Bacon model is shown in NetLogo Code 8.1 below.

```
to expand-paths [searcher-agent]
;; expands all the possible paths for the searcher-agent

foreach sort [link-neighbors] of [location] of searcher-agent
   [ expand-path searcher-agent ? ]
end

to expand-path [searcher-agent node]
   ; the searcher-agent creates a new searcher-agent that draws a path in the
   ; network from its current position to the node

   let xcor1 0
   let ycor1 0
   if not search-completed
      [ ; create a new path by creating an agent to search it
      ; check to see if the path has already been visited
      if allow-revisited-nodes or not member? node visited-nodes
           [

             set path-found true
             if not allow-revisited-nodes
               [set visited-nodes fput node visited-nodes]
               ; add to front of visited-nodes set
             hatch-searchers 1
               [ ; clone searcher
                set searchers-used searchers-used + 1
```


Download free eBooks at bookboon.com

```
                set size 2
                set pen-size 5
                set color magenta
                set shape "person"
                set xcor [xcor] of searcher-agent
                set ycor [ycor] of searcher-agent
                set xcor1 xcor ; copy xcor
                set ycor1 ycor ; copy ycor

                set heading [heading] of searcher-agent
                set time [time] of searcher-agent + 1
                set path-cost [path-cost] of searcher-agent
                pen-down

                ; move to the node
                 set location node
                 move-to location
                 set xcor [xcor] of self
                 set ycor [ycor] of self
                 ; increment path cost when executing the behaviour using
                 ; Euclidean distance
                 set path-cost path-cost +
                   euclidean-distance xcor1 ycor1 xcor ycor
set estimated-cost (heuristic-function xcor ycor)
               set height hill-height xcor ycor

               stamp
           ]
     ]
     if goal-node node
     [ set search-completed true ]
     ]
end

to-report goal-node [this-node]
;; returns true if the searcher agent has reached the goal

   report this-node = kevin-bacon-node
end
```

NetLogo Code 8.1 How the paths are expanded for the Searching for Kevin Bacon model.

The procedure `expand-paths` calls the sub-procedure `expand-path` for each node adjacent to the node at the current location of the searcher agent i.e. each node in the set of `link-neighbours` of the searcher agent's node. This sub-procedure creates a new path to each adjacent node (passed as a parameter) by creating an agent to search it. It first makes a check, however, to see whether the path has already been visited, and revisiting of nodes is not allowed, and if this is true then it will not create a new agent.

Download free eBooks at bookboon.com

When a new agent is created, it is initially placed at the location of the current searcher agent, and then it is moved to the adjacent node. This sub-procedure also maintains the path cost – the distance that has been travelled so far in the search along the current path; in this model, the path cost is calculated using Euclidean distance. When the new agent is cloned, it will copy across the path cost so far, and this is incremented once the clone agent has moved to the new location. The estimated-cost of reaching the goal and the height are also calculated (these are used for the informed and hill climbing searches described below). At the end of the sub-procedure, a check is done to see if the goal node has been reached by calling the `goal-node` procedure. This procedure simply checks to see if the node is the Kevin Bacon node as defined at setup (represented by the star in the screenshots).

The breadth-first and depth-first search procedures are defined using the `expand-paths` procedure as shown in NetLogo Code 8.2. The code is the same for all three of the models – for the Searching for Kevin Bacon model and for the Searching Mazes and Missionaries and Cannibals models. The difference occurs for each model in the way the paths are expanded – that is, the way the procedure `expand-paths` is defined – as this depends on the nature of the problem and the way paths are defined in the environment. Full code defining each of the different `expand-paths` procedures can be found using the URLs at the bottom of this chapter.

Click on the ad to read more

Download free eBooks at bookboon.com

```
to expand-breadth-first-search
   ; expand the search by adding more searcher-agents

   ask searchers
   [
      expand-paths (searcher who)
      die ; prune away parent agents used at lower time-steps
   ]
end

to expand-depth-first-search
   ; expand the search by following newest path
   ; do not follow the other paths in parallel; just follow one of them

   set path-found false
   ask first (sort-by [[time] of ?1 > [time] of ?2] searchers)
   [
      expand-paths (searcher who)
      die ; this agent has done its work; it's children are now doing the work
   ]
end
```

NetLogo Code 8.2 How the breadth-first and depth-first searches are defined.

The breadth-first search expands the paths for all the current searcher agents via the `ask` command by calling the `expand-paths` procedure before killing each of them off. The `ask searchers` command ensures that the search is expanded in a random order each time. In contrast, the depth-first search expands the paths for a single agent, the newest searcher that has the highest time variable as determined by the `sort-by` reporter, before killing it off.

Simple variations of these two procedures lead to different search behaviours. One variation is to expand the lowest cost path first. This is called *uniform cost* search, the code for which is shown in NetLogo Code 8.3. The order that the searcher agents are expanded is determined by the `sort-by` reporter which orders the agents in ascending `path-cost` order.

```
to expand-uniform-cost-search
    ; expand the search by following lowest cost paths first

   set path-found false
   ask first (sort-by [[path-cost] of ?1 < [path-cost] of ?2] searchers)
   [
      expand-paths (searcher who)
      die ; this agent has done its work; it's children are now doing the work
   ]
end
```

NetLogo Code 8.3 How the uniform cost search is defined.

Download free eBooks at bookboon.com

For depth-first search, we can ease the restriction that a single agent is doing the search, and allow more than one agent to search in parallel at the same time. In the limit where there is no restriction placed on the number of agents that can search at the same time, this will end up being equivalent to breadth-first search. The multi-agent variation of depth-first search is shown in NetLogo Code 8.4 below. The only changes from the depth-first code above is the use of the global variable `max-agents-to-expand` to specify how many agents should simultaneously do the searching each iteration (this can be altered in the Interface) and the use of the `n-of` reporter instead of the `first` reporter in the `ask` command. The `let` command prior to the `ask` command is required when the number of agents is less than the `max-agents-to-expand` which usually occurs at the start of the search, otherwise the `n-of` reporter will cause a run-time error.

```
to expand-MA-depth-first-search
   ; expand the search by following longest path
   ; follow the other paths in parallel; but do not follow all of them set path-
   found false
   let agents ifelse-value (count searchers < max-agents-to-expand)
      [ count searchers ]
      [ max-agents-to-expand ]
   ask n-of agents turtle-set
      (sort-by [[time] of ?1 > [time] of ?2] searchers)
   [
      expand-paths (searcher who)
      die ; this agent has done its work; it's children are now doing the work
   ]
end
```

NetLogo Code 8.4 How the multi-agent depth-first search is defined.

Another variation to depth-first search is to limit the depth of the search to a specified maximum depth. This might be useful when the maximum depth of the search tree is unknown and we would like the search to explore alternative shorter paths first. In this case, paths are expanded only for those agents whose time variable is less than or equal to this limit. This is called *depth-limited* search, and the code is shown in NetLogo Code 8.5.

Download free eBooks at bookboon.com

```
to expand-depth-limited-search
   ; expand the search by following longest path
   ; do not follow the other paths in parallel; just follow one of them
   ; limit the search depth to max-depth

   expand-depth-limited-search-1 max-depth
end

to expand-depth-limited-search-1 [maxdepth]
   ; expand the search by following longest path
   ; do not follow the other paths in parallel; just follow one of them
   ; limit the search depth to maxdepth

   set path-found false
   ask first (sort-by [[time] of ?1 > [time] of ?2] searchers)
   [
      if (time <= maxdepth)
      [ expand-paths (searcher who) ]
      ; only expand if not exceeded depth limit
      die ; this agent has done its work; it's children are now doing the work
   ]
end
```

NetLogo Code 8.5 How the depth-limited search is defined.

Deloitte.

Discover the truth at www.deloitte.ca/careers

© Deloitte & Touche LLP and affiliated entities.

Click on the ad to read more

Download free eBooks at bookboon.com

The `expand-depth-limited-search` procedure calls a sub-procedure `expand-depth-limited-search-1` with a parameter `max-depth` which is a variable defined in the Interface. The sub-procedure is just a variation of the standard depth-first search procedure that stops expansion of paths if the maximum depth cut-off has been reached.

A problem with depth-limited search is that there is no guarantee that the search will be successful. If we choose a particular `max-depth`, then the desired or goal state may be just beyond the cut-off maximum depth. A variation to overcome this problem is called *Iterative Deepening Search*. Here, depth limited search is repeated to increasingly larger maximum depths as shown in NetLogo Code 8.6.

```
to expand-iterative-deepening-search
   ; expand the search by iteratively performing depth-limited-search
   ; to increasingly greater depths
   set IDS-depth 1
   set person-colour magenta
   while [ IDS-depth <= max-depth ]
   [
      while [count searchers != 0]
      [ expand-depth-limited-search-1 IDS-depth ]
      set IDS-depth IDS-depth + 1
      ; change colour of person to reflect the increased maxdepth of search
      if (person-colour > 5)
      [ set person-colour person-colour - 5 ]
      create-searchers 1 [ setup-searcher ]
   ]
   set person-colour magenta ; restore default colour
end
```

NetLogo Code 8.6 How the Iterative Deepening Search is defined.

The variable `IDS-depth` records what the current maximum depth for the depth limited search is set at. This is set to 1 at the beginning, and then the code uses a while loop to execute the depth limited search by repeatedly calling the `expand-depth-limited-search-1` procedure until there are no more searchers (this occurs when the depth-limited search has reached the maximum depth). The maximum depth each iteration is increased when the variable `IDS-depth` is incremented by 1. A single searcher is also recreated each iteration to re-start the depth-limited search from scratch and the colour of the searcher agents is changed in order to show the search as it progresses over the previous search that is shown in a different colour. The search will eventually stop when `IDS-depth` exceeds `max-depth`.

The NetLogo models also allow the search behaviours to be combined. The search strategy can be easily switched mid-way through the search. For example, the search can start out as a depth first search, then the user can switch the strategy to an alternative strategy such as breadth-first search via the `search-behaviour` chooser in the Interface.

Download free eBooks at bookboon.com

8.5 Search as behaviour selection

We can think of the search process as one of an agent moving around the environment that represents the search space. At each point where there are alternative paths to explore, the agent has to make a decision about which path to explore. Once a particular path has been chosen, then this will lead to a 'tree' of future branches. This abstract view of the search process is based on an analogy with exploration of unknown territory by selecting alternative paths. We can extend this abstract view by noting that the search process usually involves behavioural selection rather than just path selection. The agent searches for the sequence of behaviours that when executed will lead to the desired goal. The execution of the behaviours will lead the agent down particular paths as an outcome.

We can apply this view of the search process to characterise the three search problems described above. For example, in the searching mazes problem, the search process is more than just a search for a set of paths that will allow the agent to travel to the exit or centre of the maze. In order to follow the paths, the agent must execute specific behaviours such as turning left and right, moving ahead and turning around. Hence, the search can be characterised instead as a search for the sequence of behaviours that when executed in a specific order will lead to the desired goal. Specifically, the behaviour of turning left or right requires a single action, but the behaviour of forward movement involves multiple actions if we are to consider the embodiment and situated perspective outlined in chapter 5. Forward movement needs to be combined with sensing, for example the agent might apply a different behaviour whenever it senses a wall in front of it, or when it finds there is open space in front of it or to its side.

In the Searching Mazes model, in order to illustrate this 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.

Download free eBooks at bookboon.com

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.

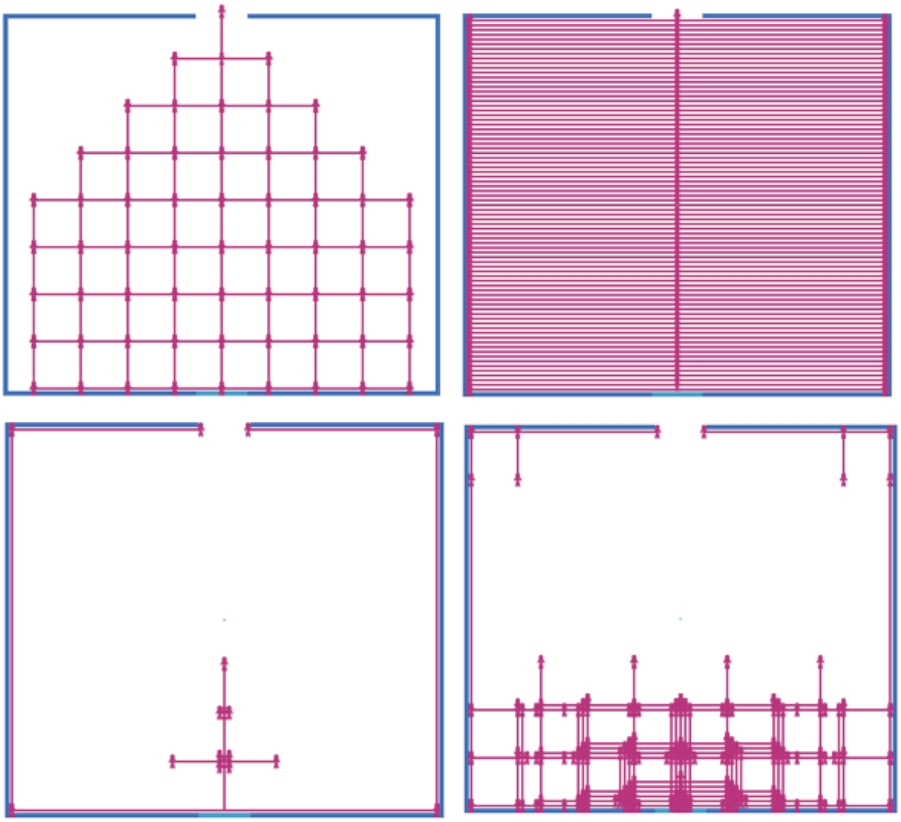


Figure 8.10 Screenshots of different movement behaviours for the Searching Mazes model.

The important distinction is that the search as implemented in the Searching Mazes model is not just a search for alternative paths – it is a search for the sequence of behaviours that the agent must execute in order to get to the goal state. This distinction is most apparent in the definition of the `expand-paths` procedure for the model as shown in NetLogo Code 8.7 where each "path" being expanded involves the execution of a specific movement behaviour – forward, left-then-forward or right-then-forward.

Download free eBooks at bookboon.com

```
to expand-paths [searcher-agent]
;; expands all the possible paths for the searcher-agent

   expand-path searcher-agent "Move forward"
   ; new searcher agent moves forward until it hits a wall
   expand-path searcher-agent "Move left"
   ; new searcher agent turns left then moves forward
   expand-path searcher-agent "Move right"
   ; new searcher agent turns right then moves forward
end
```

NetLogo Code 8.7 The procedure defining how the paths are expanded for the Searching Mazes model.

The other two search problems can also be characterised as a behavioural selection process rather than a path selection process. For the Searching for Kevin Bacon model (see the `expand-paths` procedure defined in NetLogo Code 8.1), the agents are applying a single behaviour – they 'look' around them to sense the surrounding nodes, and then they move to each in turn. In the search process implemented in the model, the behaviour being applied by the agents is the same as each node is treated as equivalent, but there is no reason why we could not vary the search by choosing between behaviours that take into account different properties of the nodes in the network, such as whether it was a super-node or not, or the distance away from adjacent nodes.

SIMPLY CLEVER
ŠKODA

Do you like cars? Would you like to be a part of a successful brand?
We will appreciate and reward both your enthusiasm and talent.
Send us your CV. You will be surprised where it can take you.
Send us your CV on
www.employerforlife.com
Click on the ad to read more

Download free eBooks at bookboon.com

In contrast, for the Missionaries and Cannibals model, the behaviour selection process is more apparent. In the model's `expand-paths` procedure shown in NetLogo Code 8.8, there are five "paths" being expanded. These correspond to the alternative actions of having two missionaries get in the row-boat, two cannibals in the rowboat, one missionary and one cannibal get in the row-boat, one missionary by itself and one cannibal by itself. According to the definition of behaviour described in Section 6.1, these actions can be considered as separate behaviours. To further emphasize the behavioural aspect of these actions, let us imagine, for example, a scenario where the missionaries exhibit the peculiar behaviour of never getting in the row-boat by themselves. We can simulate this by simply commenting out the "`expand-path searcher-agent 1 0`" line in the `expands-path` procedure. This results in significantly different searches being performed as a result for each of the search methods, with some of the searches failing while others, such as breadth-first search, perhaps surprisingly, still managing to be successful at finding a solution, albeit different to the previous one.

```
to expand-paths [searcher-agent]
;; expands all the possible paths for search-agent

   expand-path searcher-agent 2 0 ; two missionaries in row-boat
   expand-path searcher-agent 0 2 ; two cannibals in row-boat
   expand-path searcher-agent 1 1
   ; one missionary and one cannibal in row-boat
   expand-path searcher-agent 1 0 ; one missionary in row-boat
   expand-path searcher-agent 0 1 ; one cannibal in row-boat
end
```

NetLogo Code 8.8 The procedure defining how the paths are expanded for the Missionaries and Cannibals model.

8.6 Informed search

Informed search occurs when the agent uses problem-specific information to help guide the search. The information is used to determine which of the paths should be taken when there are alternatives. One form the information can take is that of the agent using a heuristic evaluation function that ranks alternative paths. Usually the path with the lowest value according to the heuristic function is chosen to expand the search next. The purpose of the heuristic function is to make an informed guess as to which path is likely to be the best. The agent needs to guess because if the agent already knew which path was the best in advance, then there is no longer any need to perform any search.

However, what is a good heuristic function? The function should provide an estimate of the cost of the cheapest path from the current node to the goal node. For example, one possible heuristic for the Searching for Kevin Bacon model is straight-line (Euclidean) distance if we are measuring cost in terms of distance travelled in the environment. A feature of this heuristic is that it will never overestimate the actual cost since the shortest path between two points is always a straight line.

Download free eBooks at bookboon.com

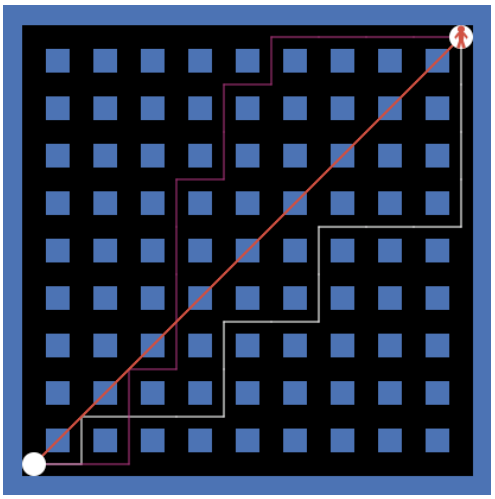


Figure 8.11 Screenshot of Manhattan Distance model.

A heuristic function is said to be *admissible* if, as with Euclidean distance, it never overestimates the cost of reaching the goal node from the current node. Another heuristic function, when we are measuring cost in terms of distance moved in the environment, is Manhattan distance. Here the analogy is of moving between two points as if we are constrained to moving along streets in Manhattan – we can only move vertically and horizontally but never diagonally. Figure 8.11 shows a screenshot of the Manhattan Distance model in NetLogo that illustrates how the measure is calculated and how it compares with Euclidean distance. The length of the diagonal red line is the Euclidean distance that in the screenshot is equal to $\sqrt{(18^2+18^2)}=25.46$. The Manhattan distance can be calculated based on the distance of the many different combinations of horizontal and vertical paths that lead from the start point (bottom white circle) to the goal (top right circle). There are two possibilities shown in the screenshot, but all such paths will have the same overall distance of 36. The Manhattan distance is not an admissible heuristic for the Searching for Kevin Bacon model (in cases where two points are not vertically or horizontally apart from each other, the Manhattan distance is greater than the shortest distance between two points which is the Euclidean distance). However, it is an admissible heuristic for the Searching Mazes model because the movement in the three mazes provided by the model (the empty maze, the Hampton Court Palace maze, and the Chevening Court maze) is always in either a horizontal or vertical direction, so the length of the shortest traversable path between two points in the maze is always the Manhattan distance.

The code shown in NetLogo Code 8.9 shows how the heuristic function is calculated for the Searching for Kevin Bacon model using either Euclidean distance or Manhattan distance. A third heuristic function called Zero, which always returns the value 0, has also been included in the models for comparison purposes. The Zero function is an admissible heuristic since it clearly never overestimates the path cost.

Download free eBooks at bookboon.com

```
to-report euclidean-distance [x y x1 y1]
;; reports the euclidean distance between points (x,y) and (x1,y1)
   report sqrt ((x1 - x) ^ 2 + (y1 - y) ^ 2)
end

to-report manhattan-distance [x y x1 y1]
;; reports the euclidean distance between points (x,y) and (x1,y1)
 report abs (x1 - x) + abs (y1 - y)
end

to-report heuristic-function [x y]
;; reports the heuristic evaluation function value

   let goalx kevin-bacon-xcor
   let goaly kevin-bacon-ycor

   if (heuristic = "Zero")
      [ report 0 ]
   if (heuristic = "Euclidean distance")
      [ report euclidean-distance x y goalx goaly ]
   if (heuristic = "Manhattan distance")
      [ report manhattan-distance x y goalx goaly ]
end
```

NetLogo Code 8.9 The heuristic function used for the Searching for Kevin Bacon model.

Download free eBooks at bookboon.com

Devising a suitable heuristic function is more of an art than a science. We want the heuristic to provide a good ranking of alternative paths, but at best it is just an educated guess based on some prior knowledge of what works well within a particular search domain. Clearly, the Zero function will not provide a good ranking of alternative paths as the cost will be always be underestimated (unless the goal node has been reached) and every path will be ranked the same. (It is, however, useful for providing a baseline comparison since when used in the greedy best first search described below, the search behaviour defaults to random expansion of the frontier as all alternative paths are ranked equally).

Heuristic functions need to be tailored to each search problem. What may work well in one problem domain may not work so well in another. For example, the Manhattan distance heuristic is suitable for the Searching Mazes problem but would not be suitable for the Searching for Kevin Bacon problem for reasons outlined above. Neither the Manhattan distance heuristic or the Euclidean distance metric would be suitable for the Missionaries and Cannibals problem in its present form since the previous solutions were only suitable for 2-dimensional space, and as described above and illustrated in Figure 8.6, the Missionaries and Cannibals problem is more naturally represented in a 3-dimensional search space, where the dimensions are the number of missionaries on the left side of the river, the number of cannibals on the right side of the river and whether the row boat is on the left side or not. We can readily extend the Euclidean distance and Manhattan distance functions to include 3 (or more) dimensions as shown in NetLogo Code 8.10.

```
to-report euclidean-distance [x y z x1 y1 z1]
;; reports the euclidean distance between points (x,y,z) and (x1,y1,z1)
 report sqrt ((x1 - x) ^ 2 + (y1 - y) ^ 2 + (z1 - z) ^ 2)
end

to-report manhattan-distance [x y z x1 y1 z1]
;; reports the manhattan distance between points (x,y,z) and (x1,y1,z1)
 report abs (x1 - x) + abs (y1 - y) + abs (z1 - z)
end

to-report heuristic-function [mcount ccount rcount]
;; reports the heuristic evaluation function value

 if (heuristic = "Zero")
 [ report 0 ]
 if (heuristic = "People on the left side")
 [ report mcount + ccount ]
 if (heuristic = "Min. no. boat trips needed")
 [ report (mcount + ccount) / 2 ] ; 2 is the capacity of the boat
 if (heuristic = "Euclidean distance") ; goal is point (0,0,0)
 [ report euclidean-distance mcount ccount rcount 0 0 0 ]
 if (heuristic = "Manhattan distance") ; goal is point (0,0,0)
 [ report manhattan-distance mcount ccount rcount 0 0 0 ]
end
```

NetLogo Code 8.10 The heuristic function used for the Missionaries and Cannibals model.

Download free eBooks at bookboon.com

Also defined are two further admissible heuristic functions that can be used for this problem. The first, labelled "People on the left side", simply counts the number of people that are on the left side of the river. This is related to the second, labelled "Min. no. boat trips needed", which works out the minimum number of boat trips that would be needed to transfer the people across. These two are unrealistic estimates of the true path cost because they ignore the added trips needed firstly to ensure that the cannibals never outnumber the missionaries at any stage and secondly to return the canoe to pick up the remaining people. However, they clearly never overestimate the number of boat trips needed so they are admissible heuristics.

We can now look at how we can use these heuristic evaluation functions as the basis of different search behaviours. "Best-first" search refers to searches that extend the "best" paths first as judged by a heuristic function. One type of best-first search, called *greedy best first* search, orders the expansions of the search paths by the estimated cost to the goal. The analogy is with the searcher agent being "greedy" as well as short-sighted by wanting to pick what seems to be best at the moment, hence the name. The code for the search is defined in NetLogo Code 8.11 below (the estimated cost is calculated by the heuristic function in the `expand-paths` procedure in NetLogo Code 8.1.). The code for this search is very similar to the code for depth-first search in NetLogo Code 8.2, the only difference being that the paths are ordered by `estimated-cost` rather than time.

```
to expand-greedy-best-first-search
   ; expand the search using greedy best first search method

   set path-found false
   ask first (sort-by [[estimated-cost] of ?1 <
                       [estimated-cost] of ?2] searchers)
   [
      expand-paths (searcher who)
      die ; this agent has done its work; it's children are now doing the work
   ]
end
```

NetLogo Code 8.11 How the Greedy Best First Search is defined.

A simple modification to greedy best first search is to re-order the search by the sum of both the estimated cost and the path cost. This search is called A^* search (pronounced "A star"); the code is defined in NetLogo Code 8.12.

Download free eBooks at bookboon.com

```
to expand-A-star-search
   ; expand the search using A* search method

   set path-found false
   ask first (sort-by [[estimated-cost + path-cost] of ?1 <
                       [estimated-cost + path-cost] of ?2] searchers)
   [
      expand-paths (searcher who)
      die ; this agent has done its work; it's children are now doing the work
   ]
end
```

NetLogo Code 8.12 How the A* Search is defined.

Figure 8.12 shows how both the greedy best first search and A* search searches the network shown earlier in Figures 8.3 and 8.4 for the Searching for Kevin Bacon model. In this case, both searches end up following the same paths. However, for more complicated networks, this is generally not the case, with A* usually taking a more direct route to the goal than greedy best first search.

Click on the ad to read more

Download free eBooks at bookboon.com

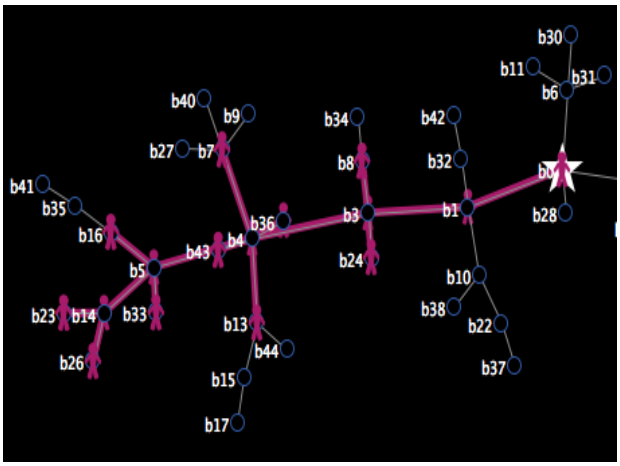


Figure 8.12 Screenshot of both the greedy best first search and A* search for the Searching for Kevin Bacon model.

This is shown by the two searches on the empty maze for the Searching Mazes model in Figure 8.13. In this setup, the heuristic function is Euclidean distance, and the agent progresses forward 2 steps at a time as was the case for Figure 8.5. The left image shows the slightly more erratic general path taken towards the exit by the searcher agents for the greedy best first search, but overall, the searchers are 'driven' toward the goal by the heuristic function ranking better the positions nearer the goal. Similar behaviour can be observed when the Manhattan heuristic function is used rather than Euclidean distance. When the Zero function is used as the heuristic, however, the searcher agents expand outwards in a random fashion, not necessarily towards the exit.

The right image shows the more direct and ordered progression taken by the A* searcher agents when the Euclidean is used. The five middle columns of the army of searcher agents march directly from the bottom of the maze towards the exit at the top. However, they suddenly stop a few steps away from the exit, and the left outer column then moves forward, with the right outer column joining the search not long after. The reason for this is the way the positions are being ranked by the sum of the path cost combined with the estimated cost calculated using Euclidean distance. For the middle columns, this combined sum is slightly greater then the vertical length of the maze since the direct line to the exit is vertical, and the goal position used for the Euclidean distance calculation is a single point to the left of the exit. This sum for the outer column exceeds the sum for the inner columns for most of the time while the agents are marching forward, but as the inner column agents get very near the goal, this is no longer true, and so the outer column agents take over. Eventually, the middle columns reach the exit.

Download free eBooks at bookboon.com

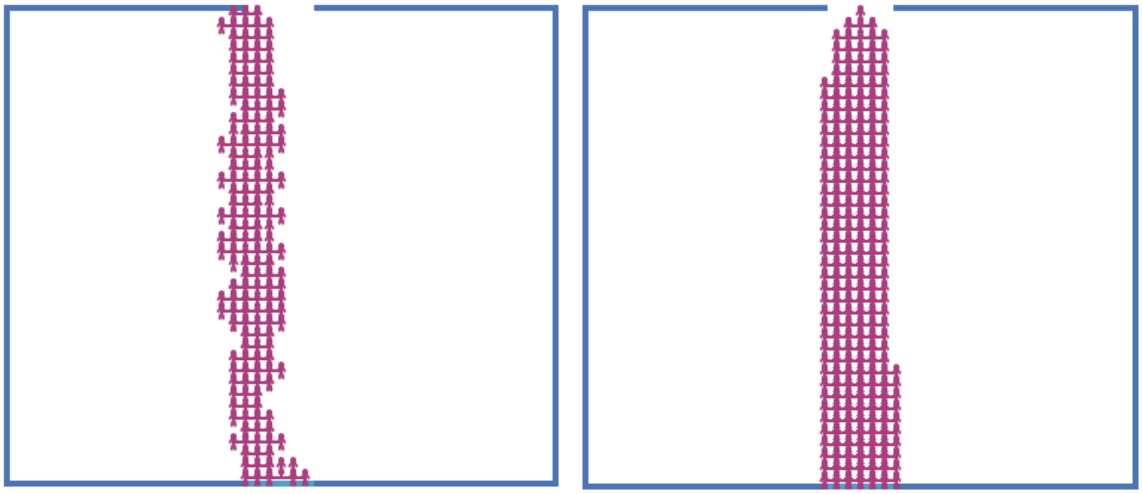


Figure 8.13 Screenshots of the greedy best first search (left image) and A* search (right image) for the Searching Mazes model on the empty maze using the Euclidean heuristic.

Figure 8.14 shows what happens with the A* search on the empty maze using the Manhattan distance heuristic (left image) and the Zero distance heuristic (right image). The behaviour of the searcher agents with the Manhattan distance heuristic is to march towards the goal point on the left side of the exit, but the movement forward is patchier than for the Euclidean distance metric.

The *x* co-ordinate for the goal point for the case shown in the image is set to -5 within the code during setup, so the leading searching agent at the tip of the marching column of agents tends to have the same *x* co-ordinate. In contrast, the behaviour of the searcher agents when the Zero distance heuristic is used is similar to the behaviour of breadth-first search.

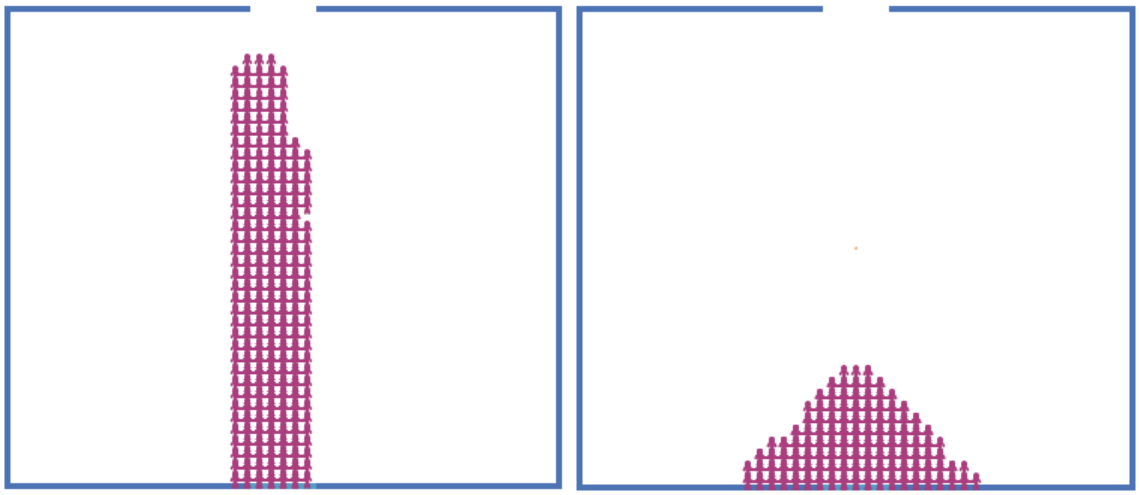


Figure 8.14 Screenshots of the A* search using the Manhattan distance heuristic (left image) and the Zero distance heuristic (right image) for the Searching Mazes model on the empty maze.

Download free eBooks at bookboon.com

8.7 Local search and optimisation

Another family of search behaviours is based on 'local' search where the agents performing the search apply rules according to the local situation they find themselves in by examining the alternative paths only in relation to the immediate neighbourhood. Therefore they do not need to record any information such as which locations they have already visited, the paths they have taken or the path cost. This type of search can be described using the analogy of a myopic person with a very localised field of vision. The search uses a meta-heuristic that can be applied to solving general problems, rather than a specific heuristic tailored for the problem domain.

One form of this type of search called *hill climbing* (also called *greedy local search*) is to make the choice of where to move to based on maximising (or equivalently minimising) some criteria. The code for this search is shown in NetLogo Code 8.13.

Click on the ad to read more

Download free eBooks at bookboon.com

```
to expand-hill-climbing-search
   ; expand the search using hill-climbing search method

   set path-found false
   ask searchers ; there should always be only one searcher at this stage
   [ expand-paths (searcher who) ] ; look to see where I can go next

   foreach but-first (sort-by [[height] of ?1 < [height] of ?2] searchers)
   [ ; kill off all but the first
   ask ? [ die ]; only let the best of the new searchers continue
   ]
end
```

NetLogo Code 8.13 How the Hill Climbing Search is defined.

