Artificial Intelligence – Agents and Environments

William John Teahan

bookboon
The eBook company

William John Teahan

Artificial Intelligence – Agents and Environments

Download free eBooks at bookboon.com

Artificial Intelligence – Agents and Environments
1st edition
© 2010 William John Teahan & bookboon.com
ISBN 978-87-7681-528-8

Download free eBooks at bookboon.com

Contents

	Preface	**7**
	AI programming languages and NetLogo	8
	Conventions used in this book series	9
	Volume Overview	11
	Acknowledgements	12
	Dedication	12
1	**Introduction**	**13**
1.1	What is "Artificial Intelligence"?	14
1.2	Paths to Artificial Intelligence	14
1.3	Objections to Artificial Intelligence	19
1.4	Conceptual Metaphor, Analogy and Thought Experiments	27
1.5	Design Principles for Autonomous Agents	31
1.6	Summary and Discussion	33

Masters in Management

Stand out from the crowd

Designed for graduates with less than one year of full-time postgraduate work experience, London Business School's Masters in Management will expand your thinking and provide you with the foundations for a successful career in business.

The programme is developed in consultation with recruiters to provide you with the key skills that top employers demand. Through 11 months of full-time study, you will gain the business knowledge and capabilities to increase your career choices and stand out from the crowd.

Applications are now open for entry in September 2011.

London Business School

London Business School
Regent's Park
London NW1 4SA
United Kingdom
Tel +44 (0)20 7000 7573
Email mim@london.edu
www.london.edu/mim/

For more information visit www.london.edu/mim/
email mim@london.edu or call +44 (0)20 7000 7573

Click on the ad to read more

Download free eBooks at bookboon.com

2	**Agents and Environments**	**34**
2.1	What is an Agent?	34
2.2	Agent-oriented Design Versus Object-oriented Design	39
2.3	A Taxonomy of Autonomous Agents	42
2.4	Desirable Properties of Agents	46
2.5	What is an Environment?	49
2.6	Environments as n-dimensional spaces	52
2.7	Virtual Environments	55
2.8	How can we develop and test an Artificial Intelligence system?	59
2.9	Summary and Discussion	61
3	**Frameworks for Agents and Environments**	**62**
3.1	Architectures and Frameworks for Agents and Environments	62
3.2	Standards for Agent-based Technologies	63
3.3	Agent-Oriented Programming Languages	65
3.4	Agent Directed Simulation in NetLogo	70
3.5	The NetLogo development environment	74
3.6	Agents and Environments in NetLogo	78
3.7	Drawing Mazes using Patch Agents in NetLogo	84
3.8	Summary	91

Click on the ad to read more

Download free eBooks at bookboon.com

4	**Movement**	**92**
4.1	Movement and Motion	93
4.2	Movement of Turtle Agents in NetLogo	94
4.3	Behaviour and Decision-making in terms of movement	96
4.4	Drawing FSMs and Decision Trees using Link Agents in NetLogo	98
4.5	Computer Animation	107
4.6	Animated Mapping and Simulation	117
4.7	Summary	120
5	**Embodiment**	**122**
5.1	Our body and our senses	123
5.2	Several Features of Autonomous Agents	125
5.3	Adding Sensing Capabilities to Turtle Agents in NetLogo	128
5.4	Performing tasks reactively without cognition	144
5.5	Embodied, Situated Cognition	156
5.6	Summary and Discussion	157
6	**References**	**159**

STUDY FOR YOUR MASTER'S DEGREE
IN THE CRADLE OF SWEDISH ENGINEERING

Chalmers University of Technology conducts research and education in engineering and natural sciences, architecture, technology-related mathematical sciences and nautical sciences. Behind all that Chalmers accomplishes, the aim persists for contributing to a sustainable future – both nationally and globally.

Visit us on **Chalmers.se** or **Next Stop Chalmers** on facebook.

Click on the ad to read more

Download free eBooks at bookboon.com

Preface

'Autumn_Landscape' by Adrien Taunay the younger.

> *The landscape we see is not a picture frozen in time only to be cherished and protected. Rather it is a continuing story of the earth itself where man, in concert with the hills and other living things, shapes and reshapes the ever changing picture which we now see. And in it we may read the hopes and priorities, the ambitions and errors, the craft and creativity of those who went before us. We must never forget that tomorrow it will reflect with brutal honesty the vision, values, and endeavours of our own time, to those who follow us.*
>
> Wall Display at Westmoreland Farms, M6 Motorway North, U.K.

Artificial Intelligence is a complex, yet intriguing, subject. If we were to use an analogy to describe the study of Artificial Intelligence, then we could perhaps liken it to a landscape, whose ever changing picture is being shaped and reshaped by man over time (in order to highlight how it is continually evolving). Or we could liken it to the observation of desert sands, which continually shift with the winds (to point out its dynamic nature). Yet another analogy might be to liken it to the ephemeral nature of clouds, also controlled by the prevailing winds, but whose substance is impossible to grasp, being forever out of reach (to show the difficulty in defining it). These analogies are rich in metaphor, and are close to the truth in some respects, but also obscure the truth in other respects.

Natural language is the substance with which this book is written, and metaphor and analogy are important devices that we, as users and producers of language ourselves, are able to understand and create. Yet understanding language itself and how it works still poses one of the greatest challenges in the field of Artificial Intelligence. Other challenges have included beating the world champion at chess, driving a car in the middle of a city, performing a surgical operation, writing funny stories and so on; and this variety is why Artificial Intelligence is such an interesting subject.

Download free eBooks at bookboon.com

Like the shifting sands mentioned above, there have been a number of important paradigm shifts in Artificial Intelligence over the years. The traditional or classical AI paradigm (the "symbolic" approach) is to design intelligent systems based on symbols, applying the information processing metaphor. An opposing AI paradigm (the "sub-symbolic" approach or connectionism) posits that intelligent behaviour is performed in a non-symbolic way, adopting an embodied behaviourist approach. This approach places an emphasis on the importance of physical grounding, embodiment and situatedness as highlighted by the works of Brooks (1991a; 1991b) in robotics and Lakoff and Johnson (1980) in linguistics. The main approach adopted in this series textbooks will predominantly be the latter approach, but a middle ground will also be described based on the work of Gärdenfors (2004) which illustrates how symbolic systems can arise out of the application of an underlying sub-symbolic approach.

The advance of knowledge is rapidly proceeding, especially in the field of Artificial Intelligence. Importantly, there is also a new generation of students that seek that knowledge – those for which the Internet and computer games have been around since their childhood. These students have a very different perspective and a very different set of interests to past students. These students, for example, may never even have heard of board games such as Backgammon and Go, and therefore will struggle to understand the relevance of search algorithms in this context. However, when they are taught the same search algorithms in the context of computer games or Web crawling, they quickly grasp the concepts with relish and take them forward to a place where you, as their teacher, could not have gone without their aid.

What Artificial Intelligence needs is a "re-imagination", like the current trend in science-fiction television series – to tell the same story, but with different actors, and different emphasis, in order to engage a modern audience. The hope and ambition is that this series textbooks will achieve this.

AI programming languages and NetLogo

Several programming languages have been proposed over the years as being well suited to building computer systems for Artificial Intelligence. Historically, the most notable AI programming languages have been Lisp and Prolog. Lisp (and related dialects such as Common Lisp and Scheme) has excellent list and symbol processing capabilities, with the ability to interchange code and data easily, and has been widely used for AI programming, but its quirky syntax with nested parenthesis makes it a difficult language to master and its use has declined since the 1990s.

Prolog, a logic programming language, became the language selected back in 1982 for the ultimately unsuccessful Japanese Fifth Generation Project that aimed to create a supercomputer with usable Artificial Intelligence capabilities.

Download free eBooks at bookboon.com

NetLogo (Wilensky, 1999) has been chosen to provide code samples in these books to illustrate how the algorithms can be implemented. The reasons for providing actual code are the same as put forward by Segaran (2007) in his book on Collective Intelligence – that this is more useful and "probably easier to follow", with the hope that such an approach will lead to a sort of new "middle-ground" in technical books that "introduce readers gently to the algorithms" by showing them working code (Segaran, 2008). Alternative descriptions such as pseudo-code tend to be unclear and confusing, and may hide errors that only become apparent during the implementation stage. More importantly, actual code can be easily run to see how it works and quickly changed if the reader wishes to make improvements without the need to code from scratch.

NetLogo (a powerful dialect of Logo) is a programming language with predominantly agent-oriented attributes. It has unique capabilities that make it an extremely powerful for producing and visualizing simulations of multi-agent systems, and is useful for highlighting various issues involved with their implementation that perhaps a more traditional language such as Java or C/C++ would obscure. NetLogo is implemented in Java and has very compact and readable code, and therefore is ideal for demonstrating complicated ideas in a succinct way. In addition, it allows users to extend the language by writing new commands and reporters in Java.

In reality, no programming language is suitable for implementing the full range of computer systems required for Artificial Intelligence. Indeed, there does not yet exist a single programming language that is up to the task. In the case of "behaviour-based AI" (and related fields such as embodied cognitive science), what is required is a fully agent-oriented language that has the richness of Java, but the agent-oriented simplicity of a language such as NetLogo.

An introduction to the Netlogo programming language and sample exercises to practice programming in NetLogo can be found throughout this series of books and in the accompanying series of books *Exercises for Artificial Intelligence* (where the chapters and related exercises mirror the chapters in this book.)

Conventions used in this book series

Important analogous relationships will be described in the text, for example: "A genetic algorithm in artificial intelligence is analogous to genetic evolution in biology". Its purpose is to make explicit the analogous relationship that underlies the natural language used in the surrounding text.

Download free eBooks at bookboon.com

An example of a design goal, design principle and design objective:

Design Goal 1:
An AI system should mimic human intelligence.

Design Principle 1:
An AI system should be an agent-oriented system.

Design Objective 1.1:
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.

The design goal is an overall goal of the system being designed. The design principle makes explicit a principle under which the system is being designed. A design objective is a specific objective of the system that we wish to achieve when the system has been built.

The meaning of various concepts (for example, agents, and environments) will be defined in the text, and alternative definitions also provided. For example, we can define an agent as having 'knowledge' if it knows what the likely outcomes will be of an action it may perform, or of an action it is observing. Alternatively, we can define knowledge as the absence of the need for search. These definitions should be regarded as 'working definitions'. The word 'working' is used here to emphasize that we are still expending effort on crafting the definition that suits our purposes and that it should *not* be considered to be a definition cast in stone. Neither should the definition be considered to be exhaustive, or all-inclusive. The idea is that we can use the definition until such time as it no longer suits our purposes, or until its weaknesses outweigh its strengths. The definitions proposed in this textbook are also working definitions in another sense – we (the author of this book, and the readers) all are learning and remoulding these definitions ourselves in our minds based on the knowledge we have gained and are gaining. The purpose of a working definition is to define a particular concept, but a concept itself is tenuous, something that is essentially a personal construct – within our own minds – so it never can be completely defined to suit everyone (see Chapter 9 for further explanation).