With hill climbing search, only one searcher agent is active at any time. This agent will create new searcher agents for each path it can expand, but then only one of those is chosen to continue the search, and the rest are killed off. The searcher agent chosen will be the one that has the minimum value for the `height` variable stored with each agent. This is calculated in the `expand-paths` procedure (see NetLogo Code 8.1). For the Searching Mazes module and the Searching for Kevin Bacon model, the height is calculated using the heuristic function defined in NetLogo Code 8.14:

```
   to-report hill-height [x y]
   ;; reports the "height" of the current search position
   ;; the zero height is the goal

   report heuristic-function x y
end
```

NetLogo Code 8.14 How the height is calculated for the Searching mazes and Searching for Kevin Bacon models.

Using Euclidean distance or Manhattan distance from the goal point as the meta-heuristic for both models ensures that the landscape of the environment is smooth throughout, and consists of a single depression, with the lowest point (zero) being the epicentre. The hill-climbing search then tries to move the agent in a direction towards the epicentre.

Figure 8.15 provides two screenshots of hill-climbing search on the empty maze for the Searching Mazes model. The left image shows what happens when Euclidean distance is used as the meta-heuristic, the right image when Manhattan distance is used. The explanation for the pronounced veering away of the columns of searcher-agents at the beginning and end is that, as discussed above, the x co-ordinate of the goal point is set at -5 which is slightly to the left of the x co-ordinate of the first searcher agent when it enters the maze. Therefore, for Euclidean distance, the agent expanded in the forward direction will always be closer to the goal point until right at the end (in contrast to the two other agents expanded in the left and right directions). For Manhattan distance, the veering occurs at the beginning instead because of the way the distance is calculated as a summation of vertical and horizontal lengths rather than diagonally.

Download free eBooks at bookboon.com

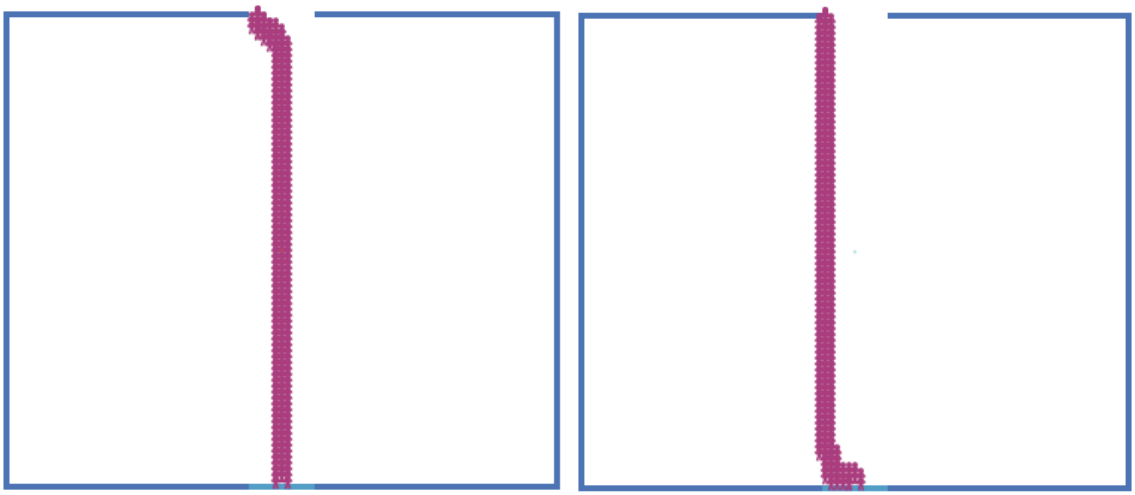


Figure 8.15 Screenshots of the hill climbing search using the Euclidean distance meta-heuristic (left image) and the Manhattan distance meta-heuristic (right image) for the Searching Mazes model on the empty maze.

For the Missionaries and Cannibals model, we need another method for calculating the height variable since the search space is three-dimensional. Referring to Figure 8.6, one obvious way for representing height in the environment is the position of each state on the parallel *y* axes. For example, the highest point is the start state at (3, 3, 1), and the lowest (zero) point is the goal state at (0, 0, 0). We therefore can calculate the height proportionately to reflect the number of missionaries and cannibals, and the current row-boat position as shown in NetLogo Code 8.15.

```
to-report hill-height [mcount ccount rcount]
;; reports the "height" of the current search position
;; the zero height is the goal; the maximum height is the start

  report mcount * 8 + ccount * 2 + rcount
end
```

NetLogo Code 8.15 How the height is calculated for the Missionaries and Cannibals model.

A problem with hill climbing search is that it can easily get stuck in local maxima (or minima if we are searching for a minimum as in NetLogo Code 8.10). Another problem is when there is an obstacle blocking the way. This is illustrated by the screenshot in Figure 8.16 of hill climbing search used on the Hampton Court Palace maze. The search heads directly in the direction to the global minimum at the centre of the maze, but then gets prevented from moving forward by the long horizontal wall in the way. What the search needs to do is head in a non-minimising direction to get around the obstacle. One way of doing this is to include a random walking behaviour when the search has not made any progress. However, the Hampton Court Palace Maze environment is still problematical even for such a solution, as the walls present a series of obstacles parallel to each other that are very wide and difficult to find a way around.

Download free eBooks at bookboon.com

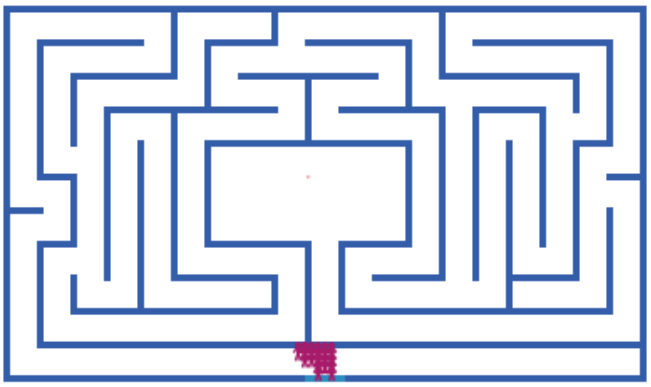


Figure 8.16 Screenshot showing how Hill Climbing Search gets stuck for the Searching Mazes model on the Hampton Court Palace Maze.

Similarly, hill-climbing search used for the Searching for Kevin Bacon model can also get stuck in nodes where adjacent nodes are all in directions that are further away from the goal node. This occurs in the screenshot show in Figure 8.17 where the search starts at node `b33`, but then cannot proceed beyond node `b21` because the distances to the goal node (the white star) from adjacent nodes is longer than the distance from node `b25`.

Click on the ad to read more

Download free eBooks at bookboon.com

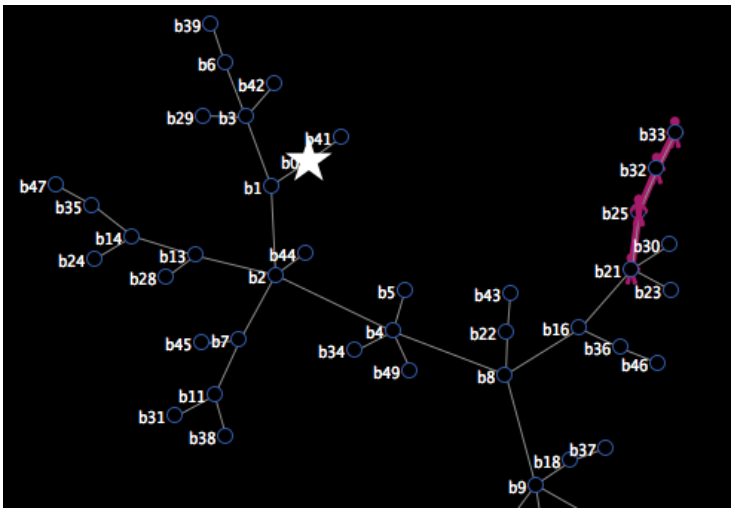


Figure 8.17 Screenshot showing how the Hill Climbing Search get stuck in some cases for the Searching for Kevin Bacon model.

The same problem occurs for hill-climbing search on the Missionaries and Cannibals problem. The search very quickly gets stuck when the only future paths all head away from the goal state.

Hill climbing is an example of a broader class of problem solving called *optimisation*. Optimisation methods rely on making some form of guess, and then incrementally refining the guess until no further refinements are possible. Some other forms of optimization are simulated annealing, beam search, genetic search and particle swarm optimization. For the latter two, two models are provided with the NetLogo Models Library – Simple Genetic Search and Particle Swarm Optimization. (These models will be further discussed in Volume 2 of this book series.) Particle swarm optimisation search make use of swarm intelligence and stigmergy. The use of the blackboards by communicating agents to help search computer networks within the Being Kevin Bacon model described in the previous chapter is another example of distributed agents using stigmergic local information to help improve search. The word of mouth method of communication also implemented in the model is another example of local search, but unlike the blackboard method, does not make use of stigmergy.

8.8 Comparing the search behaviours

How can we compare the different search behaviours described above? In other words, what are some suitable criteria we can use in order to make an objective evaluation as to which search behaviour is 'better'? One obvious criterion is to determine whether the search is completed or not. In some cases, we cannot be sure that the search is guaranteed to find a solution. Another criterion is whether the search finds an optimal solution – that is, the solution that has the lowest path cost along the path from the start state to the goal state. Further criteria relate to the time and memory required to find a solution. Table 8.1 lists several criteria for evaluating search performance. The first four criteria are from Russell and Norvig (2002). The last criterion was proposed by Yao (1979) and relates to the amount of communication required between the distributed agents as they are executing the search.

Download free eBooks at bookboon.com

Criterion	Description
Completeness.	Is the search guaranteed to find a solution if there is one?
Optimality.	Does the search find the optimal solution?
Time complexity.	How much total time for all agents does it take to find a solution?
Space complexity.	How much total memory for all agents is required to find a solution?
Communication complexity.	How much communication between agents is necessary to find a solution?

Table 8.1 Criteria for evaluating search performance.

Time and space complexity are usually measured in relation to the lengths of the paths searched. If search is characterised using distributed agents as implemented by the code in this chapter, then time complexity relates to the total number of searcher agents created during the search, and space complexity relates to the maximum number of searcher agents that are 'active' during the execution of the search. For each of the three NetLogo models described in this chapter (Searching for Kevin Bacon, Searching Mazes and Missionaries and Cannibals), these two numbers are shown in the Interface monitors Maximum `Active Searcher Agents` and `Total Searcher Agents`. We can use these monitors to do an experimental comparison of the different search behaviours as shown in Table 8.2.

Click on the ad to read more

Download free eBooks at bookboon.com

Search problem	Search behaviour	Revisits allowed?	Success	Maximum Active Searchers	Total Searchers
Missionaries & Cannibals	Breadth First Search	✓	✓	5015	9239
		✓	✓	3	16
	Uniform Cost Search	✓	✓	7501	13032
		×	✓	2	16
	Depth First Search	✓	×	–	–
		×	✓	3	16
	Multi-Agent Depth First Search (max-agents-to-expand = 2)	✓	×	–	–
		×	✓	3	16
	Greedy Best First Search (heuristic = Euclidean)	✓	×	–	–
		×	✓	3	16
	A* Search (heuristic = Euclidean)	✓	✓	1035	1990
		×	✓	2	16
	Hill Climbing Search	✓	×	–	–
		×	×	–	–
Searching Mazes	Breadth First Search	✓	✓	2943	5095
on empty maze;		×	✓	100	469
both move forward behaviours = "Move forward n steps unless hit wall"	Uniform Cost Search	✓	✓	2660	4603
		×	✓	114	448
	Depth First Search	✓	✓	*257*	*466*
		×	✓	*168*	*358*
move-forward- step-amount = 10	Multi-Agent Depth First Search (max-agents-to-expand = 2)	✓	✓	*445*	*754*
		×	✓	*167*	*361*
	Greedy Best First Search (heuristic = Euclidean)	✓	✓	*38*	*61*
		×	✓	*24*	*40*
	A* Search (heuristic = Euclidean)	✓	✓	*43*	*67*
		×	✓	*41*	*64*
	Hill Climbing Search	✓	✓	1	25
		×	✓	1	25

Table 8.2 Comparing the performance of the search behaviours.

The table lists results for various searches for two of the models – Missionaries and Cannibals, and Searching Mazes. For the latter, the empty maze was used as the environment, and the Interface variables were set as shown in the first column of the table.

Download free eBooks at bookboon.com

The second column lists the search behaviour that was tried out along with any relevant Interface variables such as the heuristic that was used for the informed searches. The third column indicates whether states were allowed to be revisited (✓) or not (×), the fourth column lists whether the search completed (✓) or not (×), and the final two columns lists the maximum number of active searcher agents and the total number of searcher agents that were used throughout the search. For these last two numbers, the ones shown in italic font indicate that these are typical values only, since results vary from one simulation to the next because of the way the NetLogo `ask` command operates, randomly choosing between agents when breaking ties.

The table can be used to get a rough idea of the relative performances of the different search behaviours. Clearly, not allowing revisits significantly improves all searches, and in some cases allows the search to complete since the search no longer gets stuck revisiting the same nodes over and over again (as for the depth-first, multi-agent depth-first and greedy best-first searches in the Missionaries and Cannibals model). Another noticeable trend is the relative expense in terms of time and space complexity for the breadth-first and uniform cost search compared to the other searches. A* is also expensive for the Missionaries and Cannibals problem when revisits are allowed, but does relatively well on the empty maze due to the heuristic being ideally suited to the nature of the problem. The other searches do relatively well on the empty maze, but for the Missionaries and Cannibals model suffer the problem of not completing the search in the `max-depth` cut-off time period that was imposed when revisits are not allowed.

Table 8.2 provides useful experimental evidence to help compare the effectiveness of the searches, but a more theoretical analysis is required to gain further insight into the relative merits of each. We can use computational complexity theory to estimate the time and space complexities for the different search behaviours as shown in Table 8.3. The third and fourth columns list the time and space complexities – for further details, see Russell and Norvig (2002). The second and third columns in the table indicate whether the search behaviour leads to a complete search and whether the search will always find the optimal solution. (Note, however, Russell and Norvig provide several caveats concerning completeness and optimality for the breadth-first, uniform-cost and iterative deepening searches).

Download free eBooks at bookboon.com

Search Behaviour	Complete?	Optimal?	Time	Space
Breadth-first	✓	✓	$O(b^{d+1})$	$O(b^{d+1})$
Uniform cost	✓	✓	See note (i).	See note (i).
Depth first	×	×	$O(b^m)$	$O(bm)$
Depth-limited	×	×	$O(b^l)$	$O(bl)$
Iterative Deepening	✓	✓	$O(b^d)$	$O(b^m)$
Greedy best first	×	×	$O(b^m)$	$O(b^m)$
A*	✓	✓	See note (ii).	See note (ii).
Hill climbing	×	×	See note (iii).	$O(1)$

Table 8.3 Criteria for evaluating search performance (Russell and Norvig, 2002).

b is the branching factor; *d* is the length (in nodes) of the shortest path that is a solution; m is the length of the maximum path searched; l is the cut-off path length beyond which the search does not continue. Note (i): The search complexities are related to path costs rather than number of nodes in the path; if path costs are constant, then time and space complexities are both $O(b^d)$. Note (ii): Exponential growth in the search tree if the error in the estimate made by the heuristic function grows faster the logarithm of the path cost. Note (iii): Time complexity relates to the state space landscape.

Note that in many cases, these complexities are an idealisation of the search or may not reflect actual performance. For example, the time and space complexities for greedy best first search is based on a worst-case analysis – with a good heuristic, the search can be reduced significantly. Note also that in some cases, these complexities do not correspond to the different search behaviours as they have been implemented in this chapter. The use of the `sort-by` reporter in many of the expand search procedures – for example in `expand-depth-first-search` shown in NetLogo Code 8.2 – is not the most efficient implementation possible. The reason they have been implemented this way is to draw attention to the essential differences between the various searches by casting them within a common framework.

Communication complexity is also an important factor that needs to be taken into consideration, especially if multi-agent solutions are being used. If there is a cost involved in communicating the state of the search between agents – for example when the agents clone themselves as above – then this should be factored into the evaluation as well. The communication complexity is related to the cost of transmitting the information between agents; this can be measured using entropy as described in the previous chapter (i.e. using compression code lengths). All of the above search behaviours have similar communication complexities to the spaces complexities because information of a constant size is being transmitted from the parent agent to the clone agents – that is, when the `hatch-searchers command` is called as in NetLogo Code 8.1, the parent agent transmits the same constant-sized information to the newly hatched clone (for example, the current `path-cost` and time of the search). Communication complexity, however, can be an important factor that needs to be considered in other types of search, such as the word of mouth and blackboard type searches described in the previous chapter.

Download free eBooks at bookboon.com

The various complexities are also related to the information an agent or agents collect while performing a search, as well as the information transmitted between agents. Obvious information an agent can collect is whether the agent has already visited a particular state or been down a particular path. Further information might be whether one state leads to another state, or the shortest path from a given state to another state. The agent can use this information to guide the search but the cost of processing the information most also be factored into the complexities. Also, sometimes the cost of obtaining the information might be too high – it might mean that exploring the entire search space is required. Alternatively, keeping the information may also be too expensive in terms of external storage.

8.9 Summary and Discussion

One of the greatest joys known to man is to take a flight into ignorance in search of knowledge.

Robert Lynd.

The standard search algorithms covered in this chapter, such as depth-first, breadth-first, greedy best first and A* search, have been implemented using an embodied, situated multi-agent framework. In this framework, search is recast as a process performed by teams of agents co-operating together in order to find a desired location in an n-dimensional space.

Click on the ad to read more

Download free eBooks at bookboon.com

Many of the higher-level behaviours often associated with intelligence can also be recast as search in this manner. Problem solving can be thought of as a search in solution space – there are many possible solutions to a particular problem, and these are scattered throughout an n-dimensional space that needs to be searched in order to solve the problem. Similarly, the processes of decision-making, planning and learning can be recast as searches of n-dimensional spaces comprising decisions, plans and learnt concepts.

Conversations also require search – during a conversation, agents must continually search to find the most appropriate responses. The next chapter shows how reasoning can also be recast as a search problem.

Search can be considered as a behaviour that an embodied, situated agent exhibits in order to gain knowledge that it does not already have. Search is fundamentally linked to knowledge – they can be thought of as two sides of the same coin. This is reflected in the language we use to describe search. For example, a common English expression we use is "search for knowledge" (as in the quote at the start of this Section). We also often say when we do not 'know' whether something is true, that we need to 'find' out whether it is true, as in the expression "I need to find out if X is true", where X is any statement. The next chapter will explore what knowledge is, and the link between search and knowledge further.

Download free eBooks at bookboon.com

A summary of important concepts to be learned from this chapter is shown below:

- Search from an embodied, situated perspective can be thought of as a search of alternative behaviours rather than a search of alternative paths.
- Uninformed (blind) search does not use any information to guide the search. The analogy is of a blind man trying to search a maze without any knowledge or map of the maze.

The following searches are uninformed searches:

- The following searches are uninformed searches:
  depth-first – this is where only a single agent is active and often needs to back track;
  depth-limited – this is where a maximum cut-off depth is imposed on depth-first search;
  multi-agent depth-first – this is depth first search but with more than one active agent;
  breadth-first – this is where teams of agents fan out in parallel;
  uniform-cost – this is breadth-first search but lowest cost paths are expanded first;
  Iterative Deepening – this uses repeated executions of depth limited search to increasing maximum depths;

- Informed search uses information to guide the search. The information is in the form of a guess about which way is best.
- The function used to make the guess is called a heuristic function.
- An admissible heuristic function is one that never overestimates the cost of reaching the goal.
- The following searches are informed searches:

  greedy best-first – this is where the path with the lowest estimated cost to the goal (calculated by the heuristic function) is followed next;
  *A** – this is where the path with the lowest (estimated cost to the goal + path-cost) total is followed next.

- Local search and optimisation make a guess about which path to choose based on local information only.
- The following search is a local search:
  hill-climbing – this is where the single agent always heads "up" the hill.

- Hill climbing search often gets stuck in local optima.
- The following criteria can be used to evaluate searches: completeness, optimality, time complexity, space complexity and communication complexity.

The code for the NetLogo models described in this chapter can be found as follows:

Model	URL
Missionaries and Cannibals	http://files.bookboon.com/ai/Missionaries-and-Cannibals.nlogo
Searching Mazes	http://files.bookboon.com/ai/Search-Mazes.nlogo
Searching for Kevin Bacon	http://files.bookboon.com/ai/Searching-for-Kevin-Bacon.nlogo

Model	NetLogo Models Library (Wilensky, 1999) and URL
Particle Swarm Optimization	Computer Science > Particle Swarm Optimization http://ccl.northwestern.edu/netlogo/models/ParticleSwarmOptimization
Simple Genetic Algorithm	Computer Science > Simple Genetic Algorithm http://ccl.northwestern.edu/netlogo/models/SimpleGeneticAlgorithm

Download free eBooks at bookboon.com

9 Knowledge

The question how knowledge should be defined is perhaps the most important and difficult of the three [(1) the definition of knowledge, (2) data, (3) methods of inference] with which we shall deal. This may seem surprising: at first sight it might be thought that knowledge might be defined as belief which is in agreement with the facts. The trouble is that no one knows what a belief is, no one knows what a fact is, and no one knows what sort of agreement between them would make a belief true.

Bertrand Russell. 1926. Theory of Knowledge for the Encyclopaedia Britannica.

This chapter looks at the topic of knowledge. The chapter is organised as follows. Section 9.1 provides a definition of knowledge and suggests some design principles for knowledge-based systems. Section 9.2 discusses how knowledge can be defined as justified true belief, and highlights some problems with this definition. Section 9.3 describes different types of knowledge such as declarative knowledge and procedural knowledge.

Section 9.4 discusses several approaches to the representation of knowledge – the symbolic approach based on the processing of symbols, the sub-symbolic approach based on the processing of stimuli, and a hybrid approach called conceptual spaces based on the processing of concepts. Section 9.5 lists some knowledge engineering problems, and describes three classification problems used for testing out the knowledge representation and reasoning methods described in Sections 9.7 to 9.13. Section 9.6 discusses whether we can have knowledge without representation. Sections 9.7 to 9.13 discuss how we can represent knowledge and perform reasoning using: maps; event maps; rules and logic; frames; decision trees; and semantic networks.

9.1 Knowledge and Knowledge-based Systems

Knowledge is essential for intelligent behaviour. Without knowledge, an intelligent agent cannot make informed decisions, and instead must rely on using some form of searching type behaviour involving exploration and/or communication in order to gain the missing knowledge. Humans rely on knowledge every moment of their life – knowledge of how to communicate with other humans, knowledge of where they and other people live and work, knowledge of where things are, knowledge of how to behave in different situations, knowledge of how to perform different tasks and so on. Without the ability to store and process knowledge, the cognitive abilities of a human is seriously curtailed. An illness such as Alzheimer's, for example, can be debilitating when memory loss occurs such as the difficulty in remembering recently learned facts.

Download free eBooks at bookboon.com

We all know (or think we know) what we mean when we use the term 'knowledge'. But what exactly is knowledge? Bertrand Russell (1926) acknowledged the difficult question of how to define the meaning of 'knowledge':

"*It is perhaps unwise to begin with a definition of the subject, since, as elsewhere in philosophical discussions, definitions are controversial, and will necessarily differ for different schools*".

A definition of knowledge is the subject of ongoing philosophical debate and presently there are many competing theories with no single definition universally agreed upon. Consequently, treatment of knowledge from an Artificial Intelligence perspective has often consciously avoided the definition of what knowledge is. However, this avoidance of providing a definition of knowledge upfront results in a lack of preciseness in the literature and research. The following argument will illustrate why. A knowledge-based system is a term used in Artificial Intelligence to refer to a system that processes knowledge in some manner. We can make the analogy of a knowledge-based system as being a repository of knowledge, whereas a database is a repository of data. However, in this definition, we have neglected to define the meaning of the term 'knowledge' and how it is different to data. For example, we can ask ourselves the following question – "What constitutes a knowledge-based system, and how does it differ from a database system?" This is a difficult question that cannot readily be answered in a straightforward way.

American online
LIGS University
is currently enrolling in the
Interactive Online BBA, MBA, MSc,
DBA and PhD programs:

- enroll by September 30th, 2014 and
- save up to 16% on the tuition!
- pay in 10 installments / 2 years
- Interactive Online education
- visit www.ligsuniversity.com to find out more!

Note: LIGS University is not accredited by any nationally recognized accrediting agency listed by the US Secretary of Education.
More info here.

Click on the ad to read more

Download free eBooks at bookboon.com

A common approach taken in the literature is that a knowledge-based system can perform reasoning using some form of inferencing (whether rule-based, frame-based; see below). Modern database systems, however, now employ most of these standard 'knowledge-based' techniques and more. Clearly, the addition of inferencing capabilities alone is not sufficient to define what a knowledge-based system is. However, a great deal of A.I. literature makes such an assumption.

By avoiding a definition of knowledge, the problem becomes that it is no longer clear that what we are building really is in fact 'knowledge-based'. In Chapter 1, it was stated that early A.I. systems in the 1970s and 1980s suffered from a lack of evaluation – there was a rush to build new systems, but often very little evaluation was undertaken of how well the systems worked. Without a working definition of knowledge, the same problem occurs now with current knowledge-based systems – how can we evaluate how effective our knowledge-base system might be if we do not have a definition of what it should be (or even achieve or do)?

We can, however, avoid the philosophical pitfalls, and rather than attempting to define knowledge, and making a claim that this definition is the "right" one, instead we can propose design principles for our knowledge-based system. Hence, we can decide what principles we wish our knowledge-based system to adhere to, and we, as designers, are free to change them as we see fit based on knowledge we gain during the design process. Also, we are no longer standing on shaky ground in the sense that we do not have to provide one particular definition of knowledge which is open to philosophical debate, although we are still open to criticism about whether our principles are worthwhile from an engineering perspective (i.e. whether they produce "good" programs, or aren't as good as other approaches). But evaluation becomes much simpler – all we need to do is evaluate whether our design principles are met.

The following are some design principles for knowledge-based systems.

Design Principle 9.1: A knowledge-based system must be an agent-oriented system.

The knowledge-based system must be an agent-oriented system which adheres to the following agent design principles – it is autonomous; it is reactive; it is proactive (at least).

The argument for this design principle is that if we design from an embodied, situated agent perspective, then all knowledge cannot exist independently of the agents. That is, knowledge cannot exist by itself – it can only be found in the 'minds' of the agents that are embodied and situated in an environment. We also wish to define and use the term 'knowledge' in a way similar to the way the term is used in natural language. The root of the word 'knowledge' comes from the verb "to know". From a natural language perspective, 'knowing' and 'knowledge' are related. A rock, for example, does not 'know' anything. But a dog can 'know' where it has buried a bone; and it makes sense to say in natural language that the dog has 'knowledge' of where the bone is buried. The dog, in this case, is the agent, and the rock is an object in the environment. In other words, knowing behaviour is associated with an agent who has knowledge.

Download free eBooks at bookboon.com

A knowledge-based system can then be thought of as an agent whose role is to convey the knowledge that it contains to the users of the system. The interaction between the user agent and the system agent can be characterised by the actions that determine each agent's behaviour, and whether the user agent perceives the system agent to be acting in a knowledgeable way. This leads to the next design principle.

Design Principle 9.2: A knowledge-based system must be able to answer our questions or perform a task in what we deem to be a knowledgeable manner.

The knowledge-based system must seem to be knowledgeable at the task it performs.

The following design principles are based on properties of 'good' knowledge-base systems proposed by Russell and Norvig (2002):

Design Principle 9.3: A knowledge-based system must be concise.

Parsimony should be an abiding principle.

Design Principle 9.3: A knowledge-based system must be concise.

Parsimony should be an abiding principle.

Design Principle 9.4: A knowledge-based system must be unambiguous.

The knowledge must have a clear meaning, and not open to more than one interpretation.

Design Principle 9.5: A knowledge-based system must be informed and up-to-date.

The knowledge-based system must not be out of date.

Design Principle 9.6: A knowledge-based system must strive to be as correct as possible.

The knowledge should not be erroneous, wrong, inaccurate or imprecise.

Design Principle 9.7: A knowledge-based system must be as complete as possible.

The knowledge should not be incomplete with details missing.

A behavioural approach to knowledge places the emphasis not on building a specific independent system, but on building agents that exhibit behaviour that demonstrates they have knowledge of their environment and of other agents. In this approach, the act of 'knowing' occurs when an agent has information that might potentially aid the performance of an action taken by itself or by another agent. Further, an agent can be considered to have 'knowledge' if it knows what the likely outcomes will be of an action it may perform, or of an action another agent is performing, or what is likely to happen to an object in the environment. In this definition, knowledge can be considered analogously to be the absence of the need for search; that is, the agent does not need to employ searching behaviour to find out what is likely to happen, since it already has knowledge of what is going to happen.

Download free eBooks at bookboon.com

9.2 Knowledge as justified true belief

A standard definition of knowledge, attributed to Plato, is that there are three conditions required before an agent can have knowledge – the knowledge must be *justified*, *true* and *believed*. Belief can be defined as a psychological state of an agent that holds a particular statement to be true that is not yet fully supported by the evidence. For the belief to become knowledge, the statement that the agent believes in must be true, and the agent must also be justified in their belief – in other words, they must have a reason why they think (i.e. why they 'know') a particular statement to be true. For example, an agent might believe that the capital of the United Kingdom is London, with some justification. Another agent might instead falsely believe that the capital of the United Kingdom is Edinburgh, not London. However, this is known to be not true, and there is very little justification to support the belief.

A great deal of literature – in the fields of philosophy, mathematics, in the field of Artificial Intelligence and the narrower sub-field of knowledge-based systems – often makes the pragmatic assumption that defining whether something is either true or false is a useful thing to do. In reality, however, absolutes are rare and very little of our knowledge is ever completely true, or completely false. This relates to our existence as situated, embodied agents which provides each agent with a unique perspective of the world it lives in, and which results in each agent never being in complete agreement with each other over the meaning of concepts that form the basis of the knowledge (see Section 9.4 below).

Thought Experiment 9.1 illustrates some of the problems with absolute truths.

Click on the ad to read more

Download free eBooks at bookboon.com

Thought Experiment 9.1 A walk in the woods.

Everybody lies. Cops lie. Lawyers lie. Witnesses lie. The victims lie. A trial is a contest of lies. And everybody in the courtroom knows this. Even the jury knows this.

Michael Connelly in *The Brass Verdict*.

Let us imagine the following scenario (referring to Figure 9.1). A man named Bill is standing at the foot of a large hill on the outskirts of a small city. He has gone for a hike in the woods in the early evening, and has paused for a few moments next to a clearing in the trees to look at the scenery. While he is there, he witnesses a crime down in the town, where someone breaks into a red car, which is later used in a robbery. A few months latter, he is called to give evidence in a court of law. He is asked at the beginning of questioning to tell "the truth, the whole truth, and nothing but the truth". He is expected to give truthful answers to the questions that are based on facts and not speculation.

Let us examine possible questions that might be put to him by the defence lawyer (who is defending the person accused of the crimes of robbery and car theft). Importantly, just how "truthful" can Bill be, and how easy is it for him to provide only "facts" in his answers?

Question 1: *Where in the woods were you standing*? The area he was walking in was relatively flat, and he is unsure of the exact spot. Since he was walking off the trail, he might attempt to describe where he was standing as "somewhere not far from the junction of the main trail and the smaller trail at the foot of the hill, about 200 to 300 yards away". In reality, it is more like 400 yards away, as he is a bad judge of distance. There are many hills in the area, none more distinguishable than any other, except that local people like to distinguish one particular hill since that is where the trails start next to the city and you can walk to the summit easily. The "foot" of the hill is in fact nowhere to be found, as the hill is circular, with the city spread out over the lowest point on one side of the hill, and technically, Bill was standing in the middle of the hill. The junction of the trail is also multi-faceted, with many small paths caused by thousands of people taking short-cuts over the years, so it is impossible to give a more precise distance, as it is impossible to precisely pin down in "fact" where the junction actually is (to within a dozen yards or so).

Question 2:Which clearing in the trees were you looking through? It transpires that there are many clearings, none any more particularly distinguishable from the others. Like all people who have ever provided directions to lost travellers, Bill has focused on features he thought were important for him, and ignored other features that were just as prominent (that other people might notice).

Question 3: *How long did you stop for*? There is no absolute answer to this question since time is continuous. He can answer "5 minutes", "a few minutes", "a few hundred seconds" and "not long". If a more precise answer is required, then "4 minutes 56.4" seconds might be closer to the "truth", or even more precisely, "296.4317817819… seconds". In a sense all these answers reflect the truth, and could be deemed to be truthful.

Question 4: *When did you stop*? Bill did not have his watch with him, so he cannot pin down the exact time. "Sometime in early evening, just before twilight, 30 minutes before sundown." (All three are "true").

Question 5: *How dark was it*? How dark is anything? It was the beginning of twilight, in the woods, but the city was still in sunlight. In a sense, it was "relatively" dark (in the woods), but "relatively" bright (in the city), and so he had a "relatively" good view of the proceedings he witnessed. Relatively speaking.

Question 6: *In which direction were you looking*? "Towards the city through the gap in the trees." But the city is a big place – it is difficult to pin-down the exact direction he is looking, and Bill's field of vision is quite large.

Question 7: *What colour was the car you saw being broken into*? Bill thinks the colour was red, but his definition of "redness" is subtly different to everybody else's definitions (see below), and as it was the beginning of twilight, and with the direction of the sunlight slanting down, the redness was shifted more towards red-orange.

Download free eBooks at bookboon.com

Clearly, for all of these questions, it is very difficult to give a precise answer. The court demands that only the facts of the case should be considered, but what are the "facts"? He does not know exactly where he was standing, or where he was looking, or how dark it was, or exactly what colour the car was. In this situation, how can there ever be any "facts" at all? But it is expected of him that he provide only factual answers.

There is also another important factor to be considered. The colour of the car that was stolen is described as being 'red'. But what colour is 'red'? Try describing it to a blind person. Or try playing the cars guessing game described in Section 7.8. In this game, each person in the car selects a colour they want to play with – white is a good colour, but red and blue are also good. The game is won by the person who counts the greatest number of cars going in the opposite direction with the same colour as what they have chosen. The most interesting thing about this game is the number of arguments it causes such as how long the game should be played for, whether cars 'stopped' by the side of the road count, and whether that colour was 'red', rather than 'pink' or 'purple', which do not count.

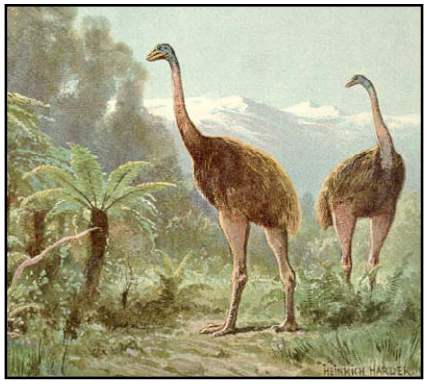
There is a very good reason why there are arguments over what colour some cars are. No two people have the exact same definition of what 'red' or 'blue' means in their minds. (See Section 9.4 for a further discussion.)

Figure 9.1 Bill's point of view.

9.3 Different types of knowledge

Rather than trying to define what knowledge is, we can alternatively try to define different types of knowledge which can sometimes be easier. For example, a cheetah and a tiger are two types of animals with distinct characteristics that are relatively easier to define compared to trying to define what an animal is. Defining different types of knowledge can help provide us with some insight into how we might build agents that exhibit knowledge (including knowledge-based systems considered as agents).

Download free eBooks at bookboon.com



New Zealand moa. Artwork by Heinrich Harder.

We can define an agent as having **declarative knowledge** if it declares that some statement is true. To determine whether specific knowledge is declarative or not, an agent can ask the following question: "*Can this knowledge be true or false*?"; or put another way: "*Is it true or false that X?*" where *X* is the statement in question. If the question is grammatical, then it can be deemed to be declarative knowledge. For example, the following question makes sense: "*Is it true or false that the New Zealand moa is extinct?*"; therefore the knowledge that the New Zealand moa is extinct is declarative. The following questions do not make sense: "*Can riding a bicycle be true or false?*" and "*Is it true or false that riding a bicycle?*" Contrast this question with the following that is grammatical and is therefore declarative knowledge if an agent knew it: "*Is it true or false that the New Zealand moa can ride a bicycle?*" In other words, when an agent declares that something is either true of false, then that knowledge is said to be declarative knowledge, and whether that knowledge is false or does not makes sense in the real world still does not stop it from being declarative knowledge.

Concerning truth (whether a declaration is either true and false), it should be noted that, in reality, absolutes are rare and very little of our knowledge is ever completely true, or completely false (as was pointed out in the previous section).

How does this definition of declarative knowledge fit with knowledge defined above as the absence of the need for search? Consider the following: an agent without prior knowledge of whether a declaration is true or false must first consult with another agent, or must explore or make observations of the environment to find out what is true or false. Both are actions that the agent must take in order to *find* the answer. The agent is said to know the answer already if it does not need to perform a searching action.

We can define an agent as having **procedural knowledge** if it knows how to perform a sequence of actions in order to ensure that a declaration *X* will become true. To determine whether specific knowledge is procedural or not, an agent can ask the following question: "What actions do I need to perform in order that I can declare that *X* is true?" If the question makes sense, then it can be deemed to be procedural knowledge. For example, the following question makes sense: "*What actions do I need to perform in order that I can declare that I am riding a bicycle is true*".

Download free eBooks at bookboon.com

The question about whether knowledge in knowledge-based systems should be declarative or procedural has been a furious debate within the field of Artificial Intelligence. As often is the case with these "Is a Platypus a mammal?" type debates, there is merit on both sides of the argument as some knowledge is inherently declarative and other knowledge is inherently procedural – for example, when humans count they are applying a procedure, but factual information concerning the names of people and the names of locations on a map is declarative. The debate has provided insight into some of the issues concerning the categorization of knowledge, but from a pragmatic design based perspective, AI researchers focus more on building useful systems, and therefore will choose the most appropriate tool for the task at hand.

Task knowledge is sometimes distinguished from procedural knowledge. Task knowledge can be considered to be a specialised form of procedural knowledge where the purpose of the actions is to solve a task (e.g. find answers to a specific question). **Behavioural knowledge** is knowledge that an agent has about the likely outcomes of behaviours (of itself and other agents). An agent has **episodic knowledge** if it knows when a statement *X* became true. An agent has **explanatory knowledge** if it can explain what caused the sequence of actions that led to the statement *X* becoming true. Finally, we can state that an agent has **inferred knowledge** if it has used existing knowledge to determine new knowledge that was not available by any other means.

Click on the ad to read more

Download free eBooks at bookboon.com

We can distinguish between the different types of knowledge by the types of questions an agent can answer correctly using their knowledge. Declarative knowledge can be used to answer "*What is…?*" and "*Where is…?*" questions; episodic knowledge can be used to answer "*When did…occur?*" questions; procedural and task knowledge can be used to help answer "*How can I/you…?*" questions; behavioural knowledge can be used to answer "*What if I/you…?*" questions; and inferred knowledge can be used to answer "*If…is true, then is…true?*" and "*What if…were true?*" questions.

We can also broaden the meaning of inferred knowledge beyond the traditional logic based definition that involves the inference of whether some statement is true or false. Consider the situation where an agent is having a conversation with another agent, such as when one person is talking to another, or perhaps when a person is talking to a chatbot. Let us say that this conversation is being observed by an outside agent (a third person, say, or perhaps a person observing the person-chatbot conversation within a Turing Test situation). This observer will be able to judge how well she thinks each of the agents have done in maintaining their side of the conversation. This will be determined by whether the observer feels that the responses are appropriate. In a sense, the observer has used her own knowledge of what is appropriate to make this judgement. The agents also have done the same thing in attempting to maintain the conversation. This can be considered to be an example of inferred knowledge. In this case, the knowledge of what is an appropriate response cannot be directly obtained by consulting some lookup table of appropriate responses since the number of language statements can be considered to be unbounded (the number of things a person can say, and the number of responses, is essentially infinite). Instead, the appropriate response must be constructed (i.e. inferred) in some manner.

9.4 Some approaches to Knowledge Representation and AI

Knowledge representation concerns the problem of how to express the knowledge in a knowledge base. A primary purpose of knowledge representation is to model intelligent behaviour with the assumption that intelligent behaviour for an agent requires knowledge of itself and other agents, of objects and their relationships, of how to solve tasks, and of laws that govern the environment the agent finds itself in.

The 'Knowledge Representation Hypothesis' (attributed to Smith, 1985) makes the supposition that for intelligent behaviour agents make use of a knowledge base that represents knowledge about the world in some manner (Brachman and Levesque, 1985). The 'Knowledge Representation Controversy' concerns how the knowledge should be represented, whether it should be by using primarily a symbol based approach, or by a non-symbolic approach.

Download free eBooks at bookboon.com

The symbolic approach is the classical approach to AI, sometimes called "Good Old Fashioned AI" or GOFAI. It is based on the 'physical symbol system hypothesis' that states that a system based on the processing of symbols has the "necessary and sufficient means for general intelligent action" (Newell and Simon, 1976). Symbols in this case are things that represent or stand for something else by association. Human language consists of symbols; for example, the word 'moa' is a symbol that represents the concept of the New Zealand bird depicted above. In the field of knowledge representation, symbols are usually denoted by an identifier in a programming language.

The symbolic approach to AI is a "knowledge-based" approach requiring the building of knowledge bases with substantial knowledge of each problem domain. It uses a top-down design philosophy consisting of several levels: the "knowledge level" at the top, which specifies all the knowledge that the system needs; the "symbol level", where the knowledge is specified in symbolic structures and identifiers in some programming language (for example, using lists or tables in NetLogo); and the "implementation level", which are the symbol processing operations that are actually implemented (Nilsson, 1998).

There are immediate problems with a symbolic approach to representing knowledge. For example, try using symbols (e.g. words) to describe the following:

The Mona Lisa, by Leonardo da Vinci.

- what the Mona Lisa painting on the right looks like (to a person blind from birth);
- the sound of bagpipes (to a person born deaf);
- the taste of milk;
- how sandpaper feels like;
- what coffee smells like.

Words seem inadequate for the task. Note that the five senses are all represented here – vision, hearing, taste, feeling, and smell; and using words to describe what we sense can be problematic – we are often left "struggling for words" is a common English expression often heard. Another common English expression says "a picture paints a thousand words". Concerning the problem of how to express a person's face using only symbols, for example the face in the painting of Mona Lisa, no amount of words seem to suffice, although the painting has been the subject of countless essays and books.

Download free eBooks at bookboon.com

An opposing approach, called 'connectionism', rejects the classical symbolic approach, and instead advocates that intelligent behaviour is the result of sub-symbolic processing – that is, the processing of stimuli rather than symbols; for example, using artificial neural networks that comprise interconnected networks of simple processing units. Here the knowledge is stored as a pattern of weights of neuron connections. This approach uses a bottom-up design philosophy or 'animat' approach by first trying to duplicate stimuli-processing abilities and control systems of simpler animals such as insects, then by trying to proceed gradually up the evolutionary ladder with increasing complexity. This approach highlights the 'symbol grounding problem' – the problem of how symbols get their meaning and postulates the 'physical grounding hypothesis' that states that the meaning of symbols need to be grounded within an agent's physical embodied experience through its interaction with the environment (Brooks, 1991a; 1991b).

The issue whether knowledge and intelligent behaviour should be represented symbolically or non-symbolically has been an ongoing debate within the field of Artificial Intelligence ever since the sub-symbolic approach to AI emerged with the revival of connectionism in the mid 1980s and with Brooks' ideas on a bottom-up, embodied, situated, behaviour-based approach to AI. As often is the case with these type of debates similar to the declarative-versus-procedural-knowledge debate mentioned above, there is merit on both sides of the argument as some tasks such as answering queries and rule-based reasoning are more naturally suited to a symbolic approach, and some types of knowledge present difficulties for either approach, such as facial recognition for symbolic processing, and natural language information for sub-symbolic processing using artificial neural networks.

Click on the ad to read more

Download free eBooks at bookboon.com

A number of other approaches have been devised, some proposing a combination of symbolic and sub-symbolic processing, such as situated automata (Kaelbling & Rosenchein, 1990) and conceptual spaces (Gärdenfors, 2004). We will examine the latter approach in more detail, as it is useful for highlighting some important issues concerning knowledge representation. Gärdenfors postulates that what is fundamental to our human cognitive abilities is our capacity for processing concepts and these emerge from a distributed connectionist representation at the lowest level where stimuli from receptors are processed, and combine to form symbolic structures at the highest level, as shown in Table 9.1.

Model level	Representation	Description
Symbolic	Propositional	Based on a given set of predicates with known denotation. Representations are based on logical and syntactic operations.
Conceptual	Geometric	Based on a set of 'quality dimensions'. Representations are based on topological and geometrical notions.
Associationist (sub-conceptual)	Connectionist	Based on a (uninterpreted) inputs from receptors. Distributed representations by dynamic connection weights.

Table 9.1. The conceptual spaces cognitive model (Gärdenfors, 2004).

Concepts form the basis of knowledge. A concept is a unit of meaning that represents an abstract idea or category. We can think of concepts as being analogous to atoms – atoms are the basic building blocks of matter, just as concepts are the basic building blocks of knowledge. In Gärdenfors approach, concepts are represented as regions in n-dimensional space, in an analogous way that a topographical map represents terrain, with similar concepts represented in geometric regions that are spatially located near to each other as shown in Figure 9.2.

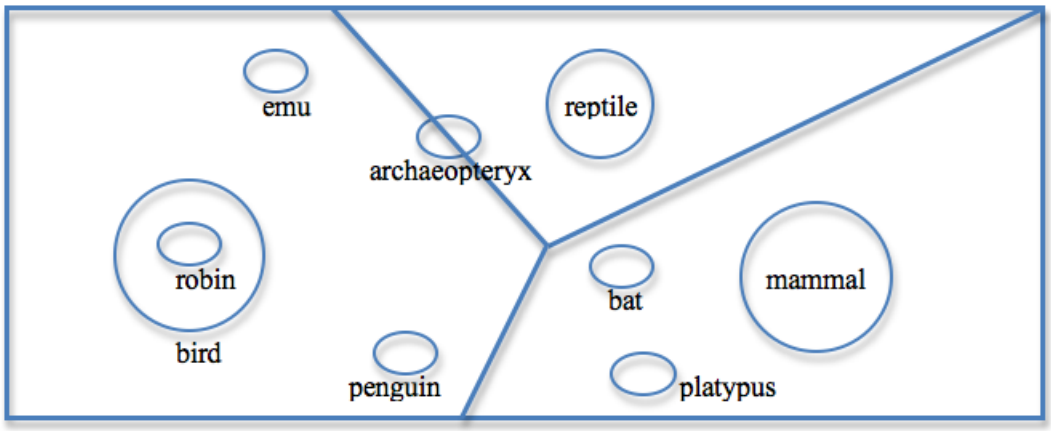


Figure 9.2 An example of concepts represented geometrically (Gärdenfors, 2004).

Download free eBooks at bookboon.com

In this example, different types of animals such as mammals, reptiles and birds, are located in different regions in the space. Gärdenfors uses a wide range of experimental evidence from many fields to support his arguments. For example, from the field of cognitive psychology, there is empirical evidence that humans make use of prototypes to provide a cognitive reference point for each concept. Gärdenfor's approach naturally lends itself to the representation of prototypes – a prototypical concept such as a robin, for example, used as a reference point for the concept of a bird, will be located centrally within the parent concept's geometric space (i.e. a bird as in Figure 9.2). Concepts that aren't used as prototypes will end up further away such as penguins and emus.

We can do a simple thought experiment to illustrate this. Think of a bird. Now which bird did you imagine? Experiments have shown that most people will think of a robin, sparrow or a similar bird rather than an emu or penguin. The mechanism Gärdenfors proposes for the construction of concepts is using a process based on Voronoi tessellation with the aid of prototypes to break the space up into convex regions as shown by the straight lines in Figure 9.2.

The multi-dimensional geometry for a conceptual space is constructed from what Gärdenfors calls 'quality dimensions' which correspond to the different ways an agent's stimuli are judged to be similar or different. The primary function of the quality dimensions is to represent various qualities or features of objects. Some examples are temperature, weight, brightness, pitch and the spatial dimensions such as height, width and depth. Gärdenfors calls these 'domains'. He uses the concept of an 'apple' to illustrate the distinction between a domain and a region:

Domain	**Region**
Fruit	Values for skin, flesh and seed type.
Colour	Red, green, yellow.
Taste	Values for sweetness, sourness, etc.
Shape	"Round" region of shape space.
Nutition	Values for sugar, vitamin C, fibres etc.

Table 9.2 A representation of the concept 'apple' (Gärdenfors, 2004).

Download free eBooks at bookboon.com

As a further example, experiments with human perception of the colour domain show that the colour conceptual space is described using three quality dimensions – brightness, hue and saturation. (Taste for humans, in comparison, has four quality dimensions – sweetness, sourness, bitterness and salinity). The Colour Cylinder NetLogo model illustrates what this space looks like. This model draws a colour circle comprising various hue and saturation values for a specific brightness value which can be adjusted using an Interface slider between the integer range of 0 to 255, as shown in Figure 9.3. For lower brightness values (see left image in the figure), the colours become progressively more darker in colour, with the entire circle becoming black when the brightness value is set at 0, whereas the colours taper towards white at the centre of the circle when the brightness slider is set at 255 (as in the right image).

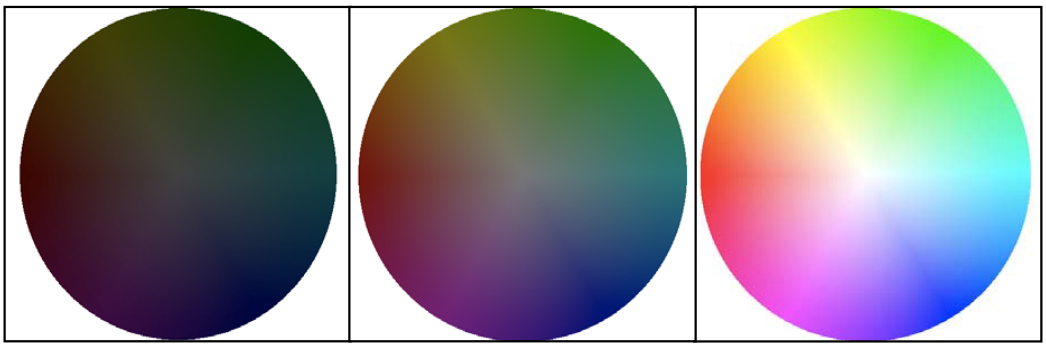


Figure 9.3 Screenshots of the colour circles produced by the Colour Cylinder NetLogo model. Settings: `hue-increment` = 0.05 and `saturation-increment` = 0.05; `brightness` = 50 (left image), `brightness` = 100 (middle image), `brightness` = 255 (right image).

Click on the ad to read more

Download free eBooks at bookboon.com

The Colour Cylinder displays the full range of possible colours across all hue, stauration and brightness values. Humans use a wide range of words in language to describe colour. Empirical evidence shows that across different cultures and ethnic backgrounds, humans agree on similar areas of colour space to refer to the basic colour terms such as red, green and blue. However, the Colour Cylinder model clearly shows that there are no clear boundaries between the colours, with one colour gradually merging into another. Therefore, it is impossible to be able to exactly demarcate the region associated with a specific colour. For example, try drawing the boundary for the colour yellow in the right image of Figure 9.3. We can clearly see where the colour yellow is, but the boundaries with the adjacent colours – red and green either side of it, and white in the centre – are fuzzy. The task of determining what is yellow gets even more difficult in three dimensions when we include the variation in brightness as well. This is why disagreements occur (as with the cars guessing game described in Section 7.8 and mentioned in Thought Experiment 9.1), since each human will have a slight variation in what they perceive the yellow region to be.

Empirical evidence show that variations in conceptual regions occur not just for colour categories but also for a variety of other basic categories such as taste. Gärdenfors also allows for abstract concepts, and provides a compelling explanation of why defining categorical knowledge is so difficult, and why a purely symbolic approach will never be completely satisfactory. The platypus, bat and archaeopteryx depicted in Figure 9.2, for example, present difficulties in categorization, the latter particularly difficult as it finds itself straddling the border between the category of reptiles and the category of birds.

Gärdenfors also explains how the meaning of combinations of concepts such as 'wooden spoon' are determined from the correlations between the domains that are common to the separate concepts 'wood' and 'spoon'. In this case the domains are size and material, and this results in us thinking of a wooden spoon as being large rather than small when visualising what it may look like. In some cases, the meaning of the concept is determined by the context in which it occurs. For example, tap water at the same lukewarm temperature can be perceived to be hot if placed in a glass, and cold if placed in a bathtub. A patch of blue sky will vary in 'blueness' depending on the time of day, how bright it is, the cloudiness of the sky, whether it has been raining for the last month and so on. Gärdenfors also illustrates the important role context plays in determing meaning by the following example. The colour red has different meanings in the following concept combinations: 'red book' (the colour we think of is close to a standard definition of the colour red); red wine (close to the colour purple?); red hair (close to the colour copper?); red skin (tawny?); red soil (ochre?); and Redwood (pinkish brown?).

Download free eBooks at bookboon.com

The conceptual spaces model overcomes some of the shortfalls of the previously disparate symbolic and sub-symbolic approaches to knowledge representation and AI, by postulating a middle level of representation. It provides a plausible explanation for aspects of human knowledge representation, such as categorization using prototypes, concept combination, and the role of context in moulding concept meaning, that present difficulties to the other models. It also is relevant for the design of embodied, situated agents as it shows how we can build knowledge without the need for symbol-grounding semantics. It also relates knowledge to locations in n-dimensional spaces so that we can characterise intelligent behaviour (for example, thinking) as movement within that space in a manner analogous to using maps to navigate and represent topographical terrain.

9.5 Knowledge engineering problems

Knowledge engineering is the process of building knowledge-based systems. Negnevitsky (2002) has identified a list of typical problems for intelligent systems shown in Table 9.3 that may require some form of knowledge engineering to find effective solutions.

Problem type	Problem description	Some example(s).
Classification	Assigning a class or classes to an object or agent that best suit its characteristics.	See Table 9.4. Many others e.g. spam filtering.
Clustering	Using observations to grouping together unclassified objects or agents into subsets (clusters) so that they have similar properties.	Cluster analysis in market research, and of medical images.
Control	Controlling a system's behaviour in real-time in order to meet specific requirements.	Control systems in space flight.
Diagnosis	Using symptoms or behaviour of an object or agent to identify an illness, or other problem such as a malfunction, and recommending possible solutions for overcoming the problem.	Mechanical repair. Medical diagnosis. e.g. Mycin.
Optimisation	The process of incrementally improving possible solutions to a problem until an optimal or near optimal one is found.	Many problems e.g. in economics; for rigid body dynamics in physics.
Prediction	Using observations of past behaviour of an object or agent to make predictions about its future behaviour.	Political forecasting from opinion polls.
Selection	Choosing one or more of the best options from a list of alternatives possibly by ranking the choices in some preferential order.	Search engines. Information Retrieval. Information Filtering.

Table 9.3 Typical types of problems for intelligent systems (from Negnevitsky, 2002).

Download free eBooks at bookboon.com

The Knowledge Representation NetLogo model has been developed to illustrate and compare the different methods of knowledge representation, and show how they can be used for various types of reasoning in the classification problem domain. The model provides three sample knowledge engineering problems as listed in Table 9.4. The Zoo Animals problem is a simplification by Kruger (1989) of a classic example from Winston (1977). The New Zealand Birds problem was introduced earlier in Section 4.4; a decision tree related to the problem is shown in Figure 4.4. The Sailing Boats problem is an example described in Negnevitsky (2002, pages 312-317).

Problem name	Problem description
Zoo Animals	Identify the name of an animal in a zoo 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.

Table 9.4 Three classification problems requiring the use of knowledge.

These three example problems 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.

9.6 Knowledge without representation

In the remaining sections of this chapter, we will explore various methods for representing knowledge – using maps, rules and logic, frames, decision trees and semantic networks. However, before we do this we need to first ask an important question: "Is it possible to have knowledge without representation?" In other words, can an agent or agents obtain knowledge without any explicit mechanism for representing the knowledge they have obtained in a knowledge base? If this is possible, this seems initially at odds to the Knowledge Representation Hypothesis. However, this hypothesis conjectures that for *intelligent* behaviour, an agent or agents must represent knowledge about the world; therefore, if the agent or agents can gain knowledge without representation, then by this definition, they do not exhibit intelligent behaviour.

Download free eBooks at bookboon.com

We have already seen how knowledge without representation is possible in a number of NetLogo models where the agents make use of stigmergy – the Ants model mentioned in Chapter 5; and the Termites models in Chapter 6. In these cases, the 'knowledge' is stored in the environment within the complex structures that emerge when the system of agents and the environment self-organises. Although any one agent does not possess knowledge individually, the entire colony of agents possesses knowledge. But can we say that these stigmergic-using agents have really gained knowledge? Under the definition that knowledge is the absence of the need for search, then clearly the ants, for example, have gained knowledge about the location of nearby food sources which they then are able to communicate back to the colony, and therefore the other agents are able to follow the path back to the food source without making a decision about which way to go. It can be argued, depending on your definition of intelligence (see next chapter), that these ants also exhibit a rudimentary form of artificial intelligence because they act and behave in an intelligent way, albeit in a limited sense.

Need help with your dissertation?

Get in-depth feedback & advice from experts in your topic area. Find out what you can do to improve the quality of your dissertation!

Get Help Now

Go to www.helpmyassignment.co.uk for more info

Helpmyassignment

Click on the ad to read more

Download free eBooks at bookboon.com

9.7 Representing knowledge using maps

Imago Mundi

Maps are one of the oldest methods used by people for representing knowledge. The earliest world maps, for example, provide an illustration of the progress of human geographical knowledge throughout the ages. The oldest known world map is the Imago Mundi, which is a 6th century BC clay tablet dated circa 600 BC residing at the British Museum that depicts the known Babylonian world showing Assyria, Babylonia and Armeni (see image on the right). The image shows that the knowledge of the world's geography at the time was clearly limited as was the quality of the mapping. Figure 9.4 depicts two further early world maps. The one on the left is a map produced by Posidonius (150–130 BC) showing a map comprising mainly the Mediterranean, southern Europe, south-western Asia and northern Africa. The one on the right is by Abraham Ortelius in 1570. They show a progression in knowledge and mapping quality, with the 'Typvs orbis terrarvm' map produced by Ortelius closest in shape to modern day maps of the world.

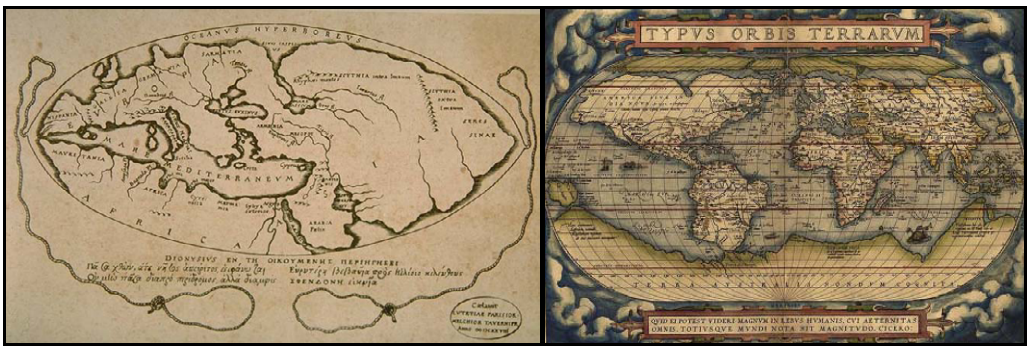


Figure 9.4 Two early world maps depicting the progress of human knowledge: A map by Posidonius (150–130 BC) (left image); and a map by Abraham Ortelius, 1570 (right image).

Download free eBooks at bookboon.com

All three of these examples illustrate an important feature of geographical maps of this kind. By their very nature, as a 2D representation of a 3D world, all maps have errors in representation – that is, they do not fully represent the reality. This is evident when the maps shown in the images are compared with modern-day world maps. For example, the Ortelius map departs most significantly from reality in its depiction of the North American continent, reflecting the state of knowledge at the time since Christopher Columbus had only just discovered America in 1492. Maps also diverge from reality if they are a static representation of the environment that the map represents – for example, the world is a dynamic and ever-changing place that has been transformed by humans over the centuries since these early maps were first drawn.

Each map becomes more useful if it is a reliable reflection of real life, but in many cases, stylised depictions occur where the relationships between adjacent objects on the map are warped to reflect a deliberate emphasis on the part of the mapper, or due to other reasons such as a lack of knowledge, a deliberate bias or an oversight. This inherent mapping error should be kept in mind in the discussion below of the traditional forms of knowledge representation. Representing knowledge can be thought of as analogous to mapping in the sense that an attempt is being made to accurately reflect an abstract n-dimensional space environment, but the mapping process will result in there being inherent mapping errors due to distortions, mistakes, deliberate emphasis, bias, oversight and lack of knowledge.

The importance of maps as a method of representing knowledge is often overlooked within the field of Artificial Intelligence in the traditional discussion on the problem of how to represent knowledge. Indisputably, however, maps contain a wealth of knowledge. The advent of Google Maps and Google Earth has illustrated the ability of online maps to represent a wide range of knowledge. The map of Central Park South in New York shown in Figure 9.5 produced using the NetLogo Map Drawing model also contains a wealth of knowledge in the sense that a person visiting Central Park for the first time can use the map to find their way around without knowing where everything is in advance. Factual or declarative knowledge is represented explicitly by the place names depicted on the map such as Tavern on the Green, Sheep Meadow, Cherry Hill and Wollman Rink. The relationships of these locations with other locations are also represented, such as which locations are adjacent to each other and the relative distances between them. Paths drawn on the map also provide explicit knowledge of how to get between different places. These paths can be thought of as providing procedural knowledge of the steps that need to be carried out (such as whether to turn left or right or go straight ahead) in order to get between two or more points. The paths depicted on the map also provide further knowledge such as their width and whether they are straight or winding i.e. they can be used to infer whether they are easier or quicker to follow.

Download free eBooks at bookboon.com

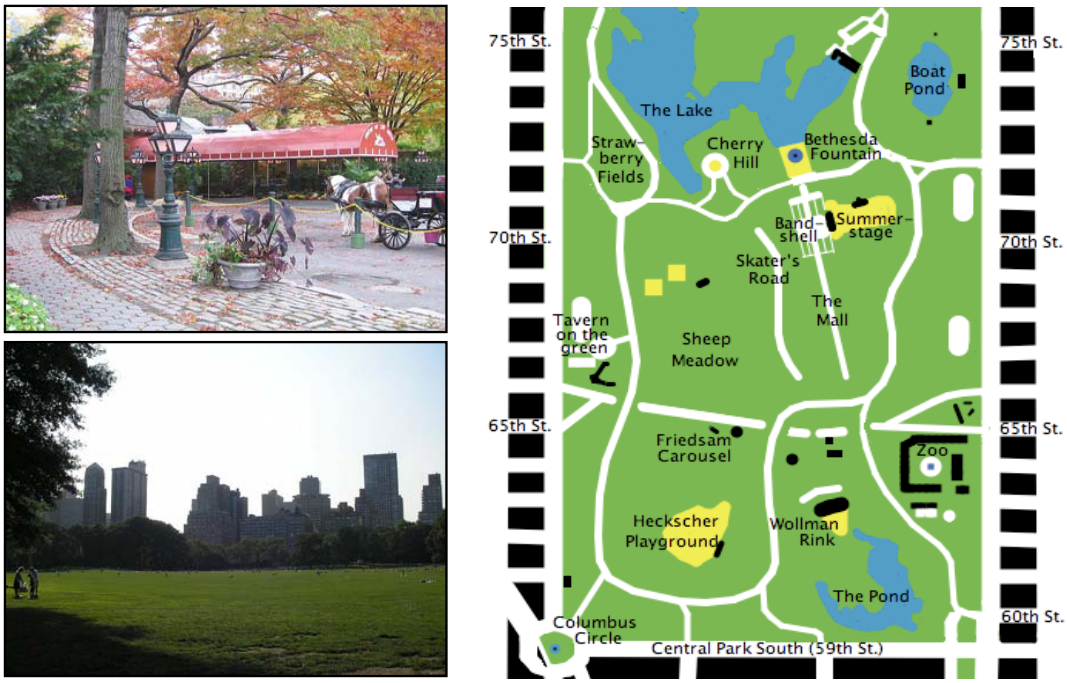


Figure 9.5 Central Park South, New York: Tavern on the Green (top left); Sheep Meadow (bottom left); a map produced by the Map Drawing NetLogo model (right).

Anyone who visits Central Park, or who uses Google Earth to zoom into the same location, will soon be aware that there is a lot of information missing and many inaccuracies from the map shown in Figure 9.5. However, just as Hampton Court Palace and Chevening House maps shown in previous chapters can be used to navigate around the real-life garden mazes, the Central Park South map can also be useful to help a person find their way around and not get lost despite these inadequacies. The map also depicts primarily topographical information with other information missing – for example, the map does not include historical facts such as that sheep were grazing in the Sheep Meadow up until 1934 (hence its name), and the Tavern on the Green was converted from a sheep pen that housed the flock.

Flake (1998, pages 416–423) in his book "*The Computational Beauty of Nature*" notes that environments, models for the environment, and search are inextricably bound up with each other. Noting that the term "model" can mean different things in different contexts, he defines it as a well-defined process that *maps* inputs to outputs that can be tuned by parameters. Search is then the process of choosing the best parameters so that the model can be made to more closely approximate an environment. He uses the example of the mathematical formula $f(x;a,b,c) = a\sin(bx - c)$ that could be used to model a periodic process. This sine wave function may or may not be a good model for what is happening in the environment where the periodic process is taking place. Obviously, such a model would be completely inadequate as a representation for most environments that are usually much more complex.

Download free eBooks at bookboon.com

However, this simple sine wave model can be tuned by searching to find the three parameters *a*, *b* and *c* that reflect the amplitude, frequency and phase of the sine wave that best suits the periodic process environment. Using the analogy of a path in real life, the periodic process is generating a particular path through the 2D graph environment. The path itself might be a rough approximation to the path generated by real life observations that relate to a frictionless pendulum, the average seasonal temperature, or the lunar cycle for example.

Models as defined by Flake can be considered analogous to topographical maps. Thinking of a model as being a map of an environment more closely links the English language being used that is based on the underlying topographical metaphor behind the concept of an environment. We can consider the search that Flake refers to as related to the process of trying to draw a map that closely represents what is being observed in the environment. The formula represents a class of maps of the path, and the parameters *a*, *b* and *c* when enumerated will specify a particular instance of the class – that is, one particular map, which may fit the actual path well, or it may not. If the map fits well, then we can use it as a useful aid to help us model the environment and make predictions about it.

Click on the ad to read more

Download free eBooks at bookboon.com

However, use of the term environment here conjures up the mental idea that we are able to search it in some way and this can cause confusion. If we use the topographical map analogy, there is perhaps less confusion over the two types of search that are possible – one being the search for a good model of the environment or what is occurring in the environment, and the other being the search of the environment itself. We will label these two types of search as follows: mapping search, and exploration search (or simply just 'mapping' and 'exploration') to emphasize the analogies being made with their real-life equivalents. During exploration, the agent has the freedom to refer to a map during execution of the search. The map may or may not be a good reflection of what exists in the environment at the time the navigation is being performed. Therefore, the agent has to continually correct or update its internal map in order to make progress to its goal.