Artificial Intelligence researchers also like to perform "thought experiments". These are shown as follows:

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?

Download free eBooks at bookboon.com

NetLogo code will be shown as follows:

```
breed [agents agent]
breed [points point]
directed-link-breed [curved-paths curved-path]

agents-own [location] ;; holds a point

to setup
 clear-all ;; clear everything
end
```

All sample NetLogo code in this book can be found using the URLs listed at the end of each chapter as follows:

Model	URL
Two States	http://files.bookboon.com/ai/Two-States.nlogo

Model	NetLogo Models Library (Wilensky, 1999) and URL
Wolf Sheep Predation	Biology > Wolf Sheep Predation http://ccl.northwestern.edu/netlogo/models/WolfSheepPredation

In this example, the Two States model at the top of the table is one that has been developed for this book. The Wolf Sheep Predation model at the bottom comes with the NetLogo Models Library, and can be run in NetLogo by selecting "Models Library" in the File tab, then selecting "Biology" followed by "Wolf Sheep Predation" from the list of models that appear.

The best way to use these books is to try out these NetLogo models at the same time as reading the text and trying out the exercises in the companion *Exercises for Artificial Intelligence* books. An index of the models used in these books can be found using the following URL:

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

Volume Overview

The chapters in this volume are organized into two parts as follows:

Volume 1: Agent-Oriented Design.

Part 1: Agents and Environments

Chapter 1: Introduction.
Chapter 2: Agents and Environments.
Chapter 3: Frameworks for Agents and Environments.
Chapter 4: Movement.
Chapter 5: Embodiment.

Download free eBooks at bookboon.com

Part 2: Agent Behaviour I

Chapter 6: Behaviour.

Chapter 7: Communication.

Chapter 8: Search.

Chapter 9: Knowledge.

Chapter 10: Intelligence.

Volume 1 champions agent-oriented design in the development of systems for Artificial Intelligence. In Part 1, it defines what agents are, emphasizes the important role that environments have that determine the types of interactions that can occur, and looks at some frameworks for building agents and environments, in particular NetLogo. It then looks at two important aspects of agents – movement and embodiment – in terms of agent-environment interaction, and how it can affect behaviour. Part 2 looks at various aspects of agent behaviour in more depth and applies a behavioural perspective to the understanding of actions agents perform and traits they exhibit such as communication, searching, knowledge, and intelligence.

Volume 2 will continue examining aspects of agent behaviour such as problem solving, decision-making and learning. It will also look at some application areas for Artificial Intelligence, recasting them within the agent-oriented design perspective. The purpose will be to illustrate how the ideas put forward in this volume can be applied to real-life applications.

Acknowledgements

I would like to express my gratitude to everyone at Ventus Publications Aps who have been involved with the production of this volume.

I would like to thank Uri Wilensky for allowing me to include sample code for some of the NetLogo models that are listed at the end of each chapter.

I would also like to thank the students I have taught, for providing me with insights into the subject of Artificial Intelligence that I could not have gained without their input and questioning.

Dedication

These books and the accompanying books *Exercises for Artificial Intelligence* are dedicated to my wife Beata and my son Jakub, and to the memory of my parents, Joyce and Bill.

Download free eBooks at bookboon.com

1 Introduction

We set sail on this new sea because there is new knowledge to be gained, and new rights to be won, and they must be won and used for the progress of all people…

We choose to go to the moon. We choose to go to the moon in this decade and do the other things, not because they are easy, but because they are hard, because that goal will serve to organize and measure the best of our energies and skills, because that challenge is one that we are willing to accept, one we are unwilling to postpone, and one which we intend to win, and the others, too.

John F. Kennedy. Address at Rice University on the Nation's Space Effort,
September 12, 1962.

Click on the ad to read more

Download free eBooks at bookboon.com

The purpose of this chapter is to provide an introduction to Artificial Intelligence (AI). The chapter is organized as follows. Section 1.1 briefly defines what AI is. Section 1.2 describes different paths that could be taken that might lead to the development of AI systems. Section 1.3 discusses the various objections to AI research that have been put forward over the years. Section 1.4 looks at how conceptual metaphor and analogy are important devices used for describing concepts in language. A further device – a thought experiment – is also described. These will be used throughout the books to introduce or highlight important concepts. Section 1.5 describes some design principles for autonomous agents.

1.1 What is "Artificial Intelligence"?

Artificial Intelligence is the study of how to build computer systems that exhibit intelligence in some manner. Artificial Intelligence (or simply AI) has resulted in many breakthroughs in computer science – many core research topics in computer science today have developed out of AI research; for example, neural networks, evolutionary computing, machine learning, natural language processing, object-oriented programming, to name a few. In many cases, the primary focus for these research topics is no longer the development of AI, they have become a discipline in themselves, and in some cases, are no longer thought of as being related to AI any more. AI itself continues to move on in the search for further insights that will lead to the crucial breakthroughs that are still needed. Perhaps the reader might be the one to provide one or more of the crucial breakthroughs in the future. One of the most exciting aspects of AI is that there are still many ideas to be invented, many avenues still to be explored.

AI is an exciting and dynamic area of research. It is fast changing, with research over the years developing and continuing to *develop* many brilliant and interesting ideas. However, we have yet to achieve the ultimate goal of Artificial Intelligence. Many people dispute whether we will ever achieve it for reasons listed below. Therefore, anyone studying or researching AI should keep an open mind about the appropriateness of the ideas put forward. They should always question how well the ideas work by asking whether there are better ideas or better approaches.

1.2 Paths to Artificial Intelligence

Let us make an analogy between AI research and exploration of uncharted territory; for example, imagine the time when the North American continent was being explored for the first time, and no maps were available. The first explorers had no knowledge of the terrain they were exploring; they would head out in one direction to find out what was out there. In the process, they might record what they found out, by writing in journals, or drawing maps. These would then aid latter explorers, but for most of the early explorers, the terrain was essentially unknown, unless they were to stick to the same paths that the first explorers used.

Download free eBooks at bookboon.com

AI research today is essentially still at the early exploration stage. Most of the terrain to be explored is still unknown. The AI explorer has many possible paths that they can explore in the search for methods that might lead to machine intelligence. Some of those paths will be easy going, and lead to fertile lands; others will lead to mountainous and difficult terrain, or to deserts. Some paths might lead to impassable cliffs. Whatever the particular path poses for the AI researchers, the search promises to be an exciting one as it is in our human nature to want to explore and find out things.

We can have a look at the paths chosen by past 'explorers' in Artificial Intelligence. For example, analyzing the question "*Can computers think*?" has lead to many intense debates in the past resulting in different paths taken by AI researchers. Nilsson (1998) has pointed out that we can stress each word in turn to put a different perspective on the question. (He used the word "machines", but we will use the word "computers" instead). Take the first word – i.e. "*Can* computers think?" Do we mean: "*Can* computers think (someday)? Or "*Can* they think (now)?" Or do we mean they might be able to (in principle) but we would never be able to build it? Or are we asking for an actual demonstration? Some people think that thinking machines might have to be so complex we could never build them. Nilsson makes an analogy with trying to build a system to duplicate the earth's weather, for example. We might have to build a system no less complex than the actual earth's surface, atmosphere and tides. Similarly, full-scale human intelligence may be too complex to exist apart from its embodiment in humans situated in an environment. For example, how can a machine understand what a 'tree' is, or what an 'apple' tastes like without being embodied in the real world?

Or we could stress the second word – i.e. "Can *computers* think?" But what do we mean by 'computers'? The definition of computers is changing year by year, and the definition in the future may be very different to what it is today, with recent advances in molecular computing, quantum computing, wearable computing, mobile computing, and pervasive/ubiquitous computing changing the way we think about computers. Perhaps we can define a computer as being a machine. Much of the AI literature uses the word 'machine' interchangeably with the word computer – that is, the question "Can *machines* think?" is often thought of as being synonymous with "Can *computers* think?" But what are machines?

And are humans a machine? (If they are, as Nilsson says, then machines can think!) Nilsson points out that scientists are now beginning to explain the development and functioning of biological organisms the same way as machines (by examining the genome 'blueprint' of each organism). Obviously, 'biological' machines made of proteins can think (us!), but could 'silicon' based machines ever be able to think?

Download free eBooks at bookboon.com

And finally we can stress the third word – i.e. "Can computers *think*?" But what does it mean to think? Perhaps we mean *to "think" like we (humans) do*. Alan Turing (1950), a British mathematician, and one of the earliest AI researchers, devised a now famous (as well as contentious) empirical test for intelligence that now bears his name – the Turing Test. In this test, a machine attempts to convince a human interrogator that it is human. (See Thought Experiment 1.1 below). This test has come in for intense criticism in AI literature, perhaps unfairly, as it is not clear whether the test is a true test for intelligence. In contrast, an early AI goal of similar ilk, the goal to have an AI system beat the world champion at chess, has come in for far less criticism.

Thought Experiment 1.1: The Turing Test.

Imagine a situation where you are having separate conversations with two other people you cannot see in separate rooms, perhaps via a teletype (as in Alan Turing's day), or perhaps in a chat room via the Internet (if we were to modernize the setting). One of these people is a man, the other a woman – you do not know which. Your goal is to determine which is which by having a conversation with each of them and asking them questions. Part of the game is that the man is trying to trick you into believing that he is the woman not the other way round (the inspiration for Turing's idea came from the common Victorian parlour game called the Imitation Game).

Now imagine that the situation is changed, and instead of a man and a woman, the two protagonists are a computer and a human instead. The goal of the computer is to convince you that it is the human, and by doing so therefore pass this test for intelligence, now called the "Turing Test".

How realistic is this test? Joesph Weizenbuam built one of the very first chatbots, called ELIZA, back in 1966. His secretary found the program running on one computer and started poring out her life's story over a period of a few weeks, and was horrified when Weizenbaum told her it was just a program. However, this was not a situation where the Turing Test was passed. The Turing Test is an adversarial test in the sense that it is a game where one side is trying to fool the other, but the other side is aware of this and trying not to be fooled. This is what makes the test a difficult test to pass for an Artificial Intelligence system. Similarly, there are many websites on the Internet today that claim that their chatbot has passed the Turing Test; however, until very recently, no chatbot has even come close.