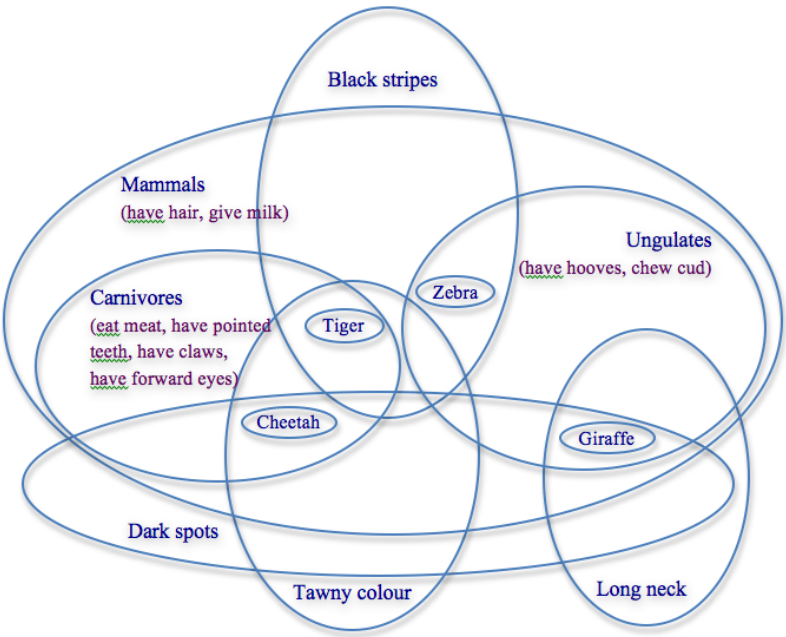


Figure 9.6 Venn diagram of the knowledge for the Zoo Animals classification problem.

Download free eBooks at bookboon.com

Conceptually, maps can be used to represent many different types of knowledge, not just topographical knowledge, since maps can be used to visualise knowledge represented by multidimensional data within an *n*-dimensional space environment. One such example is Venn diagrams devised by John Venn around 1880. Venn diagrams provide a means of representing logical relationships between sets, where regions in the diagram represent the sets or classes, and overlapping regions the intersections between the sets. These regions can be identified by labels as shown in Figure 9.6. The Venn diagram in this figure represents knowledge concerning four zoo animals, cheetahs, tigers, zebras and giraffes, whether they are mammals, carnivores, or ungulates, and features that distinguish them such as whether the animals have dark spots, a tawny colour or a long neck. Although what is depicted in this diagram are the locations of concepts such as mammals and their logical relationships instead of traditional geographical features such as countries, roads, and places as with the Central Park South and world maps, it can nevertheless be considered a map in the topological sense. There is also an obvious correlation between this method of representing knowledge and between the conceptual spaces approach of Gärdenfors shown in Figure 9.2.

9.8 Representing knowledge using event maps

A theme throughout this volume has been the use of maps to represent what is happening in different environments, whether real, virtual or abstract (i.e. an *n*-dimensional space environment). In the discussion of the different methods for representing knowledge (such as rules, logic, frames, and decision trees) in the sections that follow, each method can be thought of as being analogous to the process of representing topographical terrain using maps. The NetLogo model that has been developed to illustrate and compare the different models, the Knowledge Representation model, uses the NetLogo display environment to visualise in two dimensions the knowledge for each of the three toy problems listed in Table 9.4. As discussed above, these visualisations can be thought of as being maps of the abstract environment that represents the knowledge being described.

In order to help visualise the different types of knowledge representation, the Knowledge Representation model makes use of the NetLogo code that was also used for both the Wall Following Events (Section 6.6) and the Language Modelling Events (Section 7.10) models. It recasts the different forms of knowledge representation within a common framework based on 'events', where an event could be a sensory event (input from one of the senses), a motor event (movement of the agent's body) or an abstract event (a combination of sensory, motor or abstract events). The approach is similar to the approach adopted for Event Stream Processing (ESP) and Complex Event Processing (CEP) (Luckham, 1988). The framework used here considers that an agent simultaneously recognizes and processes multiple streams of events that reflect what is happening to itself and in the environment. Each event is represented by a state in a finite state machine that is labelled by a stream name followed by an event name.

Download free eBooks at bookboon.com

The transitions between states are not labelled – they just indicate that one event follows another. Events occur in a specific order (path) as defined by the state transitions through the event network. If a particular event does not occur after another event, then there is no particular transition between states, and therefore that event is not recognized by the agent (i.e. it is ignored and has no effect on the agent's behaviour). The processing of the events is done in a reactive manner – that is, a particular path is traversed by successively matching the events that are currently happening to the agent against the outgoing transitions from each node. If there are no outgoing transitions or none match, then the path is a dead end, at which point the traversal will stop. This is done simultaneously for every event; in other words, there are multiple starting points and therefore simultaneous activations throughout the network.

We can consider the network is an 'event map' – the states represent points in an *n*-dimensional space, with the multi-dimensionality of the points in the space reduced to two dimensions. This is done by having each state represent a single dimension and its value, and then providing links to further states along an ordered path through states representing each of the other dimensions. The reduction from *n*-dimensional space to the two dimensional space for visualisation adds further information since it imposes a specific ordering of dimensions defined by the paths to reach the final state that represents the multi-dimensional point. A three dimensional point, represented by the tuple (`D1 = value1, D2 = value2, D3 = value3`), for example, can be represented on an event map as three linked states labelled [`D1 = value1`], [`D2 = value2`] and [`D3 = value3`], with the third state representing the three dimensional point but only reachable through the two previous states.

Click on the ad to read more

Download free eBooks at bookboon.com

This specific ordering of dimensions did not exist with the original *n*-dimensional data. However, the way states are linked provides us with an opportunity to represent the viability of a particular path – in a similar manner analogous to the viable set of paths linking together two locations such as the Bethesda Fountain and Columbus Circle in the Central Park South map of Figure 9.5. Also, there is no reason why we cannot represent alternative viable paths on the event map with the other paths not depicted simply being ignored.

When using a real map for navigation, such as the Central Park South map, a person will often create a 'mental map' in their mind that is a simplification from the actual map of how to perform the navigation. For example, a glance at Figure 9.5 will show that the most straightforward way between the Zoo and the Boat Pond is to go out in the direction of the nearest skyscrapers to 65th Street, turn left then keep going until 74th Street and then turn left back into the park. This mental map focuses on the task in hand – how best to reach the Boat Pond – and has substantially less detail and less accuracy than the actual map, but is still good enough to aid the achievement of the goal.

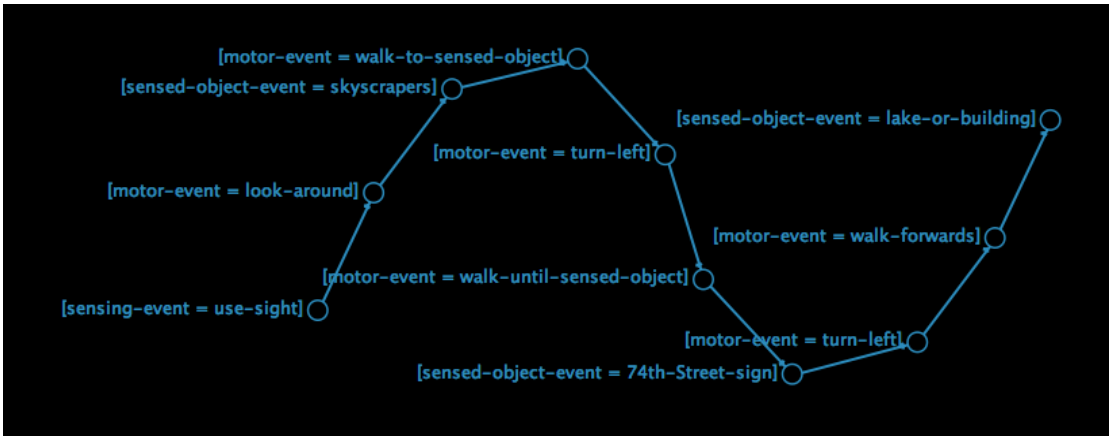


Figure 9.7 Event map of the steps needed to travel from the Zoo to the Boat Pond in Central Park. (Screenshot from the Central Park Events NetLogo model).

An event map representation of this mental map is shown in Figure 9.7. The event map depicts an ordered sequence of events that need to occur before the goal of reaching the Boat Pond can be achieved. This sequence will be successful regardless of the starting point within the Zoo as long as the agent heads towards the nearest skyscrapers, and does not require the agent to work out her current orientation with respect to north.

An event map of the knowledge for the Zoo Animals classification problem is shown in Figure 9.8.

Download free eBooks at bookboon.com

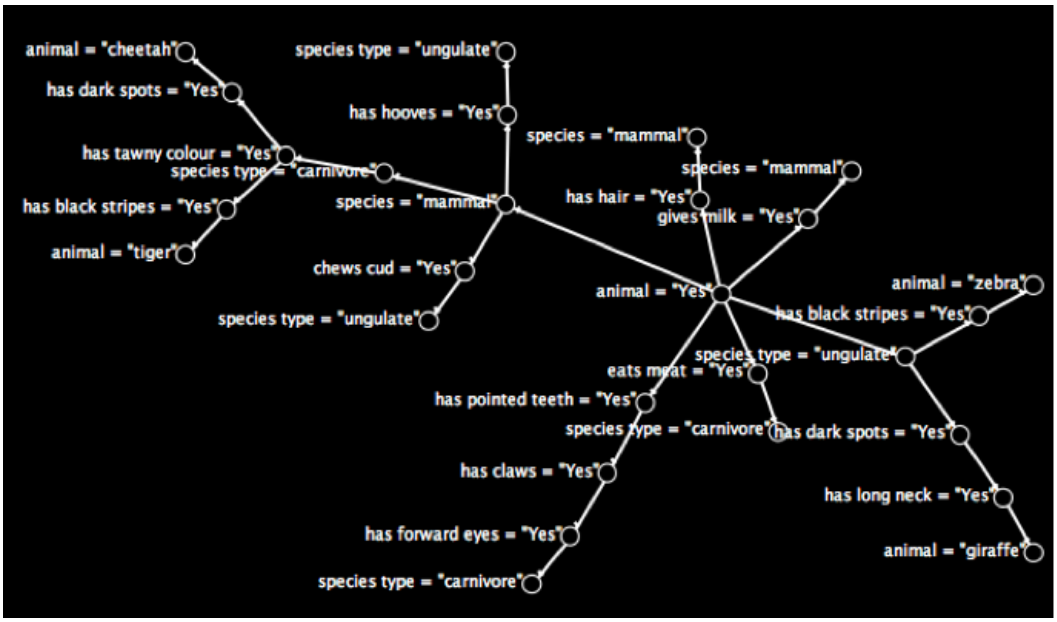


Figure 9.8 Event map of the knowledge for the Zoo Animals classification problem represented as a decision tree.

The knowledge in this case is laid out as a decision tree. The figure shows a network of events fanning out from the central right state labelled `animal = "Yes"`. The various leaf nodes are states representing when a specific type of animal has been recognized such as a carnivore (middle bottom) or ungulate (e.g. middle top), or a specific animal such as a cheetah (top left) or giraffe (bottom right). These states represent the multidimensional points in the *n*-dimensional space that represent the concepts 'carnivore', 'ungulate', 'cheetah' and 'giraffe' as with Gärdenfors conceptual spaces. It is now clearer why this representation is called an event 'map' – just as a topographical map represents 3D terrain in 2D, the event map represents points in *n*-dimensional space in 2D as well. The Knowledge Representation NetLogo model produced this particular event map by setting the `KR-type` Interface chooser to `"decision tree"`, and the `select-problem` Interface chooser to `"Zoo animals"`.

We can animate event maps by showing how the agents move around the map during reasoning, similar to how the searching algorithms in Chapters 7 and 8 were animated. What follows is a description of the more traditional forms of knowledge representation, but from an embodied, situated perspective by using the event mapping mechanism to visualise the knowledge, and animate the knowledge reasoning process.

Download free eBooks at bookboon.com

9.9 Representing knowledge using rules and logic

The classic methods of representing knowledge use either rules or logic. Table 9.5 displays the knowledge for the zoo animals problem in two formats – using rules on the left as implemented within the Knowledge Representation NetLogo model, and using first order logic on the right. Rules are often used in rule-based expert systems, and are either specified explicitly by a knowledge engineer (usually through a process called 'knowledge acquisition' from a human expert), or they are derived from data using a machine learning or data mining algorithm. Rules use a logic-based form for reasoning. Logic is the use of symbolic and mathematical techniques for deductive reasoning, and dates back as a discipline to Aristotle.

Click on the ad to read more

Download free eBooks at bookboon.com

	Rule	First-order logic
R1	IF animal has hair THEN species is mammal	$\forall x \text{Has_hair}(x) \Rightarrow \text{Mammal}(x)$
R2	IF animal gives milk THEN species is mammal	$\forall x\ \text{Gives_milk}(x) \Rightarrow \text{Mammal}(x)$
R3	IF animal eats meat THEN species type is carnivore	$\forall x\ \text{Eats_meat}(x) \Rightarrow \text{Carnivore}(x)$
R4	IF animal has pointed teeth AND animal has claws AND animal has forward eyes THEN species type is carnivore	$\forall x \text{Has_pointed_teeth}(x) \wedge \text{Has_claws}(x) \wedge \text{Has_forward_eyes}(x) \Rightarrow \text{Carnivore}(x)$
R5	IF animal is mammal AND animal has hooves THEN mammal group is ungulate	$\forall x\ \text{Mammal}(x) \wedge \text{Has_hooves}(x) \Rightarrow \text{Ungulate}(x)$
R6	IF species is mammal AND animal chews cud THEN mammal group is ungulate	$\forall x\ \text{Mammal}(x) \wedge \text{Chews_cud}(x) \Rightarrow \text{Ungulate}(x)$
R7	IF species is mammal AND species type is carnivore AND animal has tawny colour AND animal has dark spots THEN animal is cheetah	$\forall x\ \text{Mammal}(x) \wedge \text{Carnivore}(x) \wedge \text{Has_tawny_colour}(x) \wedge \text{Has_dark_spots}(x) \Rightarrow \text{Cheetah}(x)$
R8	IF species is mammal AND species type is carnivore AND animal has tawny colour AND animal has black stripes THEN animal is tiger	$\forall x\ \text{Mammal}(x) \wedge \text{Carnivore}(x) \wedge \text{Has_tawny_colour}(x) \wedge \text{Has_black_stripes}(x) \Rightarrow \text{Tiger}(x)$
R9	IF species is ungulate AND animal has dark spots AND animal has long neck THEN animal is giraffe	$\forall x \text{Ungulate}(x) \wedge \text{Has_dark_spots}(x) \wedge \text{Has_long_neck}(x) \Rightarrow \text{Giraffe}(x)$
R10	IF species is ungulate AND animal has black stripes THEN animal is zebra	$\forall x \text{Ungulate}(x) \wedge \text{Has_black_stripes}(x) \Rightarrow \text{Zebra}(x)$

Table 9.5: The rules in the Zoo Animals rules base and restated in first order logic.

The rule-based method of knowledge representation uses IF-THEN rules (sometimes called condition-action rules) to specify the knowledge. All the rules for a particular problem form the rules-base, and the knowledge-base comprises three components: the list of rules in the rules-base; the list of known facts in the facts-base; and an inferencing system, which processes the rules to derive new facts via some form of reasoning. A rule consists of an IF part which is a set of conditions (called the *antecedents*) that must be met before the rule is said to 'fire' so that the set of actions in the THEN part (called the *consequents*) are executed. For example, for Rule R1 in Table 9.5, if the condition 'animal has hair' is met – that is, there is a known fact in the knowledge base that the animal being classified has hair, then the rule is fired, and the action is to add a further fact 'species is mammal' to the knowledge-base.

Download free eBooks at bookboon.com

There may be multiple conditions in the IF part of the rule. For example, Rule R4 has three conditions that must be met before it can be fired. These conditions are separated by the AND keyword, and therefore these conditions are called 'conjunctions'. If they were separated by the OR keyword, they would be called 'disjunctions'. For the Knowledge Representation NetLogo model, only rules with conjunctions have been implemented. The set of rules and how they are defined for the zoo animals and New Zealand birds problem is shown in NetLogo Code 9.1. The third set of rules for the Sailing boats problem can be found by loading the model in NetLogo using the URL link below.

```
if (select-problem = "Zoo animals")
   [
     setup-rule ;; rule: IF animal has hair
                ;; THEN species is mammal
     [["has hair" "Yes"]]
     [["species" "mammal"]]
     setup-rule ;; rule: IF animal gives milk
                ;; THEN species is mammal
     [["gives milk" "Yes"]]
     [["species" "mammal"]]
     setup-rule ;; rule: IF animal eats meat
               ;; THEN species type is carnivore
     [["eats meat" "Yes"]]
     [["species type" "carnivore"]]
     setup-rule ;; rule: IF animal has pointed teeth AND animal has claws
               ;; AND animal has forward eyes
               ;; THEN species type is carnivore
     [["has pointed teeth" "Yes"] ["has claws" "Yes"]
     ["has forward eyes" "Yes"]]
     [["species type" "carnivore"]]
     setup-rule ;; rule: IF animal is mammal AND animal has hooves
                ;; THEN mammal group is ungulate
     [["species" "mammal"] ["has hooves" "Yes"]]
     [["species type" "ungulate"]]
     setup-rule ;; rule: IF species is mammal AND animal chews cud
               ;; THEN mammal group is ungulate
     [["species" "mammal"] ["chews cud" "Yes"]]
     [["species type" "ungulate"]]
     setup-rule ;; rule: IF species is mammal
               ;; AND species type is carnivore
               ;; AND animal has tawny colour
               ;; AND animal has dark spots
               ;; THEN animal is cheetah
     [["species" "mammal"] ["species type" "carnivore"]
     ["has tawny colour" "Yes"] ["has dark spots" "Yes"]]
     [["animal" "cheetah"]]
     setup-rule ;; rule: IF species is mammal
               ;; AND species type is carnivore
                ;; AND animal has tawny colour
                ;; AND animal has black stripes
                ;; THEN animal is tiger
     [["species" "mammal"] ["species type" "carnivore"]
     ["has tawny colour" "Yes"] ["has black stripes" "Yes"]]
     [["animal" "tiger"]]
```


Download free eBooks at bookboon.com

```
      setup-rule ;; rule: IF species is ungulate AND animal has dark spots
                  ;; AND animal has long neck
                  ;; THEN animal is giraffe
      [["species type" "ungulate"] ["has dark spots" "Yes"]
      ["has long neck" "Yes"]]
      [["animal" "giraffe"]]
      setup-rule ;; rule: IF species is ungulate
                 ;; AND animal has black stripes
                 ;; THEN animal is zebra
      [["species type" "ungulate"] ["has black stripes" "Yes"]]
      [["animal" "zebra"]]
      ]
if (select-problem = "New Zealand birds")
   [
      setup-rule ;; rule: IF bird can fly AND bird is a parrot
                 ;; AND bird is alpine
                 ;; THEN bird is a kea
      [["can fly" "Yes"] ["parrot" "Yes"] ["alpine" "Yes"]]
      [["bird" "Kea" ]]
   setup-rule ;; rule: IF bird can fly AND bird is a parrot
              ;; AND bird is not alpine
              ;; THEN bird is a kaka
      [["can fly" "Yes"] ["parrot" "Yes"] ["alpine" "No"]]
      [["bird" "Kaka"]]
   setup-rule ;; rule: IF bird can fly AND bird is not a parrot
               ;; AND bird has white throat
               ;; THEN bird is a tui
      [["can fly" "Yes"] ["parrot" "No"] ["has white throat" "Yes"]]
      [["bird" "Tui"]]
   setup-rule ;; rule: IF bird can fly AND bird is not a parrot
               ;; AND bird does not have a white throat
               ;; THEN bird is a pukeko
      [["can fly" "Yes"] ["parrot" "No"] ["has white throat" "No"]]
      [["bird" "Pukeko"]]
   setup-rule ;; rule: IF bird cannot fly AND bird is an extinct bird
               ;; AND bird is large
               ;; THEN bird is a moa
      [["can fly" "No"] ["extinct" "Yes"] ["large" "Yes"]]
      [["bird" "Moa"]]
   setup-rule ;; rule: IF bird cannot fly AND bird is extinct
              ;; AND bird is not large
              ;; THEN bird is a huia
      [["can fly" "No"] ["extinct" "Yes"] ["large" "No"]]
      [["bird" "Huia"]]
   setup-rule ;; rule: IF bird cannot fly AND bird is not extinct
              ;; AND bird has long beak
              ;; THEN bird is a kiwi
      [["can fly" "No"] ["extinct" "No"] ["has long beak" "Yes"]]
      [["bird" "Kiwi"]]
   setup-rule ;; rule: IF bird cannot fly AND bird is not extinct
              ;; AND bird does not have a long beak
              ;; THEN bird is a weta
      [["can fly" "No"] ["extinct" "No"] ["has long beak" "No"]]
      [["bird" "Weta"]]
]
```

NetLogo Code 9.1 How the rules are defined for the zoo animals and the New Zealand birds problem in the Knowledge Representation model.

Download free eBooks at bookboon.com

The rules are defined using the setup-rule procedure that is listed in NetLogo Code 9.2 below.

```
breed [rules rule]
rules-own
[
   antecedents ; if part of rule - list of necessary propositions for
               ; rule to be true
   consequents ; then part of rule - list of propositions that are
               ; the conclusions of the rule
]
to setup-rule [alist clist]
;; Creates a new rule with the specified antecedents (alist)
;; and consequents (clist)
   create-rules 1
   [ hide-turtle ; make invisible
      set antecedents alist
      set consequents clist ]
end
```

NetLogo Code 9.2 The definition of the rules breed, and the setup-rule procedure that is used to define the rules in the Knowledge Representation model.

Click on the ad to read more

Download free eBooks at bookboon.com

Rules are defined as a separate breed of agents that contain two variables: antecedents, which is a list of the necessary propositions or facts for the rule to be true (this is the IF part of the rule); and `consequents`, which is the list of conclusions (this is the THEN part of the rule). The procedure `setup-rule` takes two lists of propositions as arguments – the `alist` as the list of antecedents and the `clist` as the list of consequents. The propositions are represented using a two-element list which contains an attribute and its value. This two-element list can be converted to an English description by inserting an "is" between the two elements in the list. For example the list `["can fly" "Yes"]` represents the proposition "can fly is Yes"; that is, the proposition that the bird can fly. Similarly, the list `["species type" "mammal"]` is the proposition that the species type for the animal is a mammal.

Specifying the rules in this format allows us to visualise them using event maps. The Knowledge Representation does this by making use of the `add-events` procedure defined in Chapter 6 (NetLogo Code 6.4) as shown in NetLogo Code 9.3 below.

```
to add-rule-to-event-map [this-rule]
; adds the rule as a set of linked states to the event map network
   let events []

   foreach [consequents] of this-rule
   [
      set events lput ? ([antecedents] of this-rule)
      add-events events white white white
   ]
end

to setup-rules-event-map [problem]
;; setups an event map network of states from the rules

   foreach sort rules
   [
      add-rule-to-event-map ?
   ]

   repeat 5 [ change-layout ]
   display
end
```

NetLogo Code 9.3 How the rules are used to create an event map.

Download free eBooks at bookboon.com

The `add-rule-to-event-map` procedure adds as a list of events to the event map the list of antecedents in a specific rule for each of its consequents. It adds the particular consequent as an event at the end of the list of antecedents so that the consequent will end up at the end of the path in the event map. The `setup-rules-event-map` does this for all rules. This process will mean that rules with common antecedents will end up on the same branch. The result for the New Zealand birds problem is shown in Figure 9.9. This method of visualizing the rules allows us to see the relationships between the rules – it shows that the rules separate into two networks as in the figure, with the birds that can fly on the right (Tui, Pukeko, Kaka and Kea) and the birds that can't on the left (Moa, Huia, Kiwi and Weta).

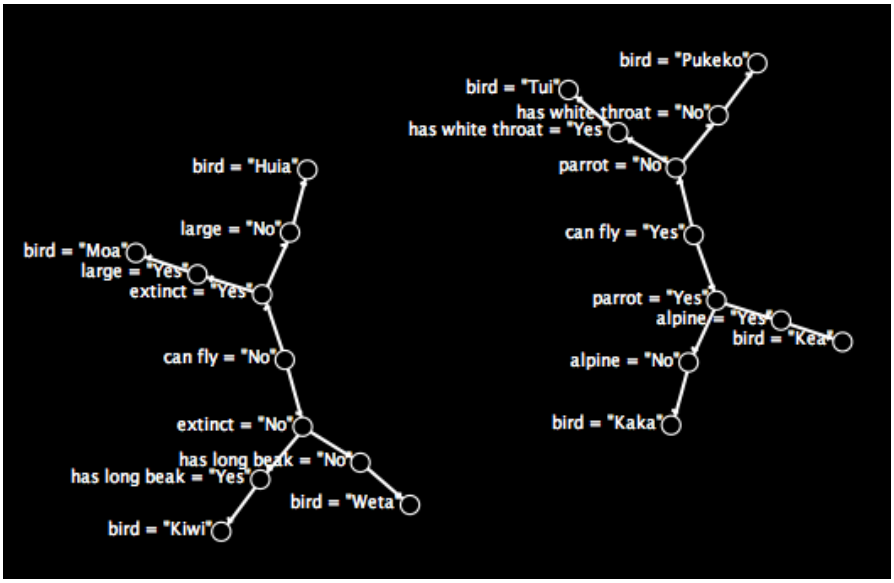


Figure 9.9 Event map of the knowledge for the New Zealand birds classification problem represented as rules.

These rules can easily be converted into first order logic form. For example, the rules shown on the left in Table 9.5 are also specified as 'sentences' in first order logic on the right. Sentences in first order logic represent statements about some world comprising of objects that have specific properties. Sentences can be comprised of a number of constituents: constant symbols that refer to a specific object in the world; terms which are logical expressions that refer to a specific object; predicate symbols that refer to a particular relation (such as 'Has_hair' and 'Mammal'); logical connectives $\Rightarrow$, $\wedge$, $\vee$ and $\Leftrightarrow$ (that are pronounced as 'implies', 'and', 'or' and 'is equivalent to' respectively); the universal quantifier $\forall$ (pronounced as 'for all'); and the existential quantifier $\exists$ (pronounced as 'there exists'). For example, the statement 'All mammals have hair' can be represented in first order logic with the sentence $\forall x\ \text{Has_hair}(x) \Rightarrow \text{Mammal}(x)$. This sentence can be read as 'for all x, if x has hair, then this implies that x is a mammal'. The universal quantifier is used for stating that a general condition holds for all objects in the world, and the existential quantifier is used for stating a specific condition holds for at least one object in the world. The symbol $\Rightarrow$ is used for stating an implication (rule), with the premise or antecedents on the left side of the symbol, and the conclusion or antecedents on the right side of the symbol. The equivalence symbol ($\Leftrightarrow$) specifies that the left hand side of the symbol is equivalent to the right hand side of the symbol. Negation is specified using — symbol (pronounced 'not').

Download free eBooks at bookboon.com

9.10 Reasoning using rules and logic

We can use the knowledge specified by the rules and logic sentences to infer new knowledge by a process called 'reasoning'. There are various 'rules of inference' that allows us to do this. For example, if we know that an animal $A1$ has pointed teeth, has forward facing eyes and has claws, i.e. Has_pointed_teeth($A1$) and Has_forward_eyes($A1$) and Has_claws($A1$) are all true, then we can infer from Rule R4 that animal $A1$ is a carnivore, i.e. Carnivore($A1$) This particular rule of inference is called 'Modus Ponens', or Implication Elimination, as it creates a new sentence by removing the implication symbol, $\Rightarrow$.

Some rules of inference are listed in Table 9.6. These rules are self-evident and follow directly from the meanings of the different symbols. For example, the Modus Ponens rule follows directly from the meaning of the $\Rightarrow$ symbol. The And Elimination, And Introduction, Or Introduction and Double Negation Elimination rules follow directly from the meanings of the $\wedge$, $\vee$ and — symbols. Similarly, the other three rules result from the meaning of the $\forall$ and $\exists$ symbols. Further rules and more details can be found in Russell and Norvig (2002).

Click on the ad to read more

Download free eBooks at bookboon.com

Rule of Inference	Description
Modus Ponens or Implication Elimination	If we know P, and $P \Rightarrow Q$, then we can infer Q.
And Elimination	If we know $P \wedge Q$, then we can infer both P, and Q.
And Introduction	If we know P, and we know Q, then we can infer $P \wedge Q$.
Or Introduction	If we know P, then for any Q, we can infer $P \vee Q$.
Double Negation Elimination	We can substitute P for $\neg\neg P$ and vice versa.
Universal Elimination	e.g. If we know $\forall x\, R(x) \Rightarrow S(x)$, we can infer $S(T)$ if $R(T)$ is true.
Existential Elimination	e.g. If we know $\exists x\, R(x) \wedge S(x)$, we can infer both $R(T1)$ and $S(T)$ as long as $T1$ is not already in the knowledge base.
Existential Introduction	e.g. If we know $R(S, T)$, we can infer $\exists x\, R(x, T)$.

Table 9.6: Some rules of inference for first order logic (from Russell and Norvig, 2002).

An example proof showing how some of these rules of inference can be applied to the animal classification problem is shown in Table 9.7.

Observations: *The animal has hooves; it has hair; it has a tawny colour; and it has black stripes.* **Goal:** *Find out what kind of animal it is.*		
	Reasoning	**Explanation**
P1	$\exists$x Has_hooves($A1$) $\wedge$ Has_Hair($A1$) Has_tawny_colour($A1$) $\wedge$ Has_black_stripes($A1$)	From the observations.
P2	Has_hooves($A1$)	From P1 and Existential Elimination.
P3	Has_hair($A1$)	From P1 and Existential Elimination.
P4	Has_tawny_colour($A1$)	From P1 and Existential Elimination.
P5	Has_black_stripes($A1$)	From P1 and Existential Elimination.
P6	Has_hair($A1$) $\Rightarrow$ Mammal($A1$)	From rule R1 and Universal Elimination.
P7	Mammal($A1$)	From P6, P3 and Modus Ponens.
P8	Mammal($A1$) $\wedge$ Has_hooves($A1$) $\Rightarrow$ Ungulate($A1$)	From R5 and Universal Elimination.
P9	Ungulate($A1$)	From P8, P7, P2 and Modus Ponens.
P10	Ungulate($A1$) $\wedge$ Has_black_stripes($A1$) $\Rightarrow$ Zebra($A1$)	From R10 and Universal Elimination.
P11	Zebra($A1$)	From P10, P9, P5 and Modus Ponens.

Table 9.7: A sample proof in first order logic.

At the beginning, some observations have been noted concerning an animal being observed on the African savannah. The goal is to determine which animal is being observed. The proof starts with converting the observations into first order logic form using the existential quantifier (P1). P2 to P5 uses existential elimination to infer individual predicates concerning an animal labelled $A1$. The remaining part of the proof uses Universal Elimination and Modus Ponens to arrive at the conclusion that the animal is a zebra – that is, Zebra($A1$) is proven to be true.

Download free eBooks at bookboon.com

Rule-based expert systems mainly make use of two forms of reasoning based on the automated application of the Modus Ponens rule of inference. 'Forward reasoning' (also called 'forward chaining') works in a forward direction by first starting with the known facts and then trying to derive a specific goal using a chain of inference by repeated application of Modus Ponens (as in Table 9.7). 'Backward reasoning' (also called 'backward chaining') works in the opposite direction, starting with the goal first, then working backwards until it has been proved.

In the Knowledge Representation model, clicking on the `go-reasoning` button starts the reasoning process. If the `KR-type` slider is set to `"rules"`, the user will be asked to select which type of reasoning they wish to use – whether forward or backward. Both of these types of reasoning use search to explore an *n*-dimensional space that represents the knowledge. This *n*-dimensional space is shown by the event map depicted in the NetLogo environment when the `setup-knowledge` button has been clicked in the Interface (as shown in Figures 9.10 and 9.11). As the reasoning process proceeds, the event map is animated with walker agents depicted using the person shape (in the same manner that the walker agents were depicted exploring the search spaces related to the different search problems described in the previous chapter). This animation of the reasoning process clearly reinforces the analogy of reasoning as a search process.

Figure 9.10 provides two screenshots of how this looks for the New Zealand birds classification problem. The left hand image shows a screenshot at the completion of forward reasoning, which has led to the conclusion that the bird being classified is a Kea. The screenshot shows the path the reasoning process has taken to get to the goal using thicker links. In this case the path has been direct. The first state visited was the `[can fly = "Yes]` state in the middle right of the image. This caused the following question to be asked of the user – "Can the bird fly?". The path then goes to the next state labelled `[parrot = "Yes"]`. This is shown by the link being slightly wider than other links that have not been traversed. This caused the question "Is the bird a parrot?" to be asked. The path then goes to the state labelled `[alpine = "Yes"]` which generates the question "Is the bird an alpine bird?". Eventually, the path ends up at the final state labelled `[bird = "Kea"]` which is a goal state, and so the conclusion is reached that the bird is a Kea. This path is followed because a single rule is being matched – in this case, the rule is the first one in the knowledge-base as defined in NetLogo Code 9.1, with antecedents list `["can fly" "Yes"] ["parrot" "Yes"] ["alpine" "Yes"]` and consequents list `["bird" "Kea"]`.

Download free eBooks at bookboon.com

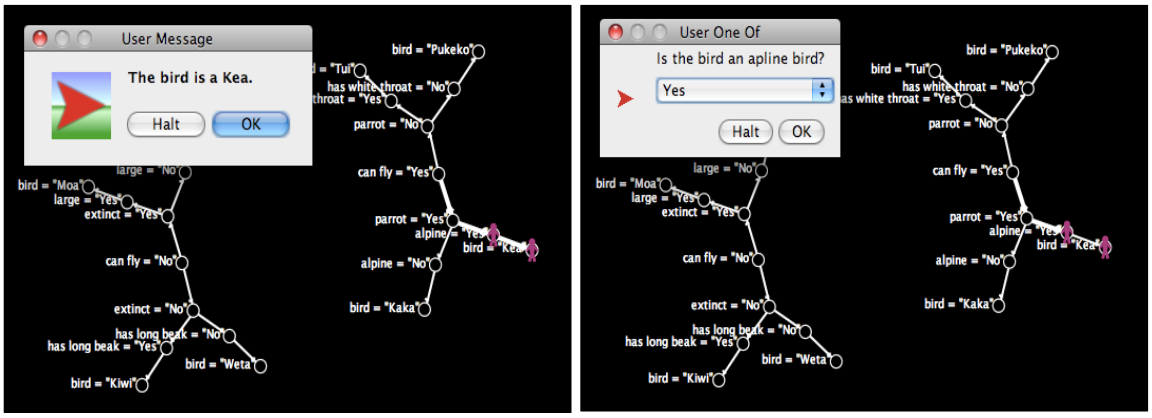


Figure 9.10 How the event map for the New Zealand birds rules-base is animated using forward reasoning (left image) and backward reasoning (right image).

The right hand image in Figure 9.10 shows how the backward reasoning proceeds. At the beginning, the user is asked which goal they wish to prove. If the user selects [`bird Kea`], then a walker agent is drawn at the state labelled [`bird = "Kea"`] to show that it is the goal state. The reasoning process then tries to prove the goal by asking the same questions as for the forward reasoning process. In the image, the path has been traversed as far as the [`alpine = "Yes"`] state, with one more link yet to be traversed needed to prove the goal.

Deloitte.

Discover the truth at www.deloitte.ca/careers

© Deloitte & Touche LLP and affiliated entities.

Click on the ad to read more

Download free eBooks at bookboon.com

This particular example is relatively straightforward. For a more complicated example, such as classifying a zoo animal as a cheetah, the reasoning process will result in the animation jumping around the event map. The forward reasoning process, in particular, can result in many queries being asked before a conclusion is reached, as shown in Figure 9.11.

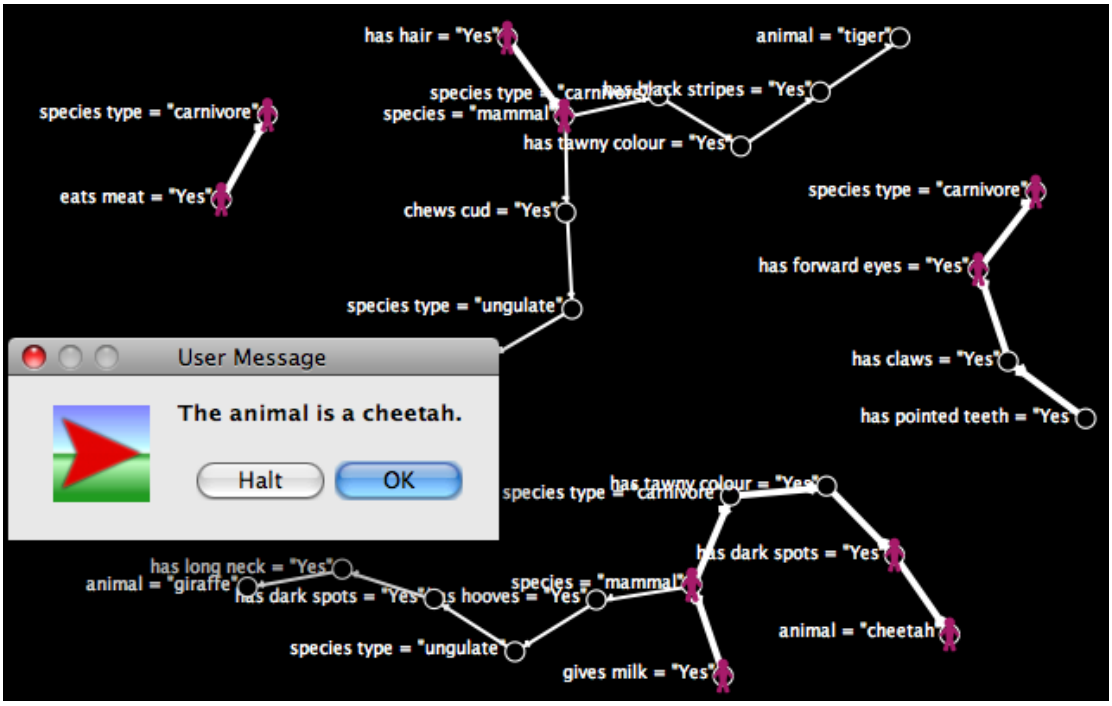


Figure 9.11 How the event map for the zoo animals rules-base is animated using forward reasoning.

For forward reasoning, the process starts from the known facts and proceeds forward by trying to find a rule in the rules-base for which the antecedents match against the facts. Each time a rule matches, it is 'fired' and the consequents are added as new facts to the knowledge-base. A rule can only be matched once. A complete 'match-fire cycle' occurs when all the rules in the rules-base have been processed. If at least one rule has been fired, then a new cycle is started at the first rule in the rules-base. The process continues until no further rules are fired during a cycle. The code that performs this is shown in NetLogo Code 9.4.

Download free eBooks at bookboon.com

```
to go-rules-forward-reasoning
;; performs forward reasoning on the problem
   let fired-count length fired-rules
   print selected-problem
   let goal-attrib [goal-attribute] of selected-problem

   set cycle-count cycle-count + 1
   output-number "Cycle " cycle-count

   loop
      [
      foreach sort rules
           [ ; for each rule in the rules base
              if not member? ? fired-rules
              [
                 if (match-rule? selected-problem ?)
                    [
                       fire-rule ?
                       output-fired-rule ?
                    ]
                 if (attribute-found? goal-attrib)
                    [ user-message (word "The " goal-attrib " is a "
                          fact-get goal-attrib ".")
                    stop ]
              ]
           ]

      if (fired-count = length fired-rules)
      [ user-message
        "No new rules fired this cycle. Sorry - I can't help you."
      stop ]
   ]
end
```

NetLogo Code 9.4 The procedure that performs forward reasoning.

The `go-rules-forward-reasoning` procedure has a loop that performs the match-fire cycle. This loop processes in sorted order each rule in the rules-base (i.e. the `rules` agentset). It first checks whether the rule has been fired previously, and if it hasn't, then it tries to match the rule by calling the `match-rule?` reporter (shown in NetLogo Code 9.5). If this reporter returns true, then it will fire the rule by calling the `fire-rule` procedure. The procedure `output-fired-rule` will output information to the Output area in the Interface as a trace of the reasoning process if the switch `trace-on` has been set to `On`.

Download free eBooks at bookboon.com

```
to-report match-rule? [problem this-rule]
; tries to match this-rule to facts in the facts-base
; otherwise asks user for data if attribute is not in non-input-attributes list
   let attribute ""
   let val ""
   let value ""
   let question ""
   let choices []
   let this-walker nobody

   set this-walker start-new-walker nobody "" ""
   foreach [antecedents] of this-rule
      [
      set attribute first ?
      set value last ?
      update-walker this-walker attribute value

      if not attribute-found? attribute
           ; not found, so need to ask user if we can
           [ ifelse (member? attribute [non-input-attributes] of problem)
              [ report false ] ; no we can't ask user
              [ ; ask user a question
                set question get-input-question problem attribute value
                set choices get-input-choices problem attribute
                set val user-one-of question choices
                fact-put attribute val
                ]
      ]
      if not fact-found? attribute value
      [ report false ]
     ]
   report true
end

to fire-rule [this-rule]
; fires the matched rule this-rule
   let attribute ""
   let value ""
   let new-walker nobody

   foreach [consequents] of this-rule
   [
      set attribute first ?
      set value last ?

      ask active-walker
      [ hatch-walkers 1 [ set new-walker self ]] ; clone active walker

      update-walker new-walker attribute value

      fact-put attribute value
      set fired-rules fput this-rule fired-rules
   ]
end
```

NetLogo Code 9.5 The code that defines how the rules are matched and fired.

Download free eBooks at bookboon.com

The `match-fire?` reporter processes each of the antecedents of the rule. It first extracts the `attribute` and its `value` from the current item in the antecedent list; for example, for `["can fly" "Yes"]`, the attribute is `"can fly"` and the value is `"Yes"`. The `attribute` combined with its `value` represents a specific proposition that needs to be true – in this case, that proposition that the bird can fly. Representing the proposition as an event in an event map, the stream name is set to the `attribute`, and the event that occurs on the stream is set to the `value`. The `match-fire?` reporter then checks to see whether a value for the attribute has not already been allocated a value on the facts-base. If it hasn't, it will ask the user what the value is if the attribute is an input attribute. Then the reporter will finally check whether the attribute's value matches the one specified by the rule.

The `fire-rule` procedure processes each of the consequents of the rule. First it hatches a new walker that moves to the new event state in the event map based using the `update-walker` procedure. This is done to animate the event map visualisation to show how the reasoning process is proceeding. Then the procedure puts a new fact consisting of the attribute and its value into the facts-base, and finally, adds the rule to the list of fired rules so that it won't be fired again.

The facts-base is implemented using the table extension as shown in NetLogo Code 9.6.

SIMPLY CLEVER
ŠKODA

Do you like cars? Would you like to be a part of a successful brand?
We will appreciate and reward both your enthusiasm and talent.
Send us your CV. You will be surprised where it can take you.
Send us your CV on
www.employerforlife.com
Click on the ad to read more

Download free eBooks at bookboon.com

```
to-report attribute-found? [attribute]
; reports true if the attribute is in the facts database
; reports false otherwise

   report (table:has-key? facts-base attribute)
end

to-report fact-found? [attribute value]
; reports true if the attribute and value are in the facts database
; reports false otherwise

   report (value = table:get facts-base attribute)
end

to-report fact-get [attribute]
; gets the value associated with the attribute from the facts-base.

   report table:get facts-base attribute
end

to fact-put [attribute value]
; puts the attribute value pair into the facts-base.

   table:put facts-base attribute value
end
```

NetLogo Code 9.6 The code that defines various procedures and reporters for the facts-base.

The code implements three reporters: `attribute-found?`, which reports true if the attribute by itself is found in the facts-base (i.e. regardless of whatever value is associated with it); `fact-found?`, which reports true if the both the attribute and specific value is found in the facts-base; and `fact-get`, which returns the value associated with the attribute. The procedure `fact-put` inserts a specific attribute and its value into the table.

A problem with forward reasoning is that the search to find a particular goal can be inefficient as it can fire many rules unrelated to the goal. In this case, backward or goal-driven reasoning may be more appropriate. Backward reasoning employs a 'find-prove' process rather than a 'match-fire' process as for forward reasoning. Given a specific goal, the backward reasoning process first searches the rules-base to find rules that have the goal in their THEN parts (i.e. as consequents). If a rule is found, and the IF part matches, then the rule is fired and the goal is proved. Otherwise, a new sub-goal is set up to try to prove the IF part that does not match. The process then recursively repeats the reasoning process with the new sub-goal, which may generate further sub-goals that need to be proved, until it has proven all the IF parts for the original goal and subsequent sub-goals.

The code for backward reasoning implemented in the Knowledge Representation model is shown in NetLogo Code 9.7.

Download free eBooks at bookboon.com

```
to go-rules-backward-reasoning
;; performs backward reasoning on the problem

   let goal-choices []
   let goal-attrib ""
   let goal-value ""
   let attribute ""
   let value ""

   set goal-attrib [goal-attribute] of selected-problem
   foreach sort rules
      [ ; add each goal in all the rules to the goal-choices
      foreach [consequents] of ?
      [
          set attribute first ?
          set value last ?

          if (attribute = goal-attrib) and
           (not member? value goal-choices)
          [ set goal-choices lput ? goal-choices ]
      ]
   ]

   let goal user-one-of "Which goal do you wish to proove?" goal-choices
   set goal-value (last goal)
   start-new-walkers goal-attrib goal-value

   ifelse not find-prove? selected-problem goal-attrib goal-value
      [ user-message "No more rules found. The goal is not proven." ]
      [ update-walker active-walker goal-attrib goal-value
      user-message (word "The goal is proven! The " goal-attrib " is a "
            goal-value ".")]
end
```

NetLogo Code 9.7 The procedure that performs backward reasoning.

The code first generates `goal-choices` which is the list of all the goals from all of the rules. The user is then asked which goal they wish to prove. A walker is then drawn on the event map at the chosen goal state, and the reasoning process starts by calling the `find-prove?` reporter. This returns true if a rule can be found to prove the goal or false if not, and the procedure then informs the user of the result. The code for the `find-prove?` reporter is shown in NetLogo Code 9.8.

Download free eBooks at bookboon.com

```
to-report find-goal? [this-rule goal-attrib goal-val]
;; reports if the goal is in the consequents of this_rule
   let attribute ""
   let val ""
   let value ""
   let response ""

   foreach [consequents] of this-rule
      [
      set attribute first ?
      set value last ?

      if (attribute = goal-attrib) and (value = goal-val)
            [ report true ]
      ]
   report false
end

to-report prove-goal? [problem this-rule goal-attrib goal-val]
; tries to prove this-rule from facts in the facts-base
; otherwise asks user for data if attribute is not in non-input-attributes list
; otherwise recursively calls the procedure find-prove? to see if
; the non-input-attributes can be proved
   let attribute ""
   let val ""
   let value ""
   let question ""
    let choices []
   let this-walker nobody

   set this-walker start-new-walker nobody "" ""

   foreach [antecedents] of this-rule
      [
      set attribute first ?
      set value last ?
      update-walker this-walker attribute value

      if not attribute-found? attribute
           ; not found, so need to ask user if we can
           [ ifelse (member? attribute [non-input-attributes] of problem)
              [ ifelse not find-prove? problem attribute value
                 [ report false ] ; no we can't ask user
                 [ fact-put attribute value ]
              ]
              [ ; ask user a question
                 set question get-input-question problem attribute value set
                 choices get-input-choices problem attribute
                 set val user-one-of question choices
                 fact-put attribute val
              ]
           ]
      if not fact-found? attribute value
      [ report false ]
      ]
   update-walker this-walker goal-attrib goal-val

   report true
end
```


Download free eBooks at bookboon.com

```
to-report find-prove? [problem goal-attrib goal-val]
;; recursive procedure called by the go-backward-reasoning procedure

   let this-walker nobody

   output-goal goal-attrib goal-val
   foreach sort rules
      [ ; for each rule in the rules base
      if (find-goal? ? goal-attrib goal-val)
           [
              if (prove-goal? problem ? goal-attrib goal-val)
                 [
                    start-new-walkers goal-attrib goal-val
                    output-fired-rule ?
                    report true
                 ]
              ]
           ]
      report false
end
```

NetLogo Code 9.8 The code that defines how the rules are found and proven.

Click on the ad to read more

Download free eBooks at bookboon.com

The `find-prove?` reporter calls two further reporters, `find-goal?` and `prove-goal?`. For each rule in the rule-base, it first checks to see if the goal is found by calling the `find-goal?` reporter, then if found, tries to prove it by calling the `prove-goal?` reporter. The `find-goal?` reporter returns true if the attribute and value specified by the `goal-attrib` and `goal-val` parameters is found in one of the consequents of a specific rule `this-rule` passed by parameter. The `prove-goal?` reporter tries to first prove the specific rule `this-rule` is true from facts in the facts-base. Otherwise it checks each of the antecedents of the rule to see if they are true, asking the user for data if the attribute is not an input attribute, otherwise it recursively calls the reporter `find-prove?` to see if the non-input attribute can be proved. The `start-new-walker, start-new-walkers` and `update-walker` procedures are used to animate the event map to show how the reasoning process is proceeding.

Which type of reasoning should be used when designing a rule-based expert system? The best answer to this question is for the knowledge engineer to observe the expert to see which type of reasoning they are using. If the expert looks at the data first, and then infers new knowledge from it, then a forward reasoning process is most appropriate. On the other hand, if the expert begins with a hypothetical solution first, and then tries to find facts that will help prove the hypothesis, then backward reasoning is likely to be the most appropriate. Analysis and interpretation type problems lend themselves to forward reasoning solutions, whereas diagnosis type problems lend themselves to backward reasoning. DENDRAL, an expert system that determines the molecular structure of unknown soil is an example of the former, whereas MYCIN, an expert system for the diagnosis of infectious blood diseases is an example of the latter (Negnevitsky, 2002).

Most expert systems today now use both forward and backward reasoning. The primary method of inference used is the latter as it minimises the number of pointless queries which is a fault of the forward reasoning approach. However, an expert system can switch to forward reasoning when a new fact is added to the facts-base to make maximum use of the new data.

Expert systems such as DENDRAL and MYCIN have taken a lot of expense and effort to develop (estimated at between 20 to 40 person years to complete). A contributory factor to this is the widely acknowledged difficulty of acquiring the knowledge from human experts – this problem is called the 'knowledge acquisition bottleneck'. However, there are now sophisticated expert system shells and toolboxes for intelligent systems that use technologies such as machine learning, neural networks and evolutionary computation that dramatically reduce the development time from years down to months. For a more comprehensive discussion of rule-based expert systems and intelligent systems in general, the reader should refer to Negnevitsky (2002).

Download free eBooks at bookboon.com

9.11 Knowledge and reasoning using frames

Frames are a form of knowledge representation proposed by Marvin Minsky (1975). A frame is a data structure that stores prototypical information relating to a specific concept. Frames consists of 'slots' which can have 'values' that are either explicitly declared when the frame is declared, or they are inherited from a parent frame. This form of knowledge representation is the one most reminiscent of Gärdenfors conceptual spaces model, with slots being analogous to quality dimensions.

There are two types of frame: a class frame; and an individual or instance frame. The notions of class, instance, and inheritance are analogous to those used in object oriented programming described in Section 2.2 (also see Figure 2.1). Inheritance is usually specified using the 'ako' slot (this stands for 'a kind of'). For example, in the zoo animals classification problem, a cheetah is a kind of mammal, and a mammal is a kind of animal. As a further example, the frames for the Sailing boat classification problem are shown in Table 9.8. The table shows the eight sailing boat frames in separate columns, with slot names and values shown in the rows. For this particular problem, the knowledge is relatively regular, with only a few table cells with missing values. Normally, missing values are the norm when the frames are listed in tabular form because there is no requirement that each frame should have the same slots, because frames represent different concepts with different properties, and can inherit information from different parent frames.

Click on the ad to read more

Download free eBooks at bookboon.com

The event map representation for this knowledge is shown in Figure 9.12. For clarity purposes, the figure shows the event map for just the first four frames for the Sailing boat problem – the Jib-headed cutter (top left), the Gaff-headed sloop (bottom left), the Jib-headed ketch (top right) and the Gaff-headed ketch (bottom right). Each frame is depicted as a separate path, starting from the event state labelled by the frame's name (for example, `name = "jib-headed-cutter" on the top left`), then proceeding to a state that declares what kind of frame the path represents (this state is always labelled as `ako = "[boat]"` as all the frames are `boat` frames in this example). The path then proceeds through the states that represent each frame's specific slots and values.

Slot	**Jib-headed Cutter**	**Gaff-headed Sloop**	**Jib-headed Ketch**	**Gaff-headed Ketch**	**Jib-headed Yawl**	**Gaff-headed Yawl**	**Gaff-headed Schooner**	**Staysail Schooner**
ako	boat	boat	boat	boat	boat	boat	boat	boat
number of masts	1	1	2	2	2	2	2	2
shape of mainsail	triangular	quadri-lateral	triangular	quadri-lateral	triangular	quadri-lateral	quadri-lateral	triangular with two foresails
main mast position			forward of the short mast	forward of the short mast	forward of the short mast	forward of the short mast	aft the short mast	aft the short mast
short mast position			forward of the helm	forward of the helm	aft the helm	aft the helm		

Table 9.8: The frames for the Sailing boat classification problem shown in tabular form.

Rather than declaring the frames separately from the rules based knowledge, the Knowledge Representation model instead uses the approach of converting the rule-based knowledge into frame format. This is in order to illustrate the correspondence between the two approaches when the propositions in the rules are specified using attribute value pairs. This also allows the different methods to be visualised using event maps. The code for doing this is shown in NetLogo Code 9.9.

Download free eBooks at bookboon.com

```
to convert-rules-to-frames-base [problem]
;; Converts the rules base to a frames base

   let this-rule nobody
   let this-frame nobody
   let attribute ""
   let value ""

   ask frames [ die ] ; kill off any previous frames

   foreach sort rules
      [ ; for each rule in the rules base
      set this-rule ?
      foreach [consequents] of this-rule
           [
              set attribute (first ?)
              set value (last ?)
              set this-frame find-frame? attribute value
              if (this-frame = nobody)
                 [ ; frame does not exist with that name and type
                   create-frames 1
                      [
                        hide-turtle ; make invisible
                        set this-frame self
                        set name value
                        set ako-list (list attribute)
                        set slots-table table:make
                      ]
                   ]
                 foreach [antecedents] of this-rule
                   [
                      ask this-frame
                         [
                           set attribute (first ?)
                           set value (last ?)
                           ifelse (member? attribute [input-attributes] of
                           problem)
                              [ table:put slots-table attribute value ]
                              [ set ako-list
                                 (lput value ako-list) ] ; add value to ako-list
                   ]
              ]
      ]
      ]
end

to-report find-frame? [this-ako this-name]
;; reports the frame that has slot value "ako" = ako and "name" = name.
;; reports nobody otherwise

   report one-of frames with
      [member? this-ako ako-list and name = this-name]
end
```

NetLogo Code 9.9 How the rules-base is converted into a frames-base.

Download free eBooks at bookboon.com

The conversion procedure first kills off any existing frames, then processes each of the rules in the rule-base in turn. For the consequents of each rule, it checks that a frame with the same `name` and `ako` slot values do not already exist by calling the `find-frame?` reporter. If it does not exist, it will create a new frame, otherwise it will use the one that was found. Then it will insert slots and their values into the `slots-table` for the frame if the proposition attribute is an input attribute, otherwise it adds the proposition's value to the `ako-list` for the frame. The event map is created from these frames using the `add-events` procedure as shown in NetLogo Code 9.10.

Click on the ad to read more

Download free eBooks at bookboon.com

```
to-report slots-events [table]
;; returns the list of events for the slots table
   let events []
   let keys []
   let value ""

      set keys table:keys table
      foreach keys
      [
           set value table:get table ?
           set events lput (list ? value) events
      ]
      report events
end

to setup-frames
;; setups a network of states from the frames

   init-problem ; initialise the selected problem

   let events []

      convert-rules-to-frames-base selected-problem
      foreach sort frames
      [
      set events slots-events [slots-table] of ?
      set events fput (fput "ako" (list [ako-list] of ?)) events
      set events fput (list "name" [name] of ?) events

      add-events events white white white
      ]
      repeat 5 [ change-layout ]
      display
end
```

NetLogo Code 9.10 How the frames are used to create an event map.

This code defines the `setup-frames` procedure that calls the `convert-rules-to-frames-base` procedure defined above, then for each of the newly created frames, it generates a list of events for each of the frame's slots in its slots table (by calling the `slots-events` reporter), and then adds two further events at the beginning of the events list, one for the `ako-list` and another for the name of the frame, before calling the `add-events` procedure.

Download free eBooks at bookboon.com

There are two main types of reasoning with frames – reasoning via inheritance, and reasoning via matching. The former fills in missing slot values for frames if there is a slot value specified for a parent frame. For example, for the Sailing boat problem, a parent 'boat' frame could specify that the number of sails by default for boats is two. Recall that the purpose of a frame is to represent a prototypical concept, so in specifying that the boat frame has two sails, then this is defining that the prototypical boat will have two sails. There are of course always exceptions to the rule, and therefore a default parent frame value can easily be overridden in a child frame by inserting a specific value. Cars, for example, mostly have four wheels, but some cars have three wheels such as the Reliant Robin manufactured in England in the 1970s, and recent hybrid and electric cars that adopt the three wheel solution for aerodynamic and performance reasons. So a parent 'car' frame might set the slot value for 'number of wheels' to 4, but set the value to 3 instead for the 'Reliant Robin car' frame, for example.

The second form of reasoning for frames based on matching involves searching the frames-base for a frame whose slots and values match the current known facts. This has been implemented in the Knowledge Representation model. Figure 9.12 is a screenshot that provides an intermediate stage of the reasoning process to show how it proceeds.

Click on the ad to read more

Download free eBooks at bookboon.com

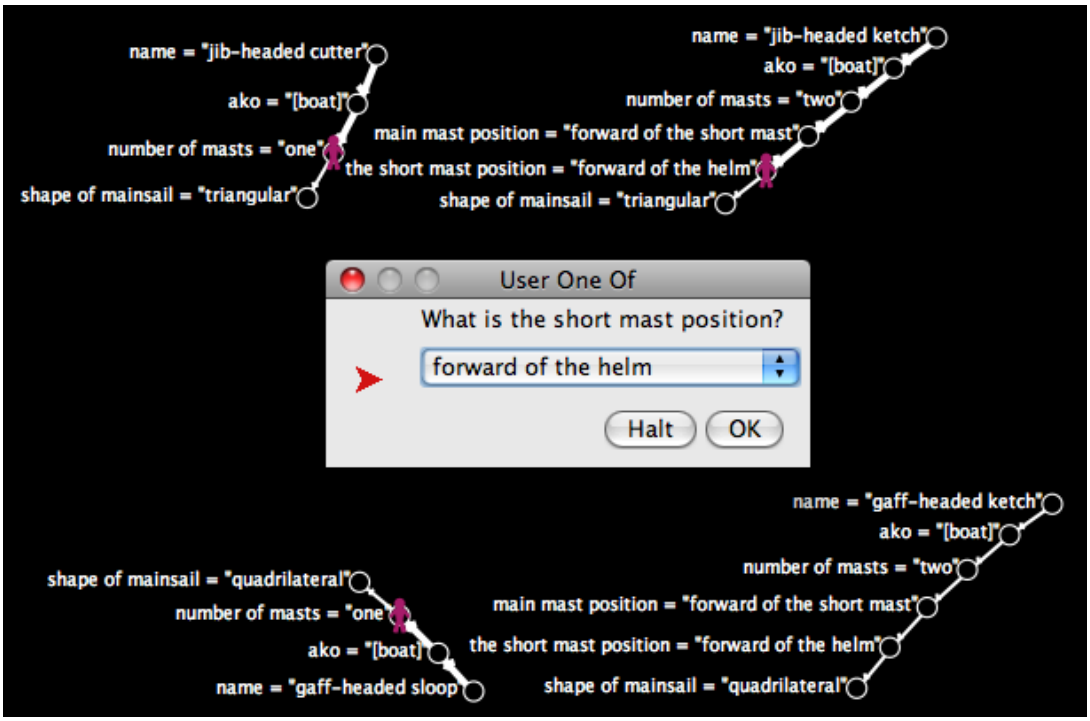


Figure 9.12 The event map of the knowledge in the first four frames for the Sailing boats problem showing the progress of the frame-matching reasoning process.

In the screenshot, the paths that have been followed are depicted by thicker white links. The first question asked, due to the Jib-headed cutter being the first frame matched, is "How many masts are there?" because the slot "number of masts" was being matched. The user replied with "two", which resulted with both the Jib-headed cutter frame and the next frame, the Gaff-headed sloop, failing to match. This is shown by the walker agents stopping just short of the end of the two paths shown on the left of the screenshot. The reasoning process then tried to match the next frame in the frames-base – the Jib-headed ketch frame. The screenshot shows the stage after the question "What is the main mast position?" had been answered with "forward of the short mast", and the user is about to answer the next question "What is the short mast position?" displayed in the centre of the screenshot.

Download free eBooks at bookboon.com

The code that performs the frame matching process is defined by the `go-frames-reasoning` procedure within the Knowledge Representation model in NetLogo. Rather than reproduce the code here, the reader can examine the code directly in NetLogo by selecting the `Procedures` button. The code consists of a reporter, `match-frame?`, that returns true if a frame matches the current facts. The `match-frame?` reporter first creates a new walker agent, and moves it along the beginning of the frame's path in the event map through the name and ako states. Then it processes each of the names in the frame's `ako-list` to make sure each of the parent frames match by recursively calling `match-frame?`. This will also mean that the grandparent frames (and great-grandparent frames and so on) will also need to be matched before the frame itself can match. Once all these parent frames have been matched, the reporter then extracts the list of slot names from the frame's `slots-table`, and then for each of the slots, extracts their values and using similar code to that used in the rule-based reasoning listed above in Section 9.10, will try to see if a fact with the slot name and value already exists in the facts-base. If not, it will ask the user for input if the slot is an input attribute, otherwise it will return false (the frame has not matched).

The `go-frames-reasoning` procedure calls the `match-frame?` reporter for each of the frames that have the goal attribute in their `ako-list` which is set when the frames are created – for example, for the Sailing boat frames-base the goal attribute is set to `boat`, and for the zoo animals frames-base the goal attribute is set to `animal`, and therefore only those frames are matched for the problem.

There are many variations to the frame-based method of knowledge representation. An extension to the basic slot-value type of representation is to allow the slots to have 'facets' to provide extended knowledge about the attribute being described by the slot. Facets can be used for controlling user queries and guiding the reasoning process, for example detailing how the slot should be matched. Methods (called 'demons') can also be associated with attributes as facets that provide the frame-based system a mechanism for expressing procedural knowledge. These methods perform actions that are triggered under certain conditions. For example, two common types of methods are WHEN CHANGED and WHEN NEEDED, which are triggered when a slot's value is changed or when it is accessed.

Many expert and intelligent systems today are hybrid systems that combine two or more technologies; for example, many systems use both frames and rules for representing knowledge and for reasoning. A term often used to describe these hybrid systems is 'soft computing' which deals with systems that are capable of reasoning with uncertainties and dynamic multiple-agent environments. Negnevitsky (2002) provides several examples of hybrid intelligent systems that combine various technologies, not just the symbolic-based technologies such as rules and frames, but also the sub-symbolic technologies such as neural networks and evolutionary computing. These latter technologies will be described in Volume 2 of this book series.

Download free eBooks at bookboon.com

9.12 Knowledge and reasoning using decision trees

Another method of representing knowledge is using decision trees. We have already seen some examples of decision trees in Section 4.4, and in Figure 9.8 above, which provides a screenshot of the decision tree for the zoo animals problem. NetLogo Code 9.11 shows how rules can be converted into a decision tree using the event map representation discussed in Section 9.8.

Click on the ad to read more

Download free eBooks at bookboon.com

```
to convert-rules-to-decision-tree
;; Converts the rules base to a decision tree

   init-problem ; initialise the selected problem

   let this-rule nobody
   let attribute ""
   let value ""
   let events []

   foreach sort rules
      [ ; for each rule in the rules base
      set this-rule ?

      set events (list (list [goal-attribute] of selected-problem "Yes"))
      foreach [antecedents] of this-rule
      [
           ask this-rule
              [
                set attribute (first ?)
                set value (last ?)
                set events lput (list attribute value) events
              ]
           ]
      foreach [consequents] of this-rule
           [
              set attribute (first ?)
              set value (last ?)
              add-events (lput (list attribute value) events) white white white
           ]
      ]
      repeat 5 [ change-layout ]
      display
end
```

NetLogo Code 9.11 How the rules-base is converted into a decision tree.

This procedure processes each rule in the rules-base by first initialising the list of events to contain a single event comprising the goal attribute with the word `"Yes"`. For the Sailing boats problem, for example, this will set the events list to be `[[boat "Yes"]]`. The effect of this is that all the rules will be connected to a central node at the centre of the decision tree. The procedure then appends all the antecedents as events to the event-list, and then for each of the consequents, it will add the consequent onto the end of this list, and will call `add-events` separately. This will result in a separate event path being created for each separate consequent. The result for the Sailing boats problem is shown in Figure 9.13. Note that there are eight leaf nodes to the tree, and these correspond to the eight rules in the rules-base.

Download free eBooks at bookboon.com

The reasoning process works by proceeding outwards from the central node asking questions and following the relevant link until a leaf is reached in the same manner as the New Zealand birds NetLogo model discussed in Chapter 4. The progress of the reasoning process at an intermediate stage is shown in Figure 9.13. The process started at the central node labelled `boat = "Yes".` It highlighted the two possible links it could follow – the link to the state labelled `number of masts = "one",` and the link to the state labelled `number of masts = "two".` The internal ordering of the links (as a result of when they were created) then resulted in the reasoning process moving to the first of those states and caused the question "How many masts are there?" to be asked. The user replied with "two" which meant that the traversal of the current path stopped, and the reasoning process then started following the other path instead. The next question asked was "What is the main mast position?" and the reply "forward of the short mast" forced the reasoning process to then move to the state labelled `main mast position = "forward of the short mast".` The next question asked was "What is the short mast position?" and the reply "forward of the helm" forced the reasoning process to end up at the state shown in Figure 9.13. The answer to the current question "What is the shape of the mainsail?" will result in the reasoning process ending up with the conclusion that the boat is a Jib-headed ketch if the answer was "triangular", and with the conclusion that it was a Gaff-headed ketch otherwise.

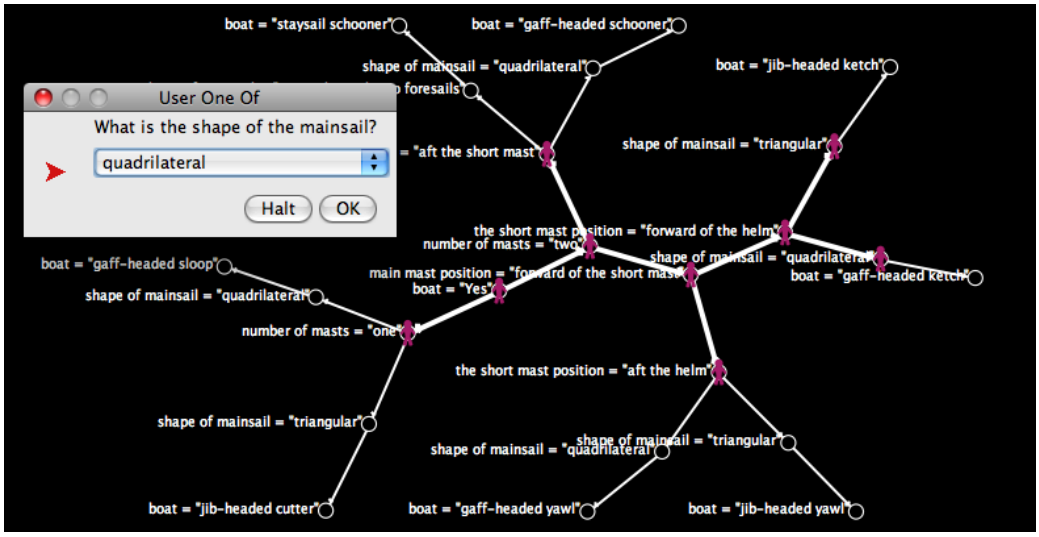


Figure 9.13 The event map of the Sailing boats decision tree showing the progress of the reasoning process.