There is an open (and often maligned) contest, called the Loebner Contest, which is held each year where developers get to test out their AI chatbots to see if they can pass the Turing Test. The 2008 competition was notable in that the best AI was able to fool a quarter of the judges into believing it was human, a substantial progress over results in previous years. This provides hope that a computer will be able to pass the Turing Test in the not too distant future.

However, is the Turing Test really a good test for intelligence? Perhaps when a computer has passed the ultimate challenge of fooling a panel of AI experts, then we can evaluate how effective that computer is in tasks other than the Turing Test situation. Then by these further evaluations will we be able to determine how good the Turing Test really is (or isn't). After all, a computer has already beaten the world chess champion, but only by using search methods with evaluation functions that use minimal 'intelligence'. And what have we really learnt about intelligence from that – apart from how to build better search algorithms? Notably, the goal of getting a computer to beat the world champion has come in for far less criticism than passing the Turing Test, and yet, the former has been achieved whereas the latter has not (yet).

The debate surrounding the Turing Test is aptly demonstrated by the work of Robert Horn (2008a, 2008b). He has proposed a visual language as a form of visual thinking. Part of his work has involved the production of seven posters that summarize the Turing debate in AI to demonstrate his visual language and visual thinking. The seven posters cover the following questions:

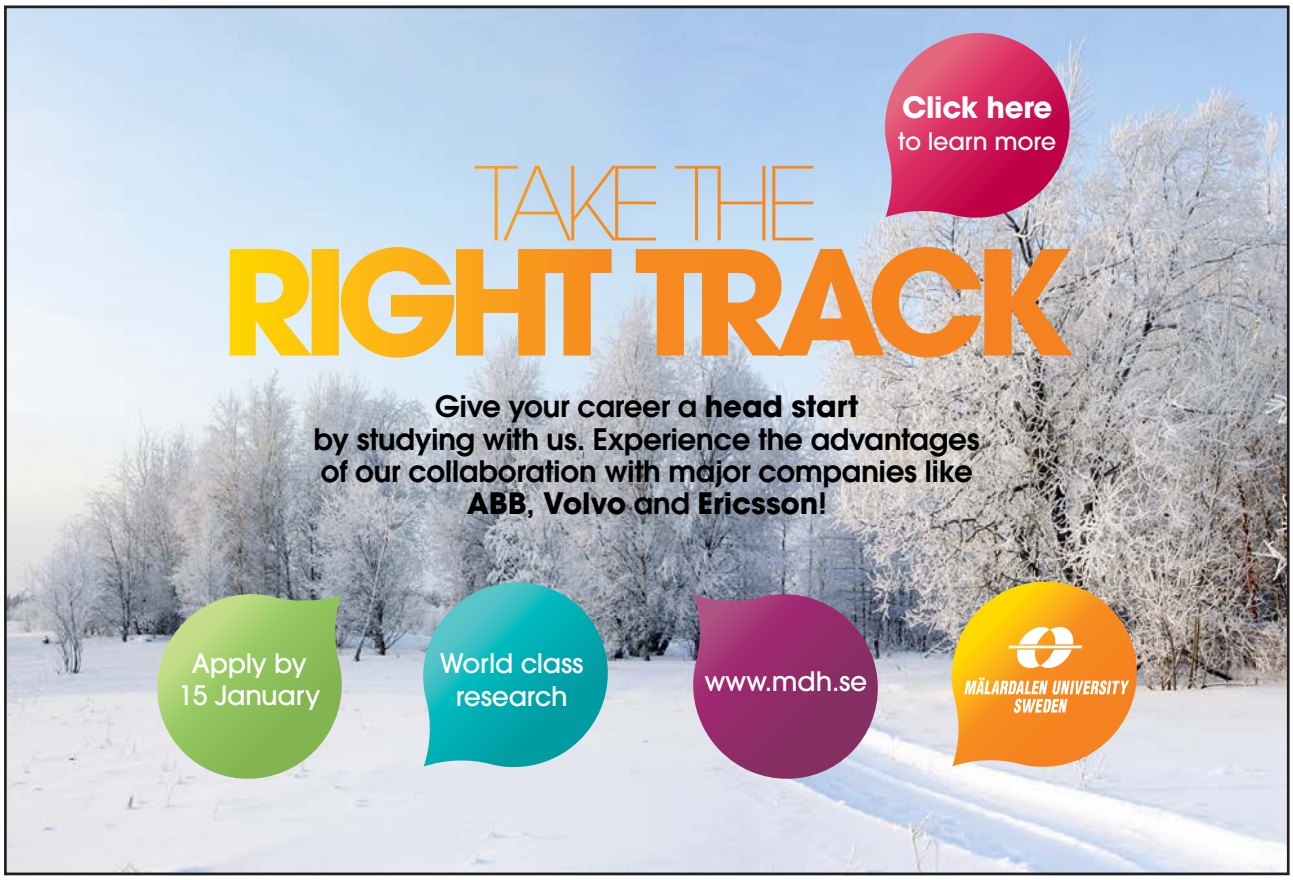
Download free eBooks at bookboon.com

1. Can computers think?
2. Can the Turing Test determine whether computers can think?
3. Can physical symbol systems think?
4. Can Chinese rooms think?
5. (i) Can connectionist networks think? and (ii) Can computers think in images?
6. Do computers have to be conscious to think?
7. Are thinking computers mathematically possible?

These posters are called 'maps' as they provide a 2D map of which questions have followed other questions using an analogy of researchers exploring uncharted territory.