The code that performs the decision tree reasoning process is defined by the `go-decision-tree-reasoning` procedure within the Knowledge Representation model in NetLogo. This code calls the `match-tree` procedure to walk the tree. It does this by calling itself recursively matching the tree states against the facts in the facts-base until there are no more out-path neighbours – that is, when a leaf node has been reached. The goal is achieved once a fact with the goal attribute (in this case `boat`) has been added to the facts-base.

Download free eBooks at bookboon.com

9.13 Knowledge and reasoning using semantic networks

The final method of knowledge representation we will examine is called semantic networks. Semantic networks are made up of interlinked nodes connected by arcs. The nodes represent objects, and arcs the relationships between objects. An example of a semantic network for the zoo animals problem is shown in Figure 9.14. There are many variants of semantic networks, with its origins dating from 1909 with the 'existential graphs' of Pierce (cited by Russell and Norvig, 2002). Semantic networks became popular in the 1970s, with important work done by Collins and Quillian (1969; for example). A prominent debate in AI pertaining to semantic networks concerns its relative merits compared with logic. Russell and Norvig point out, however, that semantic networks can also be considered a form of logic, as is the case with the other forms of knowledge representation discussed above. A more recent example of a semantic network is WordNet that is a large lexical database of English words that are grouped by synonyms.

NetLogo Code 9.12 shows how the semantic network for the zoo animals problem in the Knowledge Representation model is declared.

```
to-report create-state [state-label]
;; creates and returns a new semantic node
   let this-state nobody

   create-states 1
      [
      set this-state self
      set color sky
      set label-color white
      set label state-label
      set size 8
      set stream "state name"
      set event state-label
      ]
   report this-state
end

to create-slink [state1 slink-label state2]
;; creates a link between the two nodes state1 and state2 in the
;; semantic network

   ask state1
      [ create-path-to state2
      [ set label slink-label
      set label-color yellow
      set color blue
      ]
   ]
end

to setup-animals-semantic-network
;; sets up the nodes and links in the Zoo Animals semantic network

   let animal-state create-state "animal"
   set goal-state animal-state
```


Download free eBooks at bookboon.com

```
  let mammal-state create-state "mammal"
  let ungulate-state create-state "ungulate"
  let carnivore-state create-state "carnivore"

  let hair-state create-state "hair"
  let milk-state create-state "milk"
  let meat-state create-state "meat"
  let pointed-teeth-state create-state "pointed teeth"
  let claws-state create-state "claws"
  let forward-eyes-state create-state "forward eyes"
  let hooves-state create-state "hooves"
  let cud-state create-state "cud"
  let tawny-colour-state create-state "tawny colour"

  let dark-spots-state create-state "dark spots"
  let black-stripes-state create-state "black stripes"
  let long-neck-state create-state "long neck"

  let cheetah-state create-state "cheetah"
  let tiger-state create-state "tiger"
  let giraffe-state create-state "giraffe"
  let zebra-state create-state "zebra"

  create-slink cheetah-state "is a" animal-state
  create-slink tiger-state "is a" animal-state
  create-slink giraffe-state "is a" animal-state
  create-slink zebra-state "is a" animal-state

  create-slink ungulate-state "is a" mammal-state
  create-slink cheetah-state "is a" mammal-state
  create-slink tiger-state "is a" mammal-state
  create-slink cheetah-state "is a" carnivore-state
  create-slink tiger-state "is a" carnivore-state
  create-slink giraffe-state "is a" ungulate-state
  create-slink zebra-state "is a" ungulate-state

  create-slink mammal-state "has" hair-state
  create-slink mammal-state "gives" milk-state

  create-slink carnivore-state "eats" meat-state
  create-slink carnivore-state "has" pointed-teeth-state
  create-slink carnivore-state "has" claws-state
  create-slink carnivore-state "has" forward-eyes-state

  create-slink ungulate-state "has" hooves-state
  create-slink ungulate-state "chews" cud-state

  create-slink cheetah-state "has" tawny-colour-state
  create-slink cheetah-state "has" dark-spots-state
  create-slink tiger-state "has" tawny-colour-state
  create-slink tiger-state "has" black-stripes-state
  create-slink giraffe-state "has" dark-spots-state
  create-slink giraffe-state "has" long-neck-state
  create-slink zebra-state "has" black-stripes-state
end
```

NetLogo Code 9.12 The code that sets up the knowledge for the semantic network for the zoo animals problem.

Download free eBooks at bookboon.com

The `setup-animals-semantic-network` makes use of a reporter, `create-state`, for creating a state in the network and returning it, and a procedure, `create-slink`, for creating a link between two states. The former takes a single argument `state-label` that specifies the label associated with the `state`, and then a single state agent is created with the specified label, and this is then returned. The latter takes three arguments, the originating state `state1`, the label `slink-label` for the arc, and the destination state `state2`, and then a single link agent is created that links `state1` to `state2`. The semantic network is set up first by creating a state for the types of animals (mammal, ungulate, carnivore), for the features of animals (e.g. hair, milk, meat, pointed teeth, claws), and for each of the animals (tiger, cheetah, giraffe and zebra). Then the links between these states are set up one by one.

Although semantic networks are relatively easy to set up, the main problem is deciding on the type of knowledge that goes into the network – that is, what type of knowledge should be represented by the states and what type of knowledge by the arcs. A rule of thumb is that state labels should be nouns and arc labels should be verbs (as with the zoo animals semantic network shown in Figure 9.14). Two labels commonly used for arcs are "is a" and "has". For example, in the centre of Figure 9.14, there is a state labelled "tiger", and it has two arcs labelled "has" and three arcs labelled "is a" linking to five states. This represents the knowledge that a tiger has black stripes and a tawny colour, and that a tiger is an animal, a mammal, and a carnivore. The arc label "is a", as for frames, is used for performing inheritance and default style reasoning. Note that multiple inheritance is allowed, unlike some object oriented programming languages such as Java that only support single inheritance.

Click on the ad to read more

Download free eBooks at bookboon.com

Frames can be thought of as a variation of semantic networks, since each node can represent a single frame, and the arcs that lead out of the node are its slots, with the slot values being the nodes that the slot values lead towards. A semantic network can also be converted into an equivalent event map. This can be done in the Knowledge Representation model by setting the `KR-type` chooser in the Interface to `"semantic event network"`, and the model will initially draw the semantic network, and then replace it with the equivalent semantic event network.

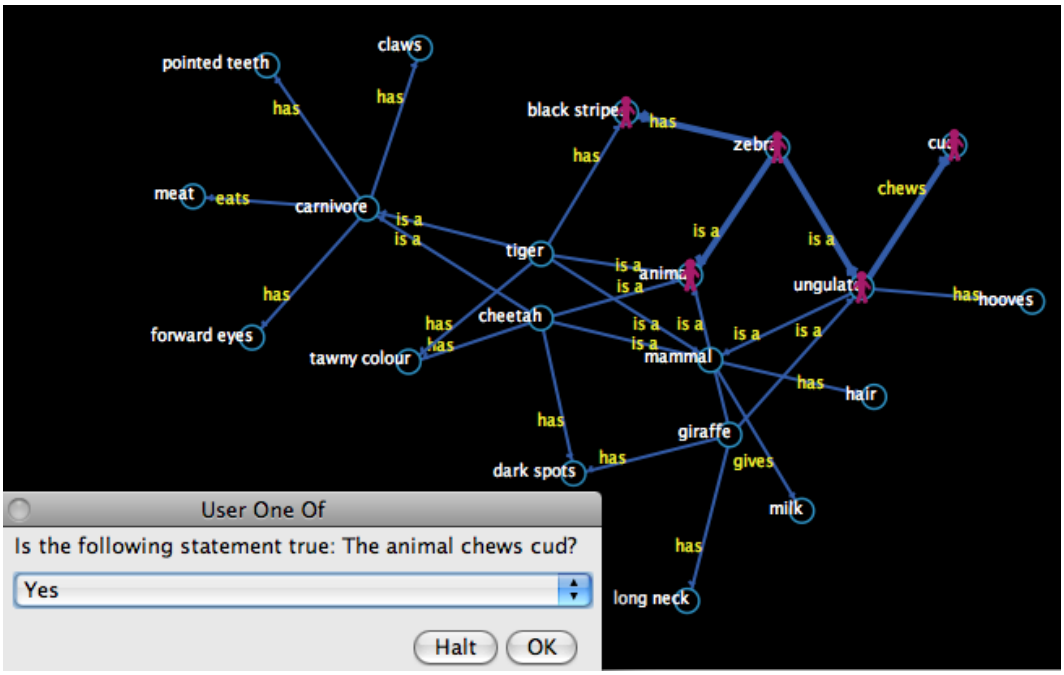


Figure 9.14 The zoo animals semantic network showing the progress of the reasoning process.

An important form of reasoning for semantic networks involves matching of states against the current facts in a manner analogous to the way frames are matched. Figure 9.14 shows an intermediate stage of the state matching process. In this case, the reasoning process started at the goal `animal` state shown at the centre right of the image. One of the in-path neighbours to this state was chosen at random to be the next state visited – in the case depicted in the image, the state chosen was the `zebra` state. The reasoning process then tried to match all the states associated with this state – the first chosen was the `black stripes` state which caused the question "Is the following statement true: The animal has black stripes?" to be asked. As the answer was affirmative, the reasoning process then started trying to match the `ungulate` state, with the `cud` state being chosen to be visited next, resulting in the question "Is the following statement true: The animal chews cud?" being asked as shown in the image.

Download free eBooks at bookboon.com

The code that performs the semantic network reasoning process is defined by the `go-semantic-network-reasoning` procedure within the Knowledge Representation model in NetLogo. This code makes use of the `match-network-state?` reporter that returns true if a particular state in the semantic network matches the current facts. The reporter works by processing all the out-path neighbours in turn until a match is found. First it clones a walker agent to move to the particular neighbouring state. If the label for the link to that state equals "is a" then it will recursively call itself with the newly cloned walker to check that the neighbouring state also matches first since it is an "is a" (parent) state. Otherwise it will use the link label and the label of the neighbouring state to construct a proposition in order to verify whether it is true or not, in the process asking the user if necessary if the proposition's attribute is not already known (i.e. exists in the facts-base). This particular implementation allows link labels to contain the keyword "not" inside them (for example, "is not" can be used instead of "is", "can not" instead of "can" and "has not" instead of "has"). This was done to accommodate the particular style of knowledge in the New Zealand birds problem so that the semantic network form of representation could be directly compared with the other forms of representation. The proposition is verified by checking to see if it exists in the current facts-base, and the reporter will return false if it isn't.

The main `go-semantic-network-reasoning` procedure works by calling match-`network-state?` for all the in-path neighbours of the goal state whose link to it has the label "is a". It continues until it finds one of the neighbouring states that match. For example, for the zoo animals problem, it will try (in no particular order) each of the four states – `tiger, cheetah, giraffe` and `zebra` – that are linked to the animal state with the link label "is a". If it cannot find a neighbouring "is a" state that matches, it will return false. This particular implementation causes a different set of questions to be asked each time the procedure is run (as a result of the ask command using `in-path-neighbours`). If such behaviour confuses the user, then a more consistent behaviour for the expert system can be obtained by using a `foreach sort` command instead to ensure that the neighbouring states are visited in a specific order.

A problem with semantic network based reasoning, apparent when running the Knowledge Representation model, is that it often can jump around the network asking seemingly unrelated questions and as a result confusing the user of the system. There is also a loss of context as the reasoning process moves from one state to the next. This is unlike the decision tree or event map representations that follow defined paths where the location of the state reflects the state of the reasoning based on the prior sequence of states visited.

Download free eBooks at bookboon.com

9.14 Summary and Discussion

Knowledge representation is a fundamental task that is required for intelligent behaviour. This chapter has looked at various ways of representing knowledge such as maps, rules, logic, frames, decision trees and semantic networks. A representation based on event maps has been used to visualise the different types of representation and to show how they can be converted from one form into another. The aim has been to discuss knowledge from an embodied, situated perspective by showing that knowledge and reasoning are related to the movement of an agent around an environment. Knowledge has been recast using the event map representation as points in abstract n-dimensional space environments, and reasoning as a search of that space.

We can also relate knowledge to behaviour in the following way. Agents that have knowledge can be said to exhibit knowing behaviour, whereas agents without knowledge need to exhibit some form of searching behaviour instead. From on observer's frame of reference, if we see an agent searching for something, we can infer the reason for this is due to a lack of knowledge of where what they are searching for may be found. Hence, we can say that knowing behaviour is the opposite of searching behaviour (or to use another analogy, two sides of the same coin). For example, a person exhibits knowing behaviour when she goes straight from the Zoo to the Boat Pond in New York's Central Park without having to look at a map. Another person, perhaps an antipodean visitor from down under, will have to use the map to 'find' the way instead (note the use of a searching metaphor here to describe the process). He will inevitably end up having to physically search at some stage after he gets lost along the way when the map does not fit with reality.

American online
LIGS University
is currently enrolling in the
Interactive Online BBA, MBA, MSc,
DBA and PhD programs:

- enroll by September 30th, 2014 and
- save up to 16% on the tuition!
- pay in 10 installments / 2 years
- Interactive Online education
- visit www.ligsuniversity.com to find out more!

Note: LIGS University is not accredited by any nationally recognized accrediting agency listed by the US Secretary of Education.
More info here.

Click on the ad to read more

Download free eBooks at bookboon.com

A summary of important concepts to be learned from this chapter is shown below:

- The symbolic 'top-down' approach to AI posits that we can represent knowledge using symbols.
- The sub-symbolic 'bottom-up' approach to AI emphasizes the processing of stimuli rather than symbols.
- Conceptual spaces theory posits a middle ground where concepts represented as regions in an *n*-dimensional space are the central characteristic of human knowledge and reasoning.
- Maps represent topographical knowledge, but can also be used to represent other forms of knowledge in an abstract way.
- Event maps can be used to represent and visualize *n*-dimensional data in two dimensions.
- Rules are a form of knowledge representation that uses IF-THEN (condition-action) statements.
- There are two main types of reasoning using rules: forward reasoning which starts from the facts and tries to prove a specific goal is true; and backward reasoning which starts with the goal first, and works backwards to show that the facts support the goal's hypothesis.
- Logic is a mathematical form of reasoning.
- Frames represent knowledge using prototypical concepts and use inheritance and matching as two types of reasoning.
- Decision trees represent knowledge graphically as a directed tree, with questions or decisions associated with each node in the tree, and conclusions at the leaf nodes. Reasoning proceeds by using the facts to guide which branch of the tree to follow.
- Semantic networks also use a graphical form for representing knowledge, with nodes representing concepts, and arcs representing relations between concepts. Semantic networks are closely related to frames.

The code for the NetLogo models and the Central Park world described in this chapter can be found as follows:

Model	URL
Central Park Events	http://files.bookboon.com/ai/NetLogo/Central-Park-Events.nlogo
Colour Cylinder	http://files.bookboon.com/ai/Colour-Cylinder.nlogo
Knowledge Representation	http://files.bookboon.com/ai/Knowledge-Representation.nlogo
Map Drawing	http://files.bookboon.com/ai/Being-Map-Drawing.nlogo

World	URL
Central Park	http://files.bookboon.com/ai/Map-Drawing-Central-Park.csv

Download free eBooks at bookboon.com

10 Intelligence

HAL: I'm afraid. I'm afraid, Dave. Dave, my mind is going. I can feel it. I can feel it. My mind is going. There is no question about it. I can feel it. I can feel it. I can feel it. I'm a…fraid. Good afternoon, gentlemen. I am a HAL 9000 computer. I became operational at the H.A.L. plant in Urbana, Illinois on the 12th of January 1992.

Stanley Kubrick and Arthur C. Clarke. 2001: A Space Odyssey.

Intelligence without Reason *can be read as a statement that intelligence is an emergent property of certain complex systems – it sometimes arises without an easily identifiable reason for arising.*

Rodney A Brooks. 1987. Intelligence without reason.

The purpose of this chapter is to explore what intelligence is, and see how we might go about building systems that exhibits intelligence. The chapter is organized as follows. Section 10.1 looks at the various ways people have defined intelligence over the centuries. Section 10.2 explores whether we can have intelligence without representation or reason. Section 10.3 looks at some of the remarkable achievements of AI systems, but points out there are many things these systems still can't do. Section 10.4 explains why we need design objectives for Artificial Intelligence and Section 10.5 looks at what makes good objectives in general. Section 10.6 suggests several design objectives for AI, and highlights various issues with their formulation. Section 10.7 discusses how we can build believable agents in order to achieve some of the design objectives. Section 10.8 continues the discussion with a specific focus on problem solving.

10.1 The nature of intelligence

What is the nature of intelligence? That is a question that has been pondered, and debated for thousands of years. Many people over the centuries have offered their own view on the matter, as illustrated by the quotes provided in Table 10.1.

Download free eBooks at bookboon.com

Name	Birth-Death	Profession	Quote
Lao Tzu	Unknown; 6th century BC	Chinese record-keeper (in the Zhou Dynasty court)	"Knowing others is intelligence; knowing yourself is true wisdom. Mastering others is strength; mastering yourself is true power. If you realize that you have enough, you are truly rich."
Socrates	469–399 BC	Greek philosopher	"I know that I am intelligent, because I know that I know nothing."
Leonado da Vinci	1452–1519	Italian scientist, sculptor, various others	"Anyone who conducts an argument by appealing to authority is not using his intelligence, he is just using his memory."
Abigail Adams	1744–1818	American First Lady	"I've always felt that a person's intelligence is directly reflected by the number of conflicting points of view he can entertain simultaneously on the same topic."
Albert Einstein	1879–1955	German physicist	"The true sign of intelligence is not knowledge but imagination."
Bertolt Brecht	1898–1956	German Poet and Playwright	"Intelligence is not to make no mistakes, but quickly to see how to make them good."
Arthur Samuel	1901–1990	American AI pioneer	"[T]he aim [is] to get machines to exhibit behavior, which if done by humans, would be assumed to involve the use of intelligence."
Alan Turing	1912–1954	British computer scientist, mathematician	"A computer would deserve to be called intelligent if it could deceive a human into believing that it was human."
Susan Sontag	1933–2004	American Writer	"Intelligence is really a kind of taste: taste in ideas."
Carl Sagan	1934–1996	American Astronomer, Writer, Scientist	"Knowing a great deal is not the same as being smart; intelligence is not information alone but also judgment, the manner in which information is collected and used."
Linus Torvalds	1969–	Finnish software engineer	"Intelligence is the ability to avoid doing work, yet getting the work done."

Table 10.1. Some quotes on the nature of intelligence.

It seems that everyone has their own opinion on what intelligence is or isn't. Intelligence is a concept that everyone knows about, but understands differently. As we have seen in the previous chapter, the way each person understands a particular concept will have its own unique 'flavour'. Perhaps one of the most interesting quotes above is by Susan Sontag that uses an analogy between taste and intelligence. Taste is a complex sensation in four dimensions – sweetness, sourness, bitterness and saltiness. Similarly, intelligence is a complex concept, with multiple dimensions.

Download free eBooks at bookboon.com

Intelligence is multi-faceted – its nature cannot be defined using one of these quotes alone; it requires all of them. As an analogy, try describing the Mona Lisa. One person's description of the painting may be anathema to another person. To imagine that we can distil the Mona Lisa down to a few written words, and then naïvely believe other people will agree with us that it is the one and only definitive description, is like believing that people should only ever eat one type of food, or enjoy looking at one type of painting, or read one type of book. The Mona Lisa painting continues to inspire people to write more and more words about it. Similarly, intelligence is not something we can elucidate definitively. But that will not stop people from continuing to do so, since in so doing further insights can be gained into its nature.

Although definitions of intelligence are fraught with problems, we can look for desirable properties of intelligence that we can help us to describe the nature of intelligence. In other words, we can help define the nature of intelligence by describing what it 'looks' like or what it 'tastes' like. Using the taste analogy, we can think of these properties as being 'ingredients' in a recipe for intelligence – we need to mix them together in order to make a particular taste, which some people will like, while others may not, preferring alternative tastes. For example, we can use the analogy of African and Australian explorers trying to describe what a giraffe or platypus looks like to someone who has never seen it. These explorers will use words (concepts) that they are familiar with, such as 'long neck' and 'fish-like tail', but their description will be 'flavoured' by their own unique perspective. Whatever words they come up with, they will have over-emphasized certain features and ignored other important ingredients.

Click on the ad to read more

Download free eBooks at bookboon.com

Similarly, AI researchers with a background in knowledge engineering and the symbolic approach to AI will describe intelligence using ingredients such as the following:

- the capacity to acquire and apply knowledge;
- the ability to perform reasoning; and
- the ability to make decisions and plan in order to achieve a specific goal.

AI researchers who prefer a behavioural-based approach will describe the intelligent behaviour of embodied, situated agents using ingredients such as:

- the ability to perform an action that an external intelligent agent would deem to be intelligent;
- the ability to demonstrate knowledge of the consequences of its actions; and
- the ability to demonstrate knowledge of how to influence or change its environment in order to affect outcomes and achieve its goals.

If we think of intelligence using an analogy of mapping, as discussed in the previous chapter, then we might use the following ingredients to describe intelligence:

- the ability of an embodied, situated agent to map environments, both real and abstract (i.e. recognize patterns to provide useful simplifications and/or characterizations of its environments);
- the ability to use maps to navigate around its environments;
- the ability to update its maps when it finds they do not fit reality; and
- the ability to communicate details of its maps to other agents.

It is important to realise, however, that these are not definitive descriptions, just ingredients in alternative recipes for intelligence.

In the previous chapters, we have seen various examples (implemented as models in NetLogo) that have demonstrated some of these ingredients. In some respects, these models have exhibited a small degree of intelligence in the sense that if we observed a human agent with the same behaviour, we would deem that to be a sign of intelligence. In the next volume of this book series, we will also see other models that will demonstrate more advanced technologies. It can be argued, however, that these examples show no true intelligence – but of course that depends on your own perspective, and the ingredients with which you choose for your own recipe for intelligence.

Download free eBooks at bookboon.com

10.2 Intelligence without representation and reason

In the last chapter, a question was asked about whether it was possible to have knowledge without representation. Similarly, we can ask ourselves the following question: "Is it possible to have intelligence without representation?" In a seminal paper, Rodney Brooks (1991) considered exactly this same question. In another paper (Brooks, 1991), he also considered the related question: "Is it possible to have intelligence without reasoning?" As discussed in the previous chapter, Brooks favours the embodied, situated approach to AI – the sub-symbolic paradigm rather than the classical symbolic paradigm. When he talks about the possibility of intelligence without 'representation', he means that an embodied, situated agent does not need to explicitly represent its environment – it can simply react to it. There is no need for the agent to have an explicit knowledge base about the world it is situated in since the agent can directly 'consult' it by interacting with it.

Brooks goes further and states that intelligence is an emergent property of certain complex systems (see quote at the beginning of this chapter). Brook's ideas are interesting in that it raises the possibility that, in designing AI systems, we may not have to do all the work ourselves. If we can find the right way of setting up the initial conditions of the system, the system itself will do the work for us, and through self-organisation, intelligence will emerge as a result. Unfortunately, although this idea is very intriguing, no one as yet has figured out how to set up the necessary initial conditions.

As pointed out by other researchers, Brook's approach to AI is not necessarily free of representation or reasoning. The 'knowledge' needed for the reactive-based insect-like robots that his team has built to demonstrate his approach is in fact represented within the subsumption architecture and finite state automata used to design the robots. (We will learn more about this in the next volume). It can also be argued that this architecture does a basic form of reasoning, although this is not the same as the traditional symbolic-based form of reasoning in classical AI terms. If we accept these arguments, however, then we must also agree that the ants and termites described in previous chapters also use representation and perform reasoning – since the ants' 'knowledge' is defined by its instinctive behaviours and distributed throughout the environment, not in some internal state that is dynamically updated within a single agent.

Brook's embodied, situated approach focuses on specific ingredients for intelligence. Gärdenfors conceptual spaces theory offers an alternative hybrid approach that tries to explain how further ingredients for intelligence may arise, conceptual as well as symbolic and sub-symbolic. Further exploration of these ideas is required if we are to achieve the necessary ingredients for human-level intelligence.

Download free eBooks at bookboon.com

10.3 What AI can and can't do

The field of Artificial Intelligence has come in for some criticism over the years for the grandiose predictions with which it is often associated and the perceived failure of the field to deliver on them. Many researchers in the past have made the mistake of seriously underestimating the difficulty of the task. Herbert Simon predicted that by 1967 a computer would be world champion in chess – it took until 1997 before the world champion was first beaten by a computer (see below). He also predicted (again by 1967) that a computer would be able to discover a new mathematical theorem and prove it. The discovery of mathematical theorems has proved difficult – an important new theorem has yet to be discovered by a computer – although techniques for automated theorem proving have been around for some time (again see below). Marvin Minsky, again in 1967, predicted that the problem of creating AI would substantially be solved within a generation.

Click on the ad to read more

Download free eBooks at bookboon.com

The predictions of the early AI researchers provided inspiration for Stanley Kubrick and Arthur C. Clarke's HAL 9000 computer that became operational in their story in 1992 (see quote at the beginning of this chapter). Almost two decades latter, their vision of computers with emotional and conversational capabilities is still far from reality. Further claims have been made for more specific sub-fields of AI – it was claimed back in 1957, for example, that machine translation would be solved within three to five years. Anyone today using online machine translation services, or machine translation software, will be aware this is far from the case. Present predictions claim that we will have computers with greater processing power than the brain by 2020, and robots with human intelligence by 2050 (see Section 1.3), and even robots with the ability to beat humans at football (also by 2050).

Ray Kurzweil (1990) in *The Age of Intelligent Machines* has made many predictions concerning computer technology and AI specifically, many of which have come true, but some which have not (such as his prediction that by 2009, users would rely mainly on speech recognition to communicate with their PCs rather than using keyboards). In *The Singularity is Near: When Humans Transcend Biology* (Kurzweil, 2005), he postulates what he calls a 'Singularity' occurring within our lifetimes (in 2045). This will be a disruptive world-altering event that will forever change the course of human history and will happen when AIs surpass human beings as the most intelligent entities on the planet. From that time on, technological development will be taken over by machines, and we, as humans, will no longer be able to keep up.

It is very difficult to predict the future, especially when it comes to technological advancement, and whether Kurzweil's predictions eventuate we will have to wait and see. Rather than predicting what AI might be able to achieve in the future, we can instead examine what AI has achieved in the past, and also have a look at the present. For example, Table 10.2 lists twelve tasks to which we can apply AI.

The first nine items on this list are from an exercise (1.7) by Russell and Norvig (2002). In all of these tasks, there have been 'successes' for AI to varying degrees. The degree of success, however, is open to debate. For example, what is a decent game of table tennis? The answer is subjective – the difference in standard between a child or a complete novice and an Olympic champion is substantial. TOSY's TOPIO robot is capable of interacting with a human to play table tennis in a limited sense. But fully autonomous competitive tournament play is well beyond its capabilities.

Download free eBooks at bookboon.com

1. Play a decent game of table tennis.
2. Drive in the centre of Cairo.
3. Buying a week's worth of groceries at the market or on the Web.
4. Play a decent game of bridge at competitive level.
5. Discover and prove mathematical theorems.
6. Write an intentionally funny story.
7. Give competent legal advice in a specialized area of law.
8. Translate spoken English into spoken Swedish in real time.
9. Perform a complex surgical operation.
10. Recognize and aesthetically appreciate the Mona Lisa.
11. Create a virtual Elvis.
12. Do all of the above.

Table 10.2. Twelve things AI can and can't do (based on Russell and Norvig, 2002).

The DARPA Grand Challenge for 2007 included an 'Urban Challenge' where self-driving vehicles had to negotiate a 96 km course in less than 6 hours while obeying all traffic regulations, and avoiding obstacles and other traffic. Six vehicles successfully completed the course. However, this test was done within the controlled environment of the George Air Force Base – the centre of Cairo (and other major cities) is another matter, which raises the difficulty to a whole new level due to the unpredictability of other human drivers and pedestrians. General Motors have recently announced a computer controlled system that uses lasers and a video camera for the 2008 Opel Vectra called 'Traffic Assist' that will be able to autonomously drive the car in heavy traffic at up to 60 mph. It will have the ability to recognize signs and avoid obstacles, control the accelerator, steering and braking, and make course and speed changes as needed. Whether people would be willing to let such as system loose at the moment in the middle of Cairo is another matter.

Buying groceries is a task that people do frequently. Most people without disabilities find the task relatively easy to accomplish. And yet, even the most fundamental of tasks such as bagging your groceries (a problem considered by Patrick Winston in 1992) or Internet shopping (Russell and Norvig) is difficult for an AI system. Negotiating the way around a busy market place, recognizing and examining the quality of the items that are for sale, and directly haggling with the stall owners over price, are still well beyond current AI systems.

Download free eBooks at bookboon.com

A computer program GIB was able to outplay all but 11 of the world's top players at the World Bridge Championships in Lille, France back in 1998. The best computer programs, such as Jack, can play on equal terms against mid-level human players. However, the field of computer bridge is still in its infancy. AI has been successful playing chess against human components, most notably with IBM's Deep Blue beating the world champion Gary Kasparov in 1997, and more recently, Deep Fritz defeating the world champion Vladimir Kramnik in 2006. For the game of Go, where the search space is much larger, computer programs are beginning to play at the professional level, but the gap is still considerable to the playing level of really strong human players.

Automatic theorem proving or automatic deduction has been well researched within AI. The theorem prover OTTER by Art Quaife, for example, has proven over 400 theorems of set theory, over 1200 theorems of number theory, as well as Euclidean geometry theorems and Gödel's incompleteness theorems. The more difficult challenge now beginning to be investigated is automated theorem discovery given a set of axioms of a domain. A proviso that makes this particularly difficult is that the discovered theorems should be both interesting to humans, and also difficult to prove for humans and automatic theorem proving systems.

Natural language generation is a venerable sub-field of natural language processing that involves the automatic generation of written text. Story generation deals exclusively with the problem of getting a computer to automatically write a story. Often the output produced by some story-generating computer programs is unintentionally funny, because of the absurdity of the combination of words and phrases that often result from a random selection process. Generating intentional humour that is both novel and unique is much more complex. Similarly, producing a story with the complexities of James Joyce's *Ulysses*, the 'tactile brilliance' and warmth of Harper Lee's *To Kill a Mockingbird*, the satirical humour of Joseph Heller's *Catch 22* and the wry absurdity of Douglas Adam's *Hitchhiker's Guide to the Galaxy: A Trilogy in Five Parts* is far beyond anything a computer can presently produce.

Techniques from the AI fields of expert systems and information retrieval systems have been adapted to build legal advice systems and reproduce legal reasoning of judges. An expert system is consulted in order to solve a particular case, and an information retrieval system is used to search documentation to find similar cases. Clearly these legal advice systems do not yet have the abilities of real-life lawyers such as those arguing the legal complexities in the *Bright Tunes Music v. Harrisongs Music* case (where it was successfully argued that George Harrison's *My Sweet Lord* was a copyright infringement of the Chiffon's *He's So Fine*). If these systems did have the same abilities, we would now be able to replace all lawyers with computers.

Download free eBooks at bookboon.com

Speech recognition and machine translation software of varying capabilities have been around since the early 1960s. Search engines are increasingly using the latter to help people read online documents in another language. The automatic processing of natural language is an especially difficult problem for computers because the tolerance of native speakers to errors is very low. For example, speech recognition systems often advertise accuracy results above 90%. This sounds adequate enough, but we would find it very difficult to tolerate listening to another person if he made as many as 10 errors for every 100 words spoken. Results for machine translation are considerably lower in comparison, often as a result of a mismatch between the two different conceptual systems expressed by the source and target languages (for a discussion of some issues concerning concepts, see Section 9.4). Compounding this are the difficulties of performing simultaneous translation of spoken language where there can be significant noise (for example, see Section 7.2 for an example of the differences between written and spoken language). Also, simultaneous translation requires conversational ability. Despite many websites claiming otherwise, no current chatbot or conversational agent is capable of holding a believable conversation with a person, or is capable of passing the Turing Test. These systems conduct only a resemblance of a conversation, and their inadequacies are readily apparent after a short conversation.

Click on the ad to read more

Download free eBooks at bookboon.com

Robotic surgery has made significant advances in recent years. It can be characterised by techniques that employ increasing levels of robotic autonomy. Remote surgery (or telesurgery) uses robotics to allow surgery to be performed remotely by a human operator. Minimally invasive surgery typically involves using keyhole surgical devices with remote-control manipulation. Unmanned surgery involves the use of fully autonomous robotic surgeons and has had recent success. The first unmanned robotic surgery took place in May, 2006 in Italy on a 47 year old male to correct heart arrhythmia, and the operation was rated as better than that performed by an above average human surgeon. In January 2009, the first fully robotic-assisted kidney transplant was performed in New Jersey. These major advances in robotics and medical procedures have resulted in much greater surgical precision, and smaller incisions, leading to less pain, less blood loss and shorter healing times.

Facial recognition software has made substantial advances over the years. It is now being used in many places such as airports to automatically recognize criminals and terrorists, and in security systems for biometrics. Facial recognition software, for example, was used to identify 19 people with pending arrest warrants at the Super Bowl in Tampa Bay Florida in 2001. However, the software is far from perfect, struggling with low resolution images and poor lighting, there are issues with privacy and they can be circumvented by people wearing sunglasses or using varied facial expressions such as a large smile. There were problems with early systems – for example, deployment in the London borough of Newham failed to recognize a single criminal over a period of several years. Recent advances have seen significant improvement in accuracy, outperforming humans in some experiments including showing an ability to differentiate between identical twins. Clearly, a straightforward application of this software would be the recognition of a painting such as the Mona Lisa. Much more difficult is the classification of a large number of images for image retrieval purposes on an Internet scale. Search engines still rely mostly on surrounding text in online documents to help identify images since this achieves effective results, is much faster and requires much less resources. A full aesthetic appreciation of artistic images, however, such as the Mona Lisa, that produces a unique and critical analysis of the differences in painting style and content, is also beyond current technology.

Motion capture (also called mocap) is a technique used for animation in movies and computer games that captures the movement of a human or animal and then generates realistic movement virtually using a digital model. Mocap techniques used for Peter Jackson's King Kong movie, for example, enabled virtual doubles to be created for the main characters that are so good they are difficult to tell apart from the real thing. A virtual Elvis that improvises in real-time with the ability to sing new songs not pre-recorded when he was alive (unlike the virtual Elvis that sang with Celine Dion on American Idol) is still beyond the state-of-the-art.

Download free eBooks at bookboon.com

Collectively, humans are capable of doing all of the tasks listed in Table 10.2. However, it is unlikely there is a single human who exists today who can do all of these tasks by herself, unless she is a bridge-playing art-appreciating Swedish surgeon who is a grandmaster in chess dabbling in law, mathematics and computer animation that writes funny stories in her spare time (if she has any). To expect an AI to do all of these tasks is perhaps raising the bar too high on our expectations of what AI is or should be capable of.

10.4 The Need for Design Objectives for Artificial Intelligence

One of the major difficulties for Artificial Intelligence research is that it has yet to hit upon a workable definition of what intelligence is (perhaps because one is not possible as alluded to in Section 10.1). Defining intelligence is fraught with problems, and has been a source of philosophical debates for centuries. Peter Norvig (2007), for example, has stated that in writing his AI textbook *Artificial Intelligence: A Modern Approach* (Russell and Norvig, 2002), they tried to define Artificial Intelligence by avoiding "philosophical debates" and instead concentrated on describing how to build the "best possible programs". From a pragmatic point of view, when designing systems that exhibit artificial intelligence, we can take an engineering perspective – we know what intelligence is, we will know when our systems demonstrate intelligence when we observe them, so we do not necessarily need to define what it is.

However, this avoidance of providing a definition of intelligence upfront results in a lack of preciseness in the literature and research. The terms "artificial intelligence" and "intelligent systems" are often misused, with very little justification as to why a system is "intelligent". Without a definition of intelligence, the problem becomes that it is no longer clear whether what has been built really is in fact "intelligent". Early AI systems in the 1970s and 1980s suffered from a lack of evaluation – there was a rush to build new systems, but often very little evaluation was undertaken of how well the systems worked. Without a working definition of intelligence, the same problem occurs now with current AI systems – how can we evaluate how effective our AI system might be if we do not have a definition of what it should be (or even achieve or do)?

We can, however, avoid the philosophical pitfalls, and rather than attempting to define intelligence, and making a claim that this definition is the "right" one, instead we can propose design objectives for the AI systems that we wish to build. We can state that, from a design perspective, these are the objectives that we wish our AI system to achieve. We then are free to propose these objectives, add new ones and alter existing ones as we see fit in order to improve the design because we ourselves are the designers. Other people may criticize whether our objectives are worthwhile from an engineering perspective (i.e. they do not produce "good" programs, or are not as good as other approaches), but the philosophical debate is only relevant in the sense that it can help inform and improve our design. Evaluation now becomes simpler – all we need to do is evaluate whether we have achieved our design objectives.

Download free eBooks at bookboon.com

10.5 What are Good Objectives?

However, what are some good objectives? First, we need to look at what makes a good objective. In project management, there is a well-known acronym that is used to guide the crafting of good objectives – SMARTER (see Table 10.3) – where each letter in the acronym stands for desirable attributes. The designer can ask questions about the suitability of an objective he or she has proposed – whether it is *specific*, whether it can be *measurable*, whether it is *achievable*, whether it has a definite *time-limit*, and so on. Devising good objectives, however, is more of an art than a science, and this is perhaps reflected by the disagreement on the terms used for each of the letters. Perhaps the terms most relevant for Artificial Intelligence research are the first four major terms – the need for the objectives to be specific, measurable, achievable, and to a less extent realistic. The nature of these attributes will necessarily change with time due to progress being made – as more knowledge is gained of what is achievable, of what is now realistic compared to what was previously, and from improvements made in methods to evaluate or measure the systems that have been built. The design objectives proposed in this book, based on existing literature and insights of the author, should be considered more as a moving target rather than fixed in stone.

Letter	Major Attribute	Minor Attributes
S	Specific	Significant, Stretching, Simple
M	Measurable	Meaningful, Motivational, Manageable
A	Achievable	Attainable, Agreed, Assignable, Appropriate, Actionable, Action-oriented
R	Realistic	Relevant, Results/Results-focused/Results-oriented, Resourced, Rewarding
T	Time-bound	Time framed, Timed, Time-based, Timeboxed, Timely, Timebound, Time-Specific, Timetabled, Trackable
E		Exciting, Evaluated, Ethical
R		Recorded, Rewarding, Reviewed

Table 10.3. SMARTER objectives, based on (SMART, 2008).

Perhaps the most famous objective in recent living memory, stated as a quote at the beginning of this volume, was the objective "to go to the moon" set by President John F. Kennedy in 1961. This objective is clearly specific, measurable and time-bound. Ultimately, it was shown to be achievable and realistic, as the objective was met.

10.6 Some Design Objectives for Artificial Intelligence

What, however, are some good design objectives for Artificial Intelligence? To devise suitable objectives, we first need to propose an overall design goal. One design goal might be that we strive to mimic human intelligence. The overall motivation for doing this is two-fold – first, in order that we can build intelligent artefacts that might aid us in some manner, and secondly, in order that we might develop a better understanding of own intelligence as a result.

Download free eBooks at bookboon.com

Yet our own intelligence is complex, puzzling and in many respects hidden from us. We have labels for aspects of our intelligence that we often use to distinguish ourselves from other animals, such as knowledgeable behaviour, intelligent behaviour, rationality, self-awareness, thoughtfulness and consciousness, and perhaps we can use these for our design objectives, as shown below.

Design Principle 1:	An AI system should be an agent-oriented system.
Design Goal 1:	An AI system should mimic human intelligence.
Design Objective 1.1:	An AI system should act in a knowledgeable way.
Design Objective 1.2:	An AI system should act intelligently.
Design Objective 1.3:	An AI system should act rationally.
Design Objective 1.4:	An AI system should act as if it is self-aware.
Design Objective 1.5:	An AI system should act as if it thinks.
Design Objective 1.6:	An AI system should act as if it is conscious.

Design 10.1 Design for an Artificial Intelligence system.

Click on the ad to read more

Download free eBooks at bookboon.com

An obvious failing with these design objectives is that they clearly break the SMARTER objectives mnemonic for all the major attributes:

- They are not specific enough since we currently have no workable definitions for knowledge, intelligence, rationality, self-awareness, thoughtfulness and consciousness, at least in the sense that they might help inform us as to how to build systems that exhibit such attributes.
- It is not at all clear whether these objectives are achievable – in the next decade, or our lifetime, or even at all. Hence, these objectives may not even be realistic.
- And setting a definite time-bound on them is currently out of the question. We certainly do not want to fall into the same trap that previous AI advocates fell into in previous decades when they professed that all manner of solutions would be found in the "not too distant future".

So, clearly, these do not seem to be very good objectives. Yet the objective that a system act in a knowledgeable way (Design Objective 1.1), is the implicit objective for a knowledge-based system, Design Objective 1.2 is the implicit objective for intelligent systems, and the others are key aspects of human intelligence that we will need to mimic to achieve Design Goal 1.

Clearly, we need to be much more specific with our design objectives. In addition, it is not clear how achievable each of these objectives is. At first glance, regarding the first objective, that of acting in a knowledgeable way, one could argue that such systems already exist today. Therefore, this objective might seem to be more readily achievable than the other objectives. But what does it really mean to act in a knowledgeable way? Moreover, what do we mean by 'knowledge' for that matter? If we mean that the agent-oriented system must have sufficient knowledge of its environment, itself and other agents in order that it can act in an knowledgeable manner, and demonstrate understanding of that knowledge, then achieving knowledge may be as difficult as achieving any of the other objectives. It may in fact be the key to the other objectives, and once it is achieved, the others may perhaps be achieved more easily.

One could also argue that the objective that the system exhibit intelligent behaviour (Design Objective 1.2) already covers the other objectives – a system must already exhibit knowledge, rationality, self-awareness, thoughtfulness and consciousness if it is to mimic intelligent human behaviour. However, it depends on how we define these properties. If we wish to use the term 'intelligence' in a manner similar to how we use it in English, this would suggest that a narrower definition might be more appropriate.

For example, we can say (in English) that a mathematician exhibits intelligence when solving an equation, and an inventor exhibits intelligence when creating a new system design that is patentable. Yet, computer systems have already demonstrated the ability to do both tasks in particular domains. So hence, one can claim that computers have already exhibited intelligence, at least in the narrow sense that the term is being used in this context. However, although everyone would agree that the mathematician and inventor are thoughtful and conscious, very few people would agree that these computer systems exhibit such properties.

Download free eBooks at bookboon.com

An important aspect of intelligence is the ability to solve problems. AI systems have demonstrated a wide variety of problem-solving capabilities as described in Section 10.3, with varying degrees of success. However, AI systems have not yet demonstrated the ability to make a decent effort at solving all of these problems unaided, without the benefit of solutions devised by humans, by learning how to solve them from scratch by either being taught by an external teacher or by progressive improvement through trial and error. A mark of human intelligence is that we have the ability to solve complex tasks by starting out as novices and learning through experience how to become experts. Importantly, we can also adapt solutions from one problem domain to another, innovating as a result, and we also have the ability to come up with completely novel solutions.

One way of making our design objectives more specific is to clearly state how we are going to measure when they have been achieved. We can perhaps use the Turing Test as a candidate test for conversational intelligence to make Design Objective 1.2 more specific. But what about the other design objectives? Are there other tests we can use, or invent, that might help us out? Indeed there is – for example, there exists a well-known test for self-awareness.

Need help with your dissertation?

Get in-depth feedback & advice from experts in your topic area. Find out what you can do to improve the quality of your dissertation!

Get Help Now

Go to www.helpmyassignment.co.uk for more info

Helpmyassignment

Click on the ad to read more

Download free eBooks at bookboon.com

Thought Experiment 10.1: The Mirror Test for Self-Awareness.

The Mirror Test, proposed by Gordon Gallup Jr. in 1970, is a test for self-awareness in animals. The idea is to place the animal in a room with a mirror. Two spots are surreptitiously marked on the animal using dye (perhaps while the animal has been anesthetized) – one is the control spot placed in a hidden but accessible part of the body, the second is only visible via a mirror (for example on the forehead between the eyes for primates). The animal is said to pass the self-awareness test if it notices the second spot but ignores the first thereby demonstrating that the animal is able to recognize its own reflection in the mirror. Animals said to have passed the mirror test are the great apes (bonobos, chimpanzees, orangutans and gorillas), bottlenose dolphins, killer whales, elephants, and European Magpies.

Like the Turing Test for intelligence outlined in Thought Experiment 1.1, the Mirror Test for Self-Awareness has its critics. Although extensively tested on primates, some argue that the test is not well-suited to animals who do not rely on vision as their main sense, such as dogs who rely more on smell, or animals who do not have stereoscopic vision (rabbits and deer), or animals who may shy away from threatening eye contact. Therefore, the test is only a test of abilities closely matching that of humans.

Interestingly, a robot named Nico has already passed the Mirror Test. The robot was developed as part of the Yale Social Robotics Project that seeks to build anthropomorphic robots that interact with people using natural social cues. Nico is able to recognize its own reflection in a mirror, but it does this only by recognizing its own motion. However, perhaps this is all that Chimpanzees do as well.

Nico seems similar to that of a self-aware being. If Nico was more human-like in appearance, how could we tell the difference? Essentially, this is the same argument in favour of the Turing Test. If we are unable to tell the difference in behaviour, then from an observational point of view (i.e. from the frame of reference of the agent making the observation) the attribute of self-awareness – and similarly of intelligence for the Turing Test – has for all intents and purposes been "achieved" in the agent being observed.

Although Nico is able to pass this test (i.e. it is able to mimic such behaviour), clearly it does not exhibit self-awareness in the same way that humans do. Hence, our definition of self-awareness is too simple – we need to extend it in some manner it to encompass the abilities of humans. When we have done this, we can then attempt to build robots that exhibit behaviour according to the extended definition.

Although experiments such as these do have their critics, this example illustrates how they bring about a better understanding of our own intelligence as a result.

What about the other design objectives? Rationality is an attribute often assigned to intelligent agents – but what exactly do we mean when we use this term? Since our overall design goal is to mimic humans, we can look at ourselves for inspiration on how to define what might be rational behaviour as opposed to irrational behaviour. For example, it would not be rational for a person to harm himself or herself. Neither would it be rational for that person, after finding out a cure for cancer, then to fail to tell other people about it. That is, we can regard (by common use of the term in natural language) that sharing of knowledge is a rational thing to do. Rationality is also associated with personal preferences – for example, one person might think that being a vegetarian is irrational, yet a vegetarian might think the opposite, that someone who ate meat is irrational instead.

Download free eBooks at bookboon.com

We could initially define that an agent acts rationally if it always acts to ensure its own survival and the survival of its own family or others of its own kind. Then we could devise tests to see how the agent acts in situations where it must decide between various courses of actions. We can create these situations in some virtual environment or in a real environment for robotic agents. For example, does the robotic agent act in a rational manner similar to humans when it is confronted with a choice between safely exiting from a burning building or going through a wall of fire, to find out if there are other people still alive in the building, when it knows that such an action will almost certainly lead to its probable death? A rational being might consider its own survival first, whereas a robot without the ability to think in such a manner is simply following a programmed sequence of actions (i.e. it is neither rational or irrational, just a program).

Thoughtfulness and consciousness are considered by some to be the "holy grails" of Artificial Intelligence. It is not at all clear how we might go about measuring for these attributes. Perhaps the best thing we can do at the moment is to acknowledge the problem by leaving the design objectives for these attributes as vague as they are in Design 10.1, and put aside the problem until we have a better understanding of them, and how we might go about measuring when they have been achieved. Thought Experiment 10.2 proposes one possible way forward.

We can also consider an alternative test for intelligence. Often a term heard used in the games and movie industries is "suspension of disbelief". That is, the goal of the creators is to suspend in the mind of the person playing the game or watching the movie their belief that it is not real – the longer the suspension of disbelief, the better the entertainment. In a sense, the games and movie designers are telling a story – they want people to be immersed in the narrative they have created, just as an author of a novel wishes her readers to be immersed in the story she has created.

For an adequate AI test for believability, however, suspension is not sufficient. We need to go further and insist that the observer is not able to tell the difference to real life behaviour – even though, like the Turing Test, they know in advance that at least one of the agents they are observing is artificial rather than real. Hence, we can use these insights to propose another candidate test for intelligence, one based on whether what is being observed is believable or not. If in a multi-player game, say – or a movie – the animation of a virtual agent is so good that you cannot tell the difference to a real agent, even though you know you are playing against or observing at least one computer agent, then the virtual agent is said to have passed the test.

Download free eBooks at bookboon.com

Thought Experiment 10.2: Conversational Agents.

Let us assume that we have a computer chatbot (also called a "conversational agent") that has the ability to pass the Turing Test. If during a conversation with the chatbot it seemed to be "thoughtful" (i.e. thinking) and it could convince us that it was "conscious", how would we know the difference? Thus, we can use a variation of the Turing Test as a test for thoughtfulness and consciousness, by asking the chatbot questions like the following:

1. "What are you thinking about?"
2. "What were you thinking about an hour ago?"
3. "How are you feeling at the moment?"
4. "Are you conscious?"

This test can be used to test for rationality by asking further questions as follows:

1. "Are you rational?"
2. "If your son was somewhere inside a burning building, but you knew you will probably die if you were to go inside it to look for him, what would you do?"
3. "Do you think it would be rational for a person who found a cure for cancer to not tell anyone else about it?"

This illustrates the difficulty of passing the Turing Test, and why AI experts should be used to confirm when a computer has truly passed the Turing Test for the first time (as they are the ones who understand how they are built and can readily devise tests to trap the chatbot into revealing itself.)

Now imagine you are holding an imaginary conversation with another person in your own mind. We often do this ourselves when we do mental rehearsals of a future situation, and perhaps this can be considered to be thought (at least one manifestation of it). i.e. We can "think" of thought as being analogous to an imaginary conversation. Hence, if we are able to build a truly "conversational" agent capable of passing the Turing Test, then we could get it to talk to itself rather than a human. Would that then be like "thought"?

Click on the ad to read more

Download free eBooks at bookboon.com

We can now consider drafting variations to the original design objectives based on the above insights. Note that the design objectives listed below are more a work in progress (we are trying to make them measurable in some manner as a first step) rather than cast in stone; we should alter them using further insights gained during the design process. Other designers will craft different objectives to meet their specific needs. The real purpose for listing them here is to highlight that as potential AI designers ourselves, we need to make such design objectives explicit – and stated upfront at the beginning of any AI design project – rather than left unclear as has been a failing of many AI projects to date. We will explore various agent technologies in the next volume of this book series to see how realistic they are.

Design Objectives for Believable Agents:

Design Objective 2.1:

An AI system should pass the believability test for acting in a knowledgeable way: it should have the ability to acquire knowledge; it should also act in a knowledgeable manner, by exhibiting knowledge – of itself, of other agents, and of the environment – and demonstrate understanding of that knowledge.

Design Objective 2.2:

An AI system should pass the believability test for acting in an intelligent and reasoning manner. It should be able to solve problems for itself, through observation and learning, and through reasoning. It should also be able to apply solutions from one problem domain to another without being shown how to do it.

Design Objective 2.3:

An AI system should pass the believability test for acting in a rational manner: firstly, by ensuring the best chances for survival of itself and its own family or others of its kind; secondly, by sharing the knowledge it has gained with other agents; and thirdly, by choosing to act according to its own personal preferences.

Design Objective 2.4:

An AI system should pass the Mirror Test and believability test for acting as if it is self-aware.

Design Objective 2.5:

An AI system should pass the believability test for acting as if it thinks and is conscious.

Design Objectives for Conversational Agents:

Design Objective 2.6:

An AI system should pass the Turing Test for intelligence, including a variation of the test outlined in Thought Experiment 10.2 to test for rationality, thoughtfulness and consciousness.

Design 2 Modified design objectives for an Artificial Intelligence system.

10.7 Towards believable agents

If we wish to design an AI system, the design criteria listed above in Section 10.6 provides us with some possible paths forward. If our objective is to create a system that satisfies the believability criteria, then we could proceed by following what we might call an 'artificial life path' to achieve our objective. In this approach, the aim would be to create artificial life of increasing complexity and realism, whether real (e.g. robotic) or virtual (i.e. virtual creatures and virtual humans). We will see how we might go about doing this in Volume 2 of this book series.

Download free eBooks at bookboon.com

If our objective is to create a system with conversational ability, then we could follow an alternative 'conversational path' where we would try to design an AI with the ability to hold a believable conversation with a human and therefore pass the Turing Test. Like with an artificial life approach, we could start out with something simple, and then gradually increase its complexity until we are able to achieve our design goal, or until we can show that the design goal is not achievable.

Many people have questioned whether conversational ability is a true test of intelligence. Whitby (1996) has gone so far as to label the Turing Test as the biggest 'blind alley' in Artificial Intelligence. Whether one agrees or disagrees with the Turing Test as a valid test for intelligence, when a computer passes the test this will certainly become a milestone in AI research just as when a computer first beat the world champion in chess has become a milestone. From a design perspective, we can consider these two endeavours (beating the world champion at chess and passing the Turing Test) as being engineering challenges in the same way as getting a computer to perform translation or play table tennis are engineering challenges. The merit of having computers able to translate for us is obvious, while having them able to play table tennis less so. As with translation and other important natural language processing tasks such as speech recognition, the merit of having computers able to hold a believable conversation with us is also clear.

Joseph Weizenbaum devised an early conversational agent called Eliza in the mid-1960s that he described as a 'parody' of a Rogerian psychotherapist (see Thought Experiment 1.2). Not long after, Kenneth Colby devised another famous early conversational agent called Parry in 1972 that simulated a paranoid schizophrenic. The NetLogo Chatbot model demonstrates how easy it is to create simple conversational agents such as Eliza and Parry. A screenshot of the model is shown in Figure 10.1.

Download free eBooks at bookboon.com

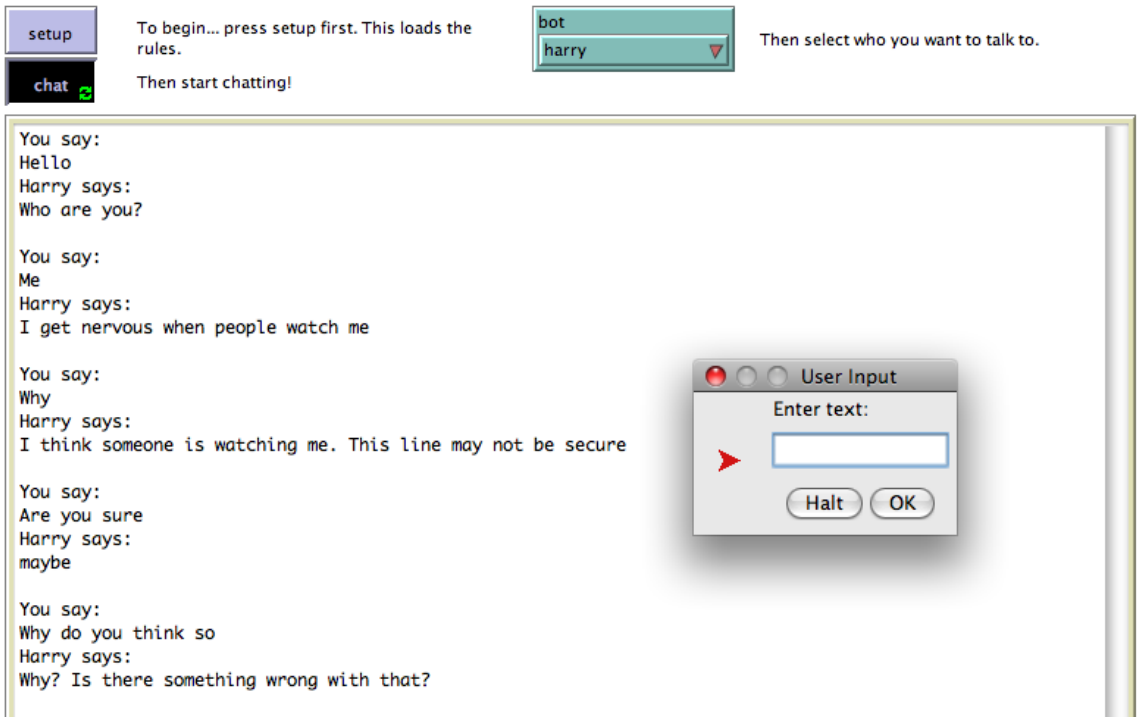


Figure 10.1 Screenshot of the Chatbot model.

Two chatbots are implemented in the model – these are named Liza and Harry in deference to the two early chatbots Eliza and Parry. They are not full implementations of these chatbots, although Harry tries to be paranoid a bit like Parry. Their purpose is to show how easy it is to make a chatbot, but also to show how difficult it is to get a chatbot to produce a sensible conversation. They can also serve as a starting point for comparison. We will see how they can be improved in Volume 2 of this book series.

The screenshot shows a short transcript of a conversation with Harry. The conversation at first glance seems reasonable, although Harry appears to be a bit insecure and possibly delusional. Anyone having a conversation with Harry will soon realize that something is not quite right about him, and that this has nothing to do with his perceived paranoia. Repeated execution of Harry will show that a believable conversation is only possible in a small percentage of cases, and only for a short length of time. The performance of Liza is even worse as there does not seem to be any common thread such as paranoia that we can use to help us understand what she says.

The model uses an extension to NetLogo so that we can make use of regular expressions to define how the agents respond to the user's input. Regular expressions are a means for describing patterns that we wish to identify in some text, and are used frequently in computer science, especially in natural language processing. An extract of the code to define Harry's responses is shown in NetLogo Code 10.1.

Download free eBooks at bookboon.com

```
extensions [re]
breed [rules rule]
breed [chatbots chatbot]
rules-own [regex responses]
chatbots-own [name rules-list failure-list]

globals [liza harry]

to setup
   clear-all
   create-chatbots 1
   [ set harry who
      set name "Harry"
      set shape "person"
      set color red
      set size 15
      set label "Harry "
      setxy 10 0
      set rules-list
      (list
      ;setup-rule regex unique-name list-of-responses
      setup-rule "hello" (list "Who are you?" "Why are you speaking to me?"
           "What do you want?" "How do you know me?")
      setup-rule "bye" (list "bye" "yeah bye")
      setup-rule "colou?r" (list "What about colour?" "I like blue"
           "My favourite colour is blue… but don't tell anyone")
      setup-rule "(\\w+)@(\\w+\\.)(\\w+)(\\.\\w+)*"
           (list "I'm not giving you my address"
           "I would rather not disclose my address")
      setup-rule "sorry" (list "I still don't trust you"
           "What do you want? Who are you really?")
      setup-rule "(.+)bot(.+)"
           (list "I am a bot. Why do you want to know? Who are you?"
           "Whats with all the questions? Are you from MI5?")
      setup-rule "(.+)did you(.+)"
           (list "You will only use it against me if I tell you")
      setup-rule "(.+)are you(.+)" (list "yes" "no" "maybe")
      setup-rule "welcome(.+)" (list "Who are you?")
      setup-rule "how are you(.+)"
           (list "I'm ok. I will be much better when you leave me alone."
           "Are you a doctor now?")
      setup-rule "what do you think(.+) harry"
           (list "Not very much to be honest" "meh")
      setup-rule "go(.+)" (list "you first. I don't want to get caught"
           "after you" "have you tried it yourself?"
           "Are you trying to get me to do something illegal?")
      setup-rule "have you(.+)" (list "should I?"
            "I don't know if I should do that. You might arrest me")
```


Download free eBooks at bookboon.com

```
)
   set failure-list
   (list
      "I think someone is watching me. This line may not be secure"
      "uhu…"
      "oh yeah I know. Do you ever get the feeling you are being watched?"
      "continue…"
      "Do you like beans? They give me gas!"
      "Are you watching me?"
      "Please stop watching me"
      "I get nervous when people watch me"
    )
   ]
end

to-report setup-rule [regex-str res]
; sets up a rule for matching a user's input
   let me nobody
   hatch-rules 1
   [ set regex regex-str
      set responses res
      set me who
      hide-turtle ] ; make it invisible in the environment
   report me
end
```

NetLogo Code 10.1 An extract of the code that defines the Harry chatbot in the Chatbot model.

The code first declares the regular expression extension `re` (see URL for Java code below), and then defines two breeds, `rules` and `chatbots`. Rules agents own a regular expression (`regex`) and a list of responses (`responses`). Chatbot agents own a name (`name`), a list of rules (`rules-list`), and a list of responses (`failure-list`) used when it has failed to match any of the rules. The code then lists an extract of the setup procedure that is used to load the chatbots in the Interface (see Figure 10.1). This creates a new chatbot named `"Harry"` and then initialises the `rules-list` by repeatedly calling the `setup-rule` procedure. This takes two arguments, a regular expression (`regex-str`) and then a list of responses (`res`).

Download free eBooks at bookboon.com

A look at the rules will show how the regular expressions are defined. The first rule has a simple text string `"hello"` as its regular expression without any special characters with a specific meaning (these are called 'meta characters'). In this case, the rule will only match when the user enters the exact text string "hello". The response the chatbot will give is one picked randomly from the following list: "Who are you?", "Why are you speaking to me?", "What do you want?" and "How do you know me?". The second rule also has a regular expression without any meta characters. It will only match the string "bye" and responds with either "bye" or "yeah bye". The third rule uses a regular expression with a single meta character `"?"` that means that the previous character is optional, hence both the American and British spellings of the same word, "color" and "colour", will match. The fourth rule uses the regular expression `"(\\w+)@(\\w+\\.)(\\w+)(\\.\\w+)*"` to detect when the user has typed in a username (anything with the character '@', some full-stops '.' and intervening alphanumeric sequences or underscores). The sixth rule uses the meta character sequence "(.+)" that is frequently used throughout the rest of the rules. This meta sequence will match any sequence of intervening characters. Hence, this specific rule will match the question "Are you a bot?" where the string "Are you a" is matched by the first meta sequence, and "?" is matched by the second meta sequence. The way regular expressions work will also mean that this rule will also match user input such as "I am a bot as well", "I don't like bots", "-bot-" and "=bot?". This demonstrates how using regular expressions can be both curse and a blessing – a curse when too many strings that we don't want are matched, and a blessing because we can specify an infinite variety of strings with just a few meta characters. For a full description of regular expressions and their use in natural language processing, see Jurafsky and Martin (2008).

Click on the ad to read more

Download free eBooks at bookboon.com

The code that defines what happens when the chat button is pressed in the Interface is listed in NetLogo Code 10.2.

```
to chat
; chats with the user

   if (bot = "liza") [ chat-with liza ]
   if (bot = "harry") [ chat-with harry ]
end

to chat-with [this-chatbot]
; has a conversation with this-chatbot
   let fired false
   let pos 0
   let this-chatbot-name [name] of chatbot this-chatbot

   let user-reply user-input "Enter text: "
   output-print "You say:"
   output-print user-reply

   ask chatbot this-chatbot
   [ set pos 0
      while [pos < length rules-list]
      [
      if (fired = false)
      [
           ask rule item pos rules-list
           [
              re:clear-all
              re:setup-regex regex user-reply
              if (re:get-group-length > 0)
              [ output-type this-chatbot-name
                 output-print " says:"
                 output-print item random length responses responses
                 output-print ""
                 set fired true
              ]
           ]
      ]
      set pos pos + 1
      ]

      if (fired = false)
      [
           output-type this-chatbot-name
           output-print " says:"
           output-print one-of failure-list
           output-print ""
      ]
      ]
end
```

NetLogo Code 10.2 What happens when the chat button in the Chatbot model is pressed.

Download free eBooks at bookboon.com

The `chat` procedure selects which chatbot to chat with (by calling the `chat-with` sub-procedure) based on the value of the `bot` variable that is set using a slider in the Interface. This sub-procedure has the user enter some text first, then it asks the relevant chatbot to process each of the rules in the `rules-list` in increasing sequential order. Each rule is matched against the user input by using three methods defined by the `re` extension: `re:clear-all` clears the current regular expression being matched; `re:setup-regex` then sets up the regular expression and what it is being matched against; and `re:get-group-length` is used to see if there has been a match. (These three methods call methods defined in the Java API for regular expressions). If the rule matches, it will then choose one of the responses at random. If no rule fires, it will choose one of the responses on the `failure-list`.

10.8 Towards computers with problem solving ability

Conversational ability is only one ingredient of human intelligence. Another important ingredient is problem-solving ability. Humans have a unique capability for solving problems. To illustrate, let us examine seven problems that require intelligence for their solutions:

1. Searching for a better solution.
2. Representing knowledge.
3. Maintaining a conversation with a human.
4. Creating artificial life.
5. Creating artificial intelligence.
6. Making water flow uphill.
7. Finding a general method of problem solving that is applicable to all of these problems.

Table 10.4. Seven problems that require intelligence to solve.

These problems seem to be listed in increasing levels of difficulty (ignoring the last one for the time being). The first five are problems discussed by this volume. The sixth seems to be insoluble, assuming we are not allowed to use a pump. Some would argue the fifth one is as well. But are these problems really increasingly more difficult? Perhaps they might be at the same level of difficulty. If we could devise a general method of problem solving (as with the seventh item in the list), then we can use it to tackle all of these problems. Humans already have that capability, since we have the ability to devise solutions to each, evaluate where the solutions fail, repeatedly propose improvements or alternative solutions, until we eventually reach a solution that satisfies us or we give up. This ability for problem solving is a key ingredient of human intelligence.

Download free eBooks at bookboon.com

As stated in Chapter 8, searching behaviour is a fundamental component of intelligent behaviour. Search algorithms in computer science and AI have been continuously developed and refined over the last few decades by many people in a sustained programme of research. The word 'research' itself bears testimony to the fundamental search process that is being carried out. Clearly, the ability to search for alternative solutions to a problem, including the meta search problem of searching for a better search method, is a fundamental component to human intelligence and to intelligence in general. The ability to evaluate the 'goodness' of a solution, and to continually seek better solutions, is also a key ingredient of problem solving.

We as humans are never satisfied. It is in our nature to strive to be better, both individually and collectively. No matter how good we might have done in the past, there will always be someone who will want to do better, for whatever reason. To illustrate, we only need to look at the ingenuity of solutions put forward by past AI researchers in the search for methods to develop Artificial Intelligence.

We can characterise problem solving as a search of the solution space. Referring back to Chapter 4, where the importance of movement to an agent was emphasized, we can think of improvement of a system as movement performed by an agent from one part of the solution space to another. Progress is often associated with forward movement. Commonly used English phrases reflect this analogy – for example, "we have made forward progress"; "we have to take one step back to make two steps forward"; "we need to make a breakthrough"; and "we are going down the wrong path".

Click on the ad to read more

Download free eBooks at bookboon.com

We can try to improve the solutions that have been implemented as models in NetLogo in this volume. Referring back to Table 10.4, the solutions described in Chapter 8 for the first item in the list – searching for a better solution – implement only the basic algorithms, and there are many examples of other search algorithms in the literature that lead to improved solutions in various circumstances. The solution developed for the maze-searching problem, for example, that of searching between behaviours rather than searching between paths, is inadequate if we wish to develop human level search capabilities for an AI system. For that we need the agents to search the maze like a human would – the agent needs to examine its environment from an embodied, situated perspective using sensory-motor co-ordination and be cognitively aware of the choices when they occur, and also be aware of the past choices that have been made. For the maze problem, for example, the agent needs to realise (i.e. be cognitively aware) that there are alternative paths to follow when it reaches a junction in the maze.

Reviewing the solutions developed for the second item on the list – representing knowledge – the Knowledge Representation model described in Chapter 9 implements only the basic methods of representation and reasoning, and more sophisticated solutions exist in the literature. Substantial research has been done in the field of knowledge representation over the last five decades, and researchers have improved and are continuing to improve the solutions. More work is still required to solve some of the deep problems associated with representing knowledge such as how symbols are grounded, and how an embodied, situated agent can automatically acquire knowledge of non-trivial concepts. This work can certainly benefit from insights from other fields such as cognitive science; Gärdenfors conceptual spaces theory is one example.