The first poster maps the explorations for the question "Can computers think?", and shows paths leading to further questions as listed below:

- Can computers have free will?
- Can computers have emotions?
- Should we pretend that computers will never be able to think?
- Does God prohibit computers from thinking?
- Can computers understand arithmetic?
- Can computers draw analogies?



Click on the ad to read more

Download free eBooks at bookboon.com

- Are computers inherently disabled?
- Can computers be creative?
- Can computers reason scientifically?
- Can computers be persons?

The second poster explores the Turing Test debate: "Can the Turing Test determine whether computers can think?" A selection of further questions mapped on this poster include:

- Can the imitation game determine whether computers can think?
- If a simulated intelligence passes, is it intelligent?
- How many machines have passed the test?
- Is failing the test decisive?
- Is passing the test decisive?
- Is the test, behaviorally or operationally construed, a legitimate intelligence test?

One particular path to Artificial Intelligence that we will follow is the design principle that an AI system should be constructed using the agent-oriented design pattern rather than an alternative such as the object-oriented design pattern. Agents embody a stronger notion of autonomy than objects, they decide for themselves whether or not to perform an action on request from another agent, and they are capable of flexible (reactive, proactive, social) behaviour, whereas the standard object model has nothing to say about these types of behaviour and objects have no control over when they are executed (Wooldridge, 2002, pages 25–27). Agent-oriented systems and their properties are discussed in more detail in Chapter 2.

Another path we will follow is to place a strong emphasis on the importance of behaviour based AI and of embodiment, and situatedness of the agents within a complex environment. The early groundbreaking work in this area was that of Brooks in Robotics (1986) and Lakoff and Johnson in linguistics (1980). Brooks' subsumption architecture, now popular in robotics and used in other areas such as behavioural animation and intelligent virtual agents, adopts a modular methodology of breaking down intelligence into layers of behaviours that control everything an agent does based on the agent being physically situated within its environment and reacting with it dynamically. Lakoff and Johnson highlight the importance of conceptual metaphor in natural language (such as the use of the words 'groundbreaking' at the beginning of this paragraph) and how it is related to our perceptions via our embodiment and physical grounding. These works have laid the foundations for the research areas of embodied cognitive science and situated cognition, and insights from these areas will also be drawn upon throughout these textbooks.

Download free eBooks at bookboon.com

1.3 Objections to Artificial Intelligence

There have been many objections made to Artificial Intelligence over the years. This is understandable, to some extent, as the notion of an intelligent machine that can potentially out-smart and out-think us in the future is scary. This is perhaps fueled by many unrealistic science fiction novels and movies produced over the last century that have dwelt on the popular theme of robots either destroying humanity or taking over the world.

Artificial Intelligence has the potential to disrupt every aspect of our present lives, and this uncertainty can also be threatening to people who worry about what changes might bring in the future. The following technologies have been identified as emerging, potentially "disruptive" technologies that offer "hope for the betterment of the human condition", in a report titled "*Future Technologies, Today's Choices*" commissioned for Greenpeace Environmental Trust (Arnall, 2007):

- Biotechnology;
- Nanotechnology;
- Cognitive Science;
- Robotics;
- Artificial Intelligence.

The last three of these directly relate to the area of machine intelligence and all can be characterized as being potentially disruptive, enabling and interdisciplinary. A major effect of these emerging technologies will be product diversity ("their emergence on the market is anticipated to 'affect almost every aspect of our lives' during the coming decades"). Disruptive technologies displace older technologies and "enable radically new generations of existing products and processes to take over", and enable completely new classes of products that were not previously feasible.

As the report says, "The implications for industry are considerable: companies that do not adapt rapidly face obsolescence and decline, whereas those that do sit up and take notice will be able to do new things in almost every conceivable technological discipline". To illustrate the profound effect a disruptive technology can have on society, one only has to consider the example of the PC, and more recently search engines such as Google, and the effect these technologies have had on modern society.

John Searle (1980) has devised a highly debated objection to Artificial Intelligence. He proposed a thought experiment now called the "Chinese Room" to argue how an AI system would never have a mind like humans have, or have the ability to understand the way we do (see Thought Experiment 1.2).

Download free eBooks at bookboon.com

Thought Experiment 1.2: Searle's Chinese Room.

Imagine you have a computer program that can process Chinese characters as input and produce Chinese characters as output. This program, if good enough, would have the ability to pass the Turing Test for Chinese – that is, it can convince a human that it is a native Chinese speaker. According to proponents of the Turing Test (Searle argues) this would then mean that computers have the ability to understand Chinese.

Now also imagine one possible way that the program works. A person who knows only English has been locked in a room. The room is full of boxes of Chinese symbols (the 'database') and contains a book of instructions in English (the 'program') on how to manipulate strings of Chinese characters. The person receives the original Chinese characters via some input communication device. He then consults a book and follows the instructions dutifully, and produces the output stream of Chinese characters that he then sends through the output communication device.

The purpose of this thought experiment is to argue that although a computer program may have the ability to converse in natural language, there is no actual understanding taking place. Computers merely have the ability to use syntactic rules to manipulate symbols, but have no understanding of the meaning (or semantics) of them. Searle (1999) has this to say: "The point of the argument is this: if the man in the room does not understand Chinese on the basis of implementing the appropriate program for understanding Chinese then neither does any other digital computer solely on that basis because no computer, qua computer, has anything the man does not have."

Click on the ad to read more

Download free eBooks at bookboon.com

There have been many responses to Searle's argument. As with many AI thought experiments such as this one, the argument can simply be considered as not being an issue. AI researchers usually ignore it, as Searle's argument does not stop us from building useful AI systems that act intelligently, and whether they have a mind or think the same way our brain does is irrelevant. Stuart Russell and Peter Norvig (2002) observe that most AI researchers "don't care about the strong AI hypothesis – as long as the program works, they don't care whether you call it a simulation of intelligence or real intelligence."

Turing (1950) himself posed the following nine objections to Artificial Intelligence which provide a good summary of most of the objections that have arisen in the intervening years since his paper was published.

1.3.1 The theological objection

This argument is raised purely from a theological perspective – only humans with an immortal soul can think, and God has given an immortal soul only to humans, not to animals or machines. Turing did not approve of such theological arguments, but did argue against this from a theological point of view. A further theological concern is that the creation of Artificial Intelligence is usurping God's role as the creator of souls. Turing used the analogy of human procreation to point out that we also have a role to play in the creation of souls.

1.3.2 The "Heads in the Sand" objection

For some people, thinking about the consequences of a machine that can think is too dreadful to think about. This argument is for people who like to keep their "heads in the sand", and Turing thought the argument so spurious that he did not bother to refute it.

1.3.3 The Mathematical objection

Turing acknowledged this objection based on mathematical reasoning as having more substance than the first two. It has been raised by a number of people since including philosopher John Lucas and physicist Roger Penrose. According to Gödel's incompleteness theorem, there are limits based on logic to the questions a computer can answer, and therefore a computer would have to get some answers wrong. However, humans are also often wrong, so a fallible machine might offer a more believable illusion of intelligence. Additionally, logic itself is a limited form of reasoning, and humans often do not think logically. To object to AI based on the limitations of a logic-based solution ignores that there are alternative non logic-based solutions (such as those adopted in embodied cognitive science, for example) where logic-based mathematical arguments are not applicable.

Download free eBooks at bookboon.com

1.3.4 The argument from consciousness

This argument states that a computer cannot have conscious experiences or understanding. A variation of this argument is John Searle's Chinese Room thought experiment. Geoffrey Jefferson in his 1949 Lister Oration summarizes the argument: "Not until a machine can write a sonnet or compose a concerto because of thoughts and emotions felt, and not by the chance fall of symbols, could we agree that machine equals brain – that is, not only write it but know that it had written it. No mechanism could feel (and not merely artificially signal, an easy contrivance) pleasure at its successes, grief when its valves fuse, be warmed by flattery, be made miserable by its mistakes, be charmed by sex, be angry or depressed when it cannot get what it wants." Turing noted that this argument appears to be a denial of the validity of the Turing Test: "the only way by which one could be sure that a machine thinks is to be the machine and to feel oneself thinking". This is, of course, impossible to achieve, just as it is impossible to be sure that anyone else thinks, has emotions and is conscious the same way we ourselves do. Some people argue that consciousness is not only the preserve of humans, but that animals also have consciousness. So the lack of a universally accepted definition of consciousness presents problems for this argument.

1.3.5 Arguments from various disabilities

These arguments take the form that a computer can do many things but it would never be able to *X*. For *X*, Turing offered the following selection: "be kind, resourceful, beautiful, friendly, have initiative, have a sense of humour, tell right from wrong, make mistakes, fall in love, enjoy strawberries and cream, make some one fall in love with it, learn from experience, use words properly, be the subject of its own thought, have as much diversity of behaviour as a man, do something really new." Turing noted that little justification is usually offered to support these arguments, and that some of them are just variations of the consciousness argument. This argument also overlooks the versatility of machines and the sheer inventiveness of humans who build them. Much of Turing's list has already been achieved in varying degrees except for falling in love and enjoying strawberries and cream. (Turing acknowledged the latter would be an "idiotic" thing to get a machine to do). Affective agents have already been built to be kind and friendly. Some virtual agents and computer game AIs have initiative and are extremely resourceful. Conversational agents know how to use words properly; some have a sense of humour and can tell right from wrong. It is very easy to program a machine to make a mistake.

Some computer generated composite faces and the face of Jules the androgynous robot (Brockway, 2008) are statistically perfect, therefore can be considered beautiful. Self-awareness, or being the subject of one's own thoughts, has already been achieved by the robot Nico in a limited sense (see Thought Experiment 10.1). Storage capacities and processing capabilities of modern computers place few boundaries on the number of behaviours a computer can exhibit. (One only has to play a computer game with complex AI to observe a large variety of artificial behaviours). And for getting computers to do something really new, see the next objection.

Download free eBooks at bookboon.com

1.3.6 Lady Lovelace's objection

This objection states that computers are incapable of original thought. Lady Loveless penned a memoir in 1842 (contained in detailed information of Babbage's Analytical Engine) stating that: "The Analytical Engine has no pretensions to *originate* anything. It can do *whatever we know how to order it* to perform" (her italics). Turing argued that the brain's storage is quite similar to that of a computer, and there is no reason to think that computers are not able to surprise humans. Indeed, the application of genetic programming has produced many patentable new inventions. For example, NASA used genetic programming to evolve an antenna that was deployed on a spacecraft in 2006 (Lohn *et al*., 2008). This antenna was considered to be human-competitive as it yielded similar performance to human designed antenna, but its design was completely novel.