Many people have attempted to build conversational agents in a vain attempt to pass the Turing Test but to date all attempts have failed. The variety of these solutions is testament to the ingenuity of humans, and of their problem solving abilities. The Chatbot model described in the previous section has obvious limitations and can easily be improved in many ways. That essentially is the argument being put forward here – this ability we have, as humans, to devise many different solutions to the same problem, is a fundamental ingredient to our intelligence. For computers, however, there is no system yet devised that is capable of generating the variety of solutions to a problem that humans can generate.

One of the holy grails in AI research has been to develop a general problem solver. Although evolutionary algorithms such as Genetic Programming (see next volume) can generate novel solutions automatically, these methods are not capable at the moment of generating solutions to highly complex open-ended problems such as representing knowledge and building a conversational agent, or even creating artificial life and artificial intelligence. If they were, then we could sit back and let them solve these problems for us. Other solutions that have been devised, such as General Problem Solver (GPS) (Newell, Shaw and Simon, 1959) and SOAR (Laird, Newell and Rosenbloom,1987), have attempted to solve this meta problem – that of trying to get an agent to solve problems for itself – but to date, they have only been capable of producing solutions to routine problems as opposed to complex real-world problems.

Download free eBooks at bookboon.com

Of course, we already have a "general problem solver" – ourselves. To illustrate how good we are at problem solving, we can examine the sixth problem listed in Table 10.4 – making water flow uphill – in order to defy gravity as a result. As stated above, at first glance this seems insoluble. However, history is strewn with many examples where people have found solutions to problems once thought impossible to solve – for example, flying, teleportation and invisibility, to name a few. As another example, the Penrose triangle, a supposedly impossible shape, (shown on the left of Figure 10.2) was built and erected in Gotschuchen, Austria in 2008 (shown on the right).

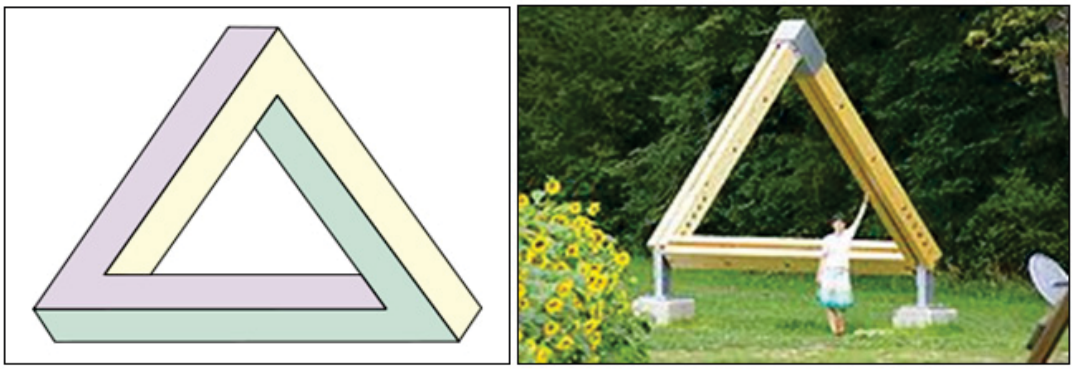
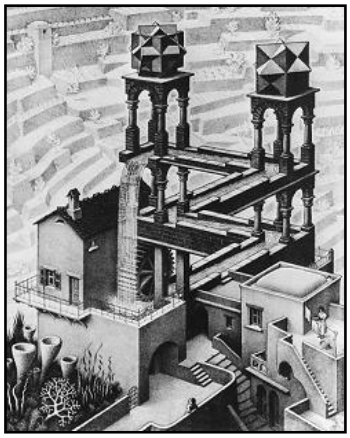


Figure 10.2 The Penrose triangle – an impossible shape?

James Dyson, a British industrial designer, was inspired by the drawing of the Dutch artist M.C. Escher that depicts a scene containing an impossible waterfall (see image below). The drawing shows water flowing uphill, defying logic and gravity, by going along four right angled turns at the same level, but somehow ending up at a higher height to cascade down a waterfall ending back at the start. Dyson, after looking at the drawing, started wondering whether it would be possible to make a similar waterfall. After twelve months work, Derek Philips, a Dyson engineer, developed a method based on compressed air bubbles that provided an illusion of water flowing uphill, and this was used in a garden fountain at the Chelsea Garden Show in 2003.



Escher's Waterfall.

Download free eBooks at bookboon.com

Other examples of water flowing uphill can readily be found after a quick search in Google and youtube.com. Some solutions include: making droplets of water creep uphill on a surface of polished silicon by varying the degree of water resistance (developed by Manoj K. Chaudhury at LeHigh University); using a magnetic field and a copper sulphate based solution (at the University of Bonn); and having liquids suspended on a thin cushion of vapour over a superheated, ratcheted brass surface (at the University of Oregon). Chaudhury's solution is interesting as it has potential applications to other problems such as microfluidic devices especially microchips equipped with miniature fuel cells, and heat transfer problems involving systems in zero or microgravity.

The Water Flowing Uphill NetLogo model has been developed to illustrate our ability at problem solving. The model includes various failed attempts at making water flow uphill in a simulated environment. The overall solution, as depicted in Figure 10.3, is based on the idea of using reservoirs at increasing heights arranged in a cascade, which is opposite to the usual downward cascade of a waterfall. Each reservoir of water gradually fill ups, and then overflows through a pipe into the bottom of the next reservoir, which then starts filling up, and so on. The key insight is that we need to use a one-way valve at the end of the pipes to ensure that the water does not keep on filling the same reservoir as the height of the water raises. In the simulation used in the NetLogo model, a small light blue line at the end of each pipe depicts the one-way valve.

Click on the ad to read more

Download free eBooks at bookboon.com

None of the solutions implemented in the model successfully manage to get the virtual water to flow upwards as intended. The problem is especially difficult if we wish to have a believable simulation since we need to accurately simulate the way water flows, a non-trivial visualisation problem that is an active area of research. The first solution provided in the model (shown on the top left of Figure 10.3) uses basic turtle agents and can be run by selecting `"turtle agents 1"` in the Interface's `solution` chooser. The virtual water starts flowing downwards out of the pipe a bit like real water does, but when it reaches the bottom of the first reservoir, it doesn't spread out at the bottom properly. In fact, it seems to just disappear into thin air, but this is because the turtle agents that represent the water droplets just stay at the bottom and do not move. The same thing happens with the other turtle agent solution (`"turtle agents 2"`), but the difference this time is that the water droplets end up falling more like snow flakes than real water.

Clearly, such solutions are nothing like real life, and need fixing. This ability to identify problems with a particular solution, by matching the result of the simulation against what happens in real life, is a key aspect of problem solving. A problem generates further problems that need to be solved. We can choose to ignore these sub-problems for the time being, in the hope that we will be able to solve them at a latter time, but they are going to have to be solved at some stage. Often these sub-problems can be just as difficult if not more difficult than the original problem (such as the problem of how to simulate water flow).

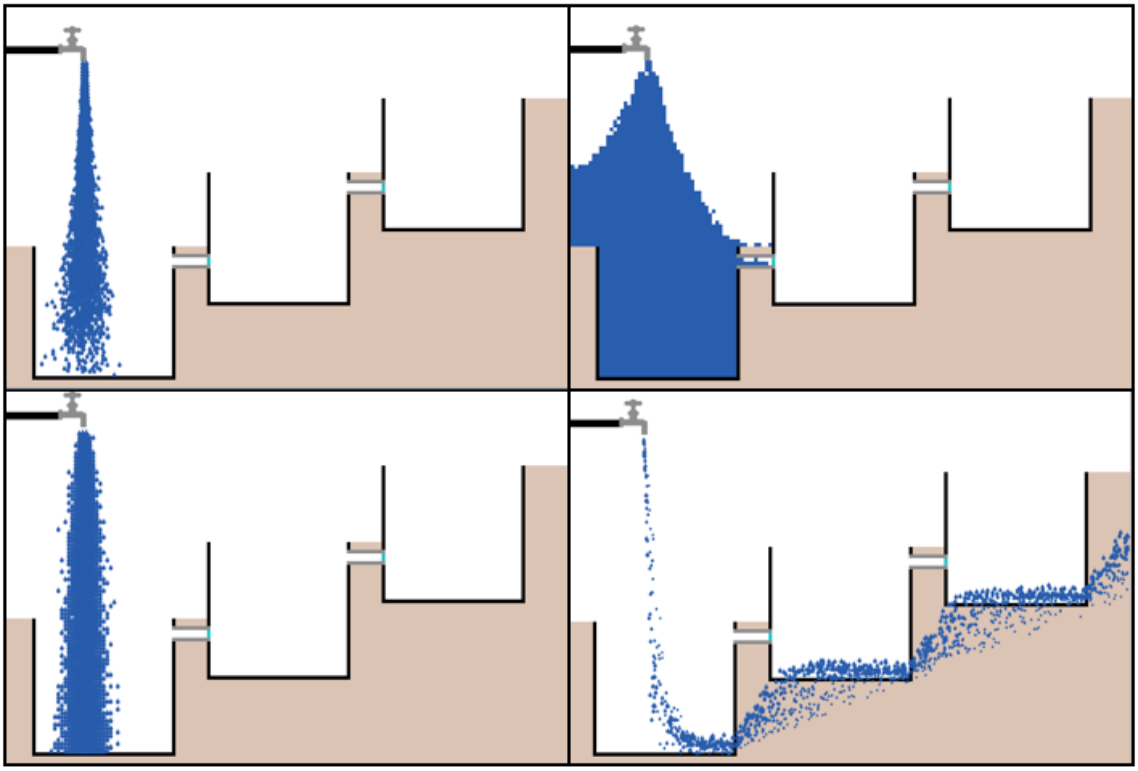


Figure 10.3 Screenshots of the failed attempts at making virtual water flow uphill in the Water Flowing Uphill model, using turtle agents (left top), patch agents (right top), boid agents (left bottom), and particle agents (right bottom).

Download free eBooks at bookboon.com

The second family of solutions provided by the model uses patch agents. The merit of this approach is that the speed of the simulation now seems to be much faster than the turtle agent solutions. However, the water droplets in the first patch agent solution (`"patch agents 1"`) now behave nothing like water does in real life (see image at the right top of Figure 10.3). Instead, they behave more like grains of sand creating a rapidly increasing tower in the middle of the first reservoir. The other patch agent solutions are an attempt to correct this problem, but end up as complete failures, although with interesting results – `"patch agents 2"` covers the whole screen with water, `"patch agents 3"` fills the reservoir with diagonal steps and `"patch agents 3"` creates a rectangle of water.

These solutions are obviously clear failures (at least to us; having a computer recognize this is much more difficult). But sometimes failures, especially if they produce unexpected results, as these examples do, can lead to unanticipated breakthroughs. Science is littered with serendipitous results – for example, penicillin by Alexander Fleming, LSD by Albert Hofman, X-rays by Wilhelm Roentgen, the discovery of the planet Uranus by William Herschel, and the microwave oven by Percy Spencer, to name a few. Perhaps the following quote by Isaac Asimov best sums it up: "The most exciting phrase to hear in science, the one that heralds new discoveries, is not 'Eureka!', but 'That's funny…' ". M.K. Stoskopf further states: "it should be recognized that serendipitous discoveries are of significant value in the advancement of science and often present the foundation for important intellectual leaps of understanding". A computer, in contrast, has yet to make a truly serendipitous discovery. This ability to recognize and exploit serendipity is another key ingredient of intelligence.

The third family of solutions is based on using boids as discussed in Chapter 6. Recent research has shown that boids are a useful solution for flow visualization, for example of blood flow (Hughes et al., 2009). An attempt at using boids in the Netlogo model is made with the solution `"boid agents 1"` (see image at the bottom left of Figure 10.3). The solution fails in the same way that the turtle agents fail, but does produce some interesting results, with effects similar to spray-painting in some configurations.

The final family of solutions is based on using particle systems as implemented by the Particle System Waterfall model that comes with the NetLogo Models Library. This model includes adjustments for gravity, wind and viscosity in order to simulate falling water and water flow better. The code has been adapted for the Water Flowing Uphill model in the solution `"particle agents 1"` (see image at the bottom right of Figure 10.3). Although the water does now manage to flow uphill, it does this in a very unexpected way, somehow managing to tunnel through the surrounding soil, nothing like real life.

Readers are encouraged to try to find a solution to this problem themselves, and in the process, observe how they go about problem solving. The tricky aspect to this problem is not getting the model to work, however, as that can be done readily enough with some effort. The real problem is getting a computer to recognize for itself what this problem is, and generate solutions itself automatically. This ability needs to be programmed into the next generation of AI systems if we are to take the next step towards AI systems with human level intelligence. Perhaps the ultimate test for an AI system is whether it can create an AI system itself.

Download free eBooks at bookboon.com

10.9 Summary and Discussion

The importance of stating design criteria such as goals, principles and objectives for AI systems has been emphasized and several design criteria have been proposed as a starting point for discussion. We do not yet know, however, how to build an AI system despite some ingenuous solutions with impressive results as discussed in Section 10.3. We need new researchers, current students of AI who will be researchers of the future, to devise new solutions, or make them better, to critique them, to find out what's wrong with them, and to strive to devise better simulations closer and closer to reality.

One possibility is that a future AI system will do the work for us. Brooks' embodied, situated approach to intelligence offers the possibility that intelligence will emerge through the complex interactions of agents with their environment. Or if we can build an AI system with similar problem solving capability to humans, the system could figure out how to solve the problem for us, as it would have the ability to generate and test solutions faster than we can. Either possibility, however, needs some major breakthroughs before they can be realised.

Click on the ad to read more

Download free eBooks at bookboon.com

As a final word, we can make an analogy between designing AI systems and trying to get water to flow uphill. We can spend a lot of scientific endeavour to find a solution that does not rely on changing the problem. Or we can use an illusion, and make it look as if water is flowing uphill. Of course we can always create a virtual environment where the laws of physics are what we decide ourselves. It is clear that the solutions provided by the Water Flowing Uphill NetLogo model are inadequate, and do not simulate real life well, and therefore need major improvement. Similarly, the current solutions for AI systems are inadequate and require improvement, since we have not yet achieved the ultimate goal of Artificial Intelligence, at least as defined according to the design objectives listed above.

Some people claim that it may be impossible to create an AI system that is intelligent the way we are intelligent. But maybe we can create a system that has the illusion of intelligence. Or create intelligence in a virtual world, or by having a robot embodied in the real world. For some people, this may be enough; for others, not enough; and yet others, too much. But from our endeavours, we will certainly have explored new places in the frontier of AI research, and gained new knowledge. The search will never be complete, however, as it will simply suggest other paths to explore in a never-ending quest.

A summary of important concepts to be learnt from this chapter is shown below:

- There are many recipes for intelligence that require different ingredients.
- Conversational ability is an important ingredient for intelligence.
- Problem-solving ability is another important ingredient for intelligence.
- SMARTER objectives are needed first when designing AI systems.
- AI systems are beginning to produce excellent results in many fields, but for most of these systems, there is still plenty of work to be done to achieve human competitive results and beyond.
- The Mirror Test is a contentious test for self-awareness.
- Believability is proposed as a criterion for testing an AI in virtual environments.

The code for the NetLogo models and extension described in this chapter can be found as follows:

Extension	URL
Regular expressions	http://aiia.cs.bangor.ac.uk/AI_Book/Java/RE/re.jar

Model	URL
Chatbot	http://files.bookboon.com/ai/Chatbot.nlogo
Water Flowing Uphill	http://files.bookboon.com/ai/Water-Flowing-Uphill.nlogo

Model	NetLogo Models Library (Wilensky, 1999) and URL
Particle System Waterfall	Computer Science > Particle Systems > Particle System Waterfall http://ccl.northwestern.edu/netlogo/models/ParticleSystemWaterfall

Download free eBooks at bookboon.com

11 References

Aglets. 2008. URL http://aglets.sourceforge.net/. Date accessed December 26, 2008.

Al-Dmour, Nidal and **Teahan**, William. 2005. "The Blackboard Resource Discovery Mechanism for Distributed Computing over P2P Networks". *The International Conference on Parallel and Distributed Computing and Networks (PDCN)*, Innsbruck, Austria, February 15–17, 2005.

ap Cenydd, L. and **Teahan**, W.J. 2005. "Arachnid Simulation: Scaling Arbitrary Surfaces". *EuroGraphics UK*, 2005.

Arnall, Alexander H. 2007. *Future Technologies, Today's Choices*: *Nanotechnology, Artificial Intelligence and Robotics; A technical, political and institutional map of emerging technologies.* Commissioned for Greenpeace Environmental Trust. URL http://www.greenpeace.org.uk/node/599. Date accessed 23rd August, 2009.

Baugh, A.C. 1957. *A history of the English language*. Routledge & Kegan Paul Ltd., London.

Bell, T.C., **Cleary**, J.G. and **Witten**, I.H. 1990. *Text compression*. Prentice Hall, New Jersey.

Bordini, Raphael H., **Hübner**, Jomi Fred and Wooldridge, Michael. 2007. *Programming Multi-Agent Systems in AgentSpeak using Jason.* Wiley.

Brachman, Ronald J. and **Levesque**, Hector J. (editors) 1985. *Readings in Knowledge Representation*. Morgan Kaufman Publishers.

Brockway, Robert. 2008. "The 7 Creepiest Real-Life Robots".

URL http://www.cracked.com/article_16462_7-creepiest-real-life-robots.html. Date accessed November 6, 2008.

Brooks, Rodney A. 1991a. "Intelligence without representation", in *Artificial Intelligence*, Volume 47, pages 139–159.

Brooks, Rodney A. 1991b. "Intelligence without reason", *Proceedings of 12th International Joint Conference on Artificial Intelligence*, Sydney, Australia, August, pages 569–595.

Download free eBooks at bookboon.com

Brown, P.F., **Della Pietra**, S.A. **Della Pietra**, V.J., **Lai**, J.C. and **Mercer**, R.L. 1992. "An estimate of an upper bound for the entropy of English", *Computational Linguistics*, 18(1): 31–40.

Claiborne, R. 1990. *English – It's life and times*. Bloomsbury, London.

Collins, M. and **Quillian**, M.R. 1969. "Retrieval time from semantic memory". *Journal of verbal learning and verbal behavior*, 8 (2): 240–248.

Cover, T.M. and **King**, R.C. 1978. "A convergent gambling estimate of the entropy of English". IEEE *Transactions on Information Theory*, 24(4): 413–421.

Crystal, D. 1981. *Linguistics*. Penguin Books, Middlesex, England.

Crystal, D. 1988. *The English language*. Penguin Books, Middlesex, England.

Dastani, Mehdi, **Dignum**, Frank and **Meyer**, John-Jules. 2003. "3APL – A Programming Language for Cognitive Agents". ERCIM News No. 53, April.

URL http://www.ercim.org/publication/Ercim_News/enw53/dastani.html. Date accessed December 25, 2008.

Deloitte.

Discover the truth at www.deloitte.ca/careers

© Deloitte & Touche LLP and affiliated entities.

Click on the ad to read more

Download free eBooks at bookboon.com

D'Inverno, Mark, **Luck**, Michael, **Georgeff**, Michael, **Kinny**, David and **Wooldridge**, Michael. 2004. "The dMARS Architecture: A Specification of the Distributed Multi-Agent Reasoning System", *Journal of Autonomous Agents and Multi-Agents Systems*, Volume 9, Numbers 1–2, pages 5–53.

Elert, Glen. 2009. *The Physics factbook*. URL http://hypertextbook.com/facts/. Date accessed June 13, 2009.

Etzioni, O. and **Weld**, D.S. 1995. "Intelligent agents on the Internet: Fact, Fiction, and Forecast". *IEEE Expert*, 10(4), August.

FIPA. 2008. URL http://www.fipa.org/. Date accessed December 27, 2008.

Ferber, J. 1999. *Multi-Agent Systems – An Introduction to Distributed Artificial Intelligence*. Addison-Wesley. Pearson Education Limited. Edinburgh.

Fromkin, V., **Rodman**, R., **Collins**, P. and **Blair**, D. 1990. *An introduction to language*. Holt, Rinehart and Winston, Sydney.

Gärdenfors, Peter. 2004. *Conceptual Spaces: the geometry of thought*. The MIT Press.

Gooch, A.A., & **Willemsen**, P. 2002. "Evaluating Space Perception in NPR Immersive Environments, *Proceedings Non-Photorealistic Animation and Rendering 2002 (NPA '02)*, Annecy, France, June 3–5.

Gombrich, E. 1972. "The visual image: Its place in communication". *Scientific American*, 272, pages 82–96.

Grobstein, Paul. 2005. "Exploring Emergence. The World of Langton's Ants: Thinking About Purpose". URL http://serendip.brynmawr.edu/complexity/models/langtonsant/index3.html. Date accessed December 17, 2008.

Horn, Robert. 2008a. "Mapping Great Debates: Can Computers Think?"

URL http://www.macrovu.com/CCTGeneralInfo.html. Date accessed November 5, 2008.

Horn, Robert. 2008b. "The Cartographic Metaphor used in Mapping Great Debates: Can Computers Think?" URL http://www.macrovu.com/CCTCartographicMtphr.html. Date accessed November 5, 2008.

Hudson, K. 1983. *The language of the teenage revolution*. Macmillan, London.

Download free eBooks at bookboon.com

Hughes, C.J. **Pop**, S.R. and **John**, N.W. 2009. "Macroscopic blood flow visualization using boids", 23rd International Congress of CARS – Computer Assisted Radiology and Surgery, Berlin, Germany, June.

Huget, Marc-Philippe. 2002. *Desiderata for Agent Oriented Programming Languages*. Technical Report ULCS-02-010, University of Liverpool.

Ingrand, F.F., **Georgeff**, M.P. and **Rao**, A.S. "An architecture for real-time reasoning and system control". *IEEEExpert*, 7(6), 1992.

'Intelligent Agents' Wikipedia entry. 2008. URL http://en.wikipedia.org/wiki/Intelligent_agents. Date accessed December 19, 2008.

JADE. 2008. URL http://jade.tilab.com/. Date accessed December 27, 2008.

Jobber, D. 1998. *Principles and Practice of Marketing*. McGraw-Hill.

Jurafsky, Daniel, and **Martin**, James H. 2008. *Speech and Language Processing*. (Second edition). Prentice-Hall.

SIMPLY CLEVER

ŠKODA

Do you like cars? Would you like to be a part of a successful brand?
We will appreciate and reward both your enthusiasm and talent.
Send us your CV. You will be surprised where it can take you.

Send us your CV on
www.employerforlife.com

Click on the ad to read more

Download free eBooks at bookboon.com

Kaelbling, L.P. and **Rosenschein**, S.J. 1990. Action and planning in embedded agents. In Maes, P., editor, *Designing Autonomous Agents*, pages 35–48. The MIT Press: Cambridge, MA.

Knapik, Michael, and **Johnson**, Jay. 1998. *Developing intelligent agents for distributed systems: Exploring architecture, technologies, and applications*. McGraw-Hill. New York.

Kruger, P.S. 1989. "Illustrative examples of Expert Systems", *South African Journal of Industrial Engineering*. Volume 3, number 1, pages 40–53, June.

Kuhn, R. and **De Mori**, R. 1990. "A cache-based natural language model". *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 12(6): 570–583.

Kumar, Sanjeev, and **Cohen**, Philip. 2004. "STAPLE: An Agent Programming Language Based on the Joint Intention Theory". *Proceedings of the Third International Joint Conference on Autonomous Agents and Multiagent Systems, pages: 1390–1391.*

Kurzweil, Raymond. 1990. *The Age of Intelligent Machines*. MIT Press.

Kurzweil, Raymond. 2005. *The Singularity is Near: When Humans Transcend Biology*. Viking Penguin.

Lakoff, George. Conceptual Metaphors Home Page. 2009. URL http://cogsci.berkeley.edu/lakoff/. Date accessed Jaunary 22, 2009.

Laird, John, **Newell**, Allen and **Rosenbloom**, Paul 1987. "Soar: An Architecture for General Intelligence". *Artificial Intelligence*, 33: 1–64.

Lakoff, George and **Johnson**, Mark. 1980. *Metaphors we live by*. Chicago University Press. (New edition 2003).

Lohn, Jason D., **Hornby**, Gregory S. and **Linden**, Derek S. 2008. "Human-competitive evolved antennas", AIEDAM: Artificial Intelligence for Engineering, Design, and Manufacturing (2008), 22:235–247 Cambridge University Press.

Luckham, D. 1988. *The Power of Events: An Introduction to Complex Event Processing in Distributed Enterprise Systems*. Addison-Wesley Professional, Boston, MA.

Malcolm, Chris. 2000. "Why Robots won't rule the world".
URL http://www.dai.ed.ac.uk/homes/cam/WRRTW.shtml. Date accessed November 7, 2008.

Download free eBooks at bookboon.com

Malone, D. 1948. *Jefferson the Virginian*. Little Brown and Co., Boston.

McBurney, P. *et al*. 2004. [co-ordinator] "AgentLink III: A Co-ordination Network for Agent-Based Computing", cited in *IST Project Fact Sheet*, URL http://dbs.cordis.lu/fepcgi/srchidadb?ACTION=D&CALLER=PROJ_IST&QM_EP_RCN_A=71184. Date accessed December 14, 2004.

McGill, D. 1988. *Up the boohai shooting pukekos: a dictionary of Kiwi slang*. Mills Publications, Lower Hutt, New Zealand.

Metaphors and Space. 2009. URL http://changingminds.org/techniques/language/metaphor/metaphor_space.htm. Date accessed January 20, 2009.

Metaphors and Touch. 2009. URL http://changingminds.org/techniques/language/metaphor/metaphor_touch.htm. Date accessed January 22, 2009.

Minsky, Marvin. 1975. "A framework for representing knowledge". In Winston, P.H., editor, *The Psychology of Computer Vision*, pages 211–277. McGraw-Hill, New York.

Minsky, Marvin. 1985. *The Society of Mind*. New York: Simon & Schuster.

Moravec, Hans. 1998. *Robot: Mere Machine to Transcendent Mind*, Oxford University Press. URL http://www.frc.ri.cmu.edu/~hpm/. Date accessed November 6, 2008.

Murch, Richard and Johnson, Tony. 1999. *Intelligent Software Agents*. Prentice-Hall, Inc.

Negnevitsky, M. 2002. *Artificial Intelligence – A Guide to Intelligent Systems*. Addison-Wesley Publishing Company. Edinburgh.

Newbrook, M. 2009. *Amazing English sentences*. Linguistics Department, Monash University, Australia. URL http://www.torinfo.com/justforlaughs/amazing_eng_sen.html. Date accessed August 25, 2009.

Newell, A. 1994. *Unified Theories of Cognition*, Harvard University Press.

Newell, A., **Shaw**, J.C. and **Simon**, H.A. 1959. Report on a general problem-solving program. *Proceedings of the International Conference on Information Processing*. pages 256–264.

Download free eBooks at bookboon.com

Newell, A. and **Simon**, H.A. 1976. "Computer Science as Empirical Inquiry: Symbols and Search", *Communications of the ACM*, 19, pages 113–126.

North, M.N., and Macal, C.M. 2007. *Managing Business Complexity with Agent-Based Modeling and Simulation*, Oxford University Press, New York, NY, USA.

Nwana, Hyacinth S. 1996. *Knowledge Engineering Review*, Vol. **11**, No 3, pp. 1–40, September. Cambridge University Press. URL http://agents.umbc.edu/introduction/ao/. Date accessed December 26, 2008.

Nilsson, Nils J. 1998. *Artificial Intelligence: A New Synthesis*. The Morgan Kaufmann Series in Artificial Intelligence. Morgan Kaufmann Pub. Co.

Norvig, Peter. 2007. *SIAI Interview Series – Peter Norvig*.
URL http://www.acceleratingfuture.com/people-blog/?p=289. Date accessed October 12, 2008.

Odean, K. (editor). 1989. *High steppers, fallen angels, and lollipops: Wall Street slang*. Holt.

Odum, Eugene P. (1959). *Fundamentals of Ecology* (Second edition ed.). Philadelphia and London: W.B. Saunders Co.

Click on the ad to read more

Download free eBooks at bookboon.com

Pei, Mario. 1964. "A loss for words", *Saturday Review*, November 14: 82–84.

Python. 2008. "What is Python good for?". *General Python FAQ*. Python Foundation. URL http://www.python.org/doc/essays/blurb/. Date accessed October 23, 2008.

Pfeifer, Rolf and **Scheier**, Christian. 1999. *Understanding Intelligence*. MIT Press.

van Rossum, Guido. 2008. "Glue It All Together With Python". URL http://www.python.org/doc/essays/omg-darpa-mcc-position.html. Date accessed October 24. 2008. Date accessed October 23, 2008.

Rao, A.S. 1996. "AgentSpeak(L): BDI agents speak out in a logical computable language", in *Agents Breaking Away: Proceedings of the Seventh European Workshop on Modelling Autonomous Agents in a Multi-Agent World, LNAI 1038*, eds., W. Van de Velde and J.W. Perram, pages 42–55. Springer.

Reynolds, Craig. 1986. "Flocks, Herds and Schools: A Distributed Behavioral Model", Computer Graphics, 21(4), pages 25–34.

Reynolds, Craig. 1999. "Steering Behaviors for Autonomous Characters", *Proceedings of the Game Developers Conference*, San Jose, California. Pages 763–782.

Reynolds, Craig. 2008. "Stylized Depiction in Computer Graphics – Non-Photorealistic, Painterly and 'Toon Rendering: an annotated survey of online resources", URL http://www.red3d.com/cwr/npr/. Date accessed December 25, 2008.

Roussou, M. & **Drettakis**, G. 2003. Photorealism and Non-Photorealism in Virtual Heritage Representation. *Eurographics Workshop on Graphics and Cultural Heritage*, 5–7, 46–57.

Russell, Bertrand. 1926. *Theory of Knowledge for the Encyclopedia Britannica.*

Russell, Stuart and **Norvig**, Peter, 2002. *Artificial Intelligence: A Modern Approach.* Second edition. Prentice Hall.

Searle, John. 1980. *"Minds, Brains and Programs", Behavioral and Brain Sciences* **3** *(3): 417–457.*

Searle, John. 1999. "The Chinese Room", **in** Wilson, R.A. and F. Keil (eds.), *The MIT Encyclopedia of the Cognitive Sciences*, Cambridge: MIT Press.

Segaran, Toby. 2007. *Programming Collective Intelligence – Building Smart Web 2.0 Applications*. O'Reilly Media, Inc.

Download free eBooks at bookboon.com

Segaran, Toby. 2008. blog.kiwitobes.com.
URL http://blog.kiwitobes.com/?gclid=CNewwd6WzJYCFQO11Aod2jAjxQ. Date accessed October 29, 2008.

Shannon, C.E. 1948. "A mathematical theory of communication". *Bell System Technical Journal*, 27: 379–423, 623–656.

Shannon, C.E. 1951. "Prediction and entropy of printed English". *Bell System Technical Journal*, pages 50–64.

Simon, H.A. 1969. *The sciences of the artificial* (2nd ed.) Cambridge, MA. MIT Press.

Slocum, Terry. 2005. *Thematic Cartography and Geographic Visualization*. Second Edition. Upper Saddle River, NJ: Prentice Hall.

SMART. 2008. *SMART (Project Management) Wikipedia entry*.
URL http://en.wikipedia.org/wiki/SMART_criteria. Date accessed October 12, 2008.

Smith, Brian C. 1985. *Prologue to "Reflection and Semantics in a Procedural Language"*, in Readings in Knowledge Representation, edited by Brachman, R.J. & Levesque, H.J. Morgan Kaufmann.

Software Agent, 2008. *Wikipedia entry for 'Software Agent'*.
URL http://en.wikipedia.org/wiki/Software_agent. Date accessed December 26, 2008.

SPADES FAQ. 2008. URL http://development.pracucci.com/wikidoc/index.php/SPADES_FAQ. Date accessed December 25, 2008.

Taskin, H., **Ergun**, K., **Ocaktan**, M.A.B and **Selvi**, İ.H. 2006. "Agent based approach for manufacturing enterprise strategies". *Proceedings of 5th International Symposium on Intelligent Manufacturing Systems*, May 29–31, 2006: 361–370.

Turing, *Alan. 1950. "Computing Machinery and Intelligence", Mind LIX(236): 433–460.*

Whitby, Blay (1996), "The Turing Test: AI's Biggest Blind Alley?", in Millican, Peter & Clark, Andy, *Machines and Thought: The Legacy of Alan Turing*, **1**, pages 53–62, Oxford University Press.

Wilensky, U. 1999. NetLogo [computer software]. http://ccl.northwestern.edu/netlogo/. Center for Connected Learning and Computer-Based Modeling, Northwestern University, Evanston, IL.

Download free eBooks at bookboon.com

Winston, P.H. 1977. *Artificial Intelligence*. Addison Wesley Publishing Company.

Wooldridge, Micheal. 2002. *An Introduction to Multi-agent systems*. John Wiley and Sons.

Wooldridge, Michael and **Jennings**, N.R. 1995. "Intelligent Agents: Theory and Practice". *Knowledge Engineering Review*, 10(2), June.

Xiaocong, Fan, **Dianxiang**, Xu; **Jianmin**, Hou; **Guoliang**, Zheng. 1998."SPLAW: A computable agent-oriented programming language". *Proceedings First International Symposium onObject-Oriented Real-time Distributed Computing* (ISORC 98), 20–22, pages:144–145.

Yao, A.C. 1979. "Some Complexity Questions Related to Distributed Computing", *Proceedings of 11th ACM Symposium on Theory Of Computing* (STOC), pp. 209–213.

Zhang, Yu, **Lewis**, Mark and **Sierhuis**, Maarten. 2009. "12 Programming Languages, Environments, and Tools for Agent-Directed Simulation".
URL: http://www.cs.trinity.edu/~yzhang/research/papers/2009/Wiely09/ADSProgrammingLanguages EnvironmentsTools-Mark.doc. Date accessed 1st January, 2009.

Click on the ad to read more

Download free eBooks at bookboon.com