1.3.7 Argument from Continuity in the Nervous System

Turing acknowledged that the brain is not digital. Neurons fire with pulses that have analog components. Turing suggests that any analog system can readily be simulated to any degree of accuracy. Another form of this argument is that the brain processes signals (from *stimuli*) rather than *symbols*. There are two paradigms in AI – symbolic and sub-symbolic (or connectionist) – that protagonists claim as the best way forward in developing intelligent systems. The former emphasizes a top-down symbol processing approach in the design (knowledge-based systems are one example), whereas the latter emphasizes a bottom-up approach with symbols being physically grounded in some way (for example, neural networks). The symbolic versus sub-symbolic paradigms has been a fierce debate in AI and cognitive science over the years, and as with all debates, proponents have often taken mutually exclusive viewpoints. Methods which combine aspects of both approaches have some merit such as conceptual spaces (Gärdenfors, 2000), which emphasizes that we represent information on the conceptual level – that is, *concepts* are a key component, and provide a link between stimuli and symbols.

1.3.8 The Argument from Informality of Behaviour

Humans do not have a finite set of behaviours – they improvise based on the circumstances. Therefore, how could we devise a set of rules or laws that would describe what a person should do in every conceivable set of circumstances? Turing put this argument in the following way: "if each man had a definite set of rules of conduct by which he regulated his life he would be no better than a machine. But there are no such rules, so men cannot be machines." Turing argues that just because we do not know what the laws are, this does not mean that no such laws exist. This argument also reveals a misconception of what a computer is capable of. If we think of computers as a 'machine', we can easily make the mistake of using the narrower meaning of the term which we may associate with the many machines we use in daily life (such as a power-drill or car). But some machines – i.e. computers – are capable of much more than these simpler machines. They are capable of autonomous behaviour, and can observe and react to a complex environment, thereby producing the desired complexity of behaviour as a result. Some also exhibit *emergent* (non pre-programmed) behaviour from their interactions with the environment, such as the feet tapping behaviour of virtual spiders (ap Cenydd and Teahan, 2005), which mirrors the behaviour of spiders in real life.

Download free eBooks at bookboon.com

1.3.9 The Argument from Extrasensory Perception

This last objection is of less relevance today as it reflects the interest in Extra Sensory Perception (ESP) that was prevalent at the time Turing published his paper. The argument is that if ESP is possible in humans, then that could be exploited to invalidate the Turing Test. (A computer might only be able to make random predictions in a card guessing game, whereas a human with mind-reading abilities might be able to guess better than chance.) Turing discussed ways in which the conditions of the test could be altered to overcome this.

Another objection relates to the perceived lack of concrete results that AI research has produced in over half a century of endeavour. The Greenpeace report mentioned earlier made clear the continuing failure of AI research: "*Current AI systems are, it is argued, fundamentally incapable of exhibiting intelligence as we understand it.*" The term "AI Winter" refers to the view that research and development into Artificial Intelligence is on the wane, and has been for some time. Related to this is the belief that Artificial Intelligence is no longer a worthy research area since it has (in some people's minds) failed spectacularly in delivering on its promises ever since the term was coined at the seminal Dartmouth conference in 1956 (this conference is now credited with introducing the term "Artificial Intelligence").

Click on the ad to read more

Download free eBooks at bookboon.com

Contrary to the myth that there exists an AI winter, the rate of research is rapidly expanding in Artificial Intelligence. One of the main drivers for future research will be the entertainment industry – the need for realistic interaction with NPCs (Non-Playing Characters) in the games industry, and the striving for greater believability in the related movie and TV industries. These industries have substantial financial clout, and have almost unlimited potential for the application of AI technology. For example, a morphing of reality TV with online computer games could lead to fully interactive TV in the not too distant future where the audience will become immersed in, and be able to influence, the story they are watching (through voting on possible outcomes – e.g. whether to kill off one of the main actors). An alternative possibility could be the combination of computer animation, simulation and AI technologies that could lead to movies that one could watch many times, each time with different outcomes depending on what happened during the simulation.

Despite these interesting developments in the entertainment industry where AI is not seen as much of a threat, the increasing involvement of AI technologies in other aspects of our daily lives has been of growing concern to many people. Kevin Warwick in his 1997 book *The March of the Machines* has predicted that robots or super-intelligent machines will forcibly take over from the human race within the next 50 years. Some of the rationale behind this thinking is the projection that computers will outstrip the processing power of the human brain by as early as 2020 (Moravec, 1998; see Figure 1.1). For example, this projection has predicted that computers already have the processing ability of spiders – and recent Artificial Life simulations of arthropods has shown how it is possible now to produce believable dynamic animation of spiders in real-time (ap Cenydd and Teahan, 2005). The same framework used for the simulations has been extended to encompass lizards. Both lizard and spider equivalent capability was projected by Moravec to already have been achieved. However, unlike Moravec's graph, the gap between virtual spiders and virtual lizards was much smaller. If such a framework can be adapted to mimic mammals and humans, then believable human simulations may be closer than was first thought.

Misconceptions concerning machines taking over the human race which play on people's uninformed worries and fears, can unfortunately have an effect on public policy towards research and development. For example, a petition from the Institute of Social Inventions states the following:

"In view of the likelihood that early in the next millennium computers and robots will be developed with a capacity and complexity greater than that of the human brain, and with the potential to act malevolently towards humans, we, the undersigned, call on politicians and scientific associations to establish an international commission to monitor and control the development of artificial intelligence systems." (Reported in Malcolm, 2008).

Download free eBooks at bookboon.com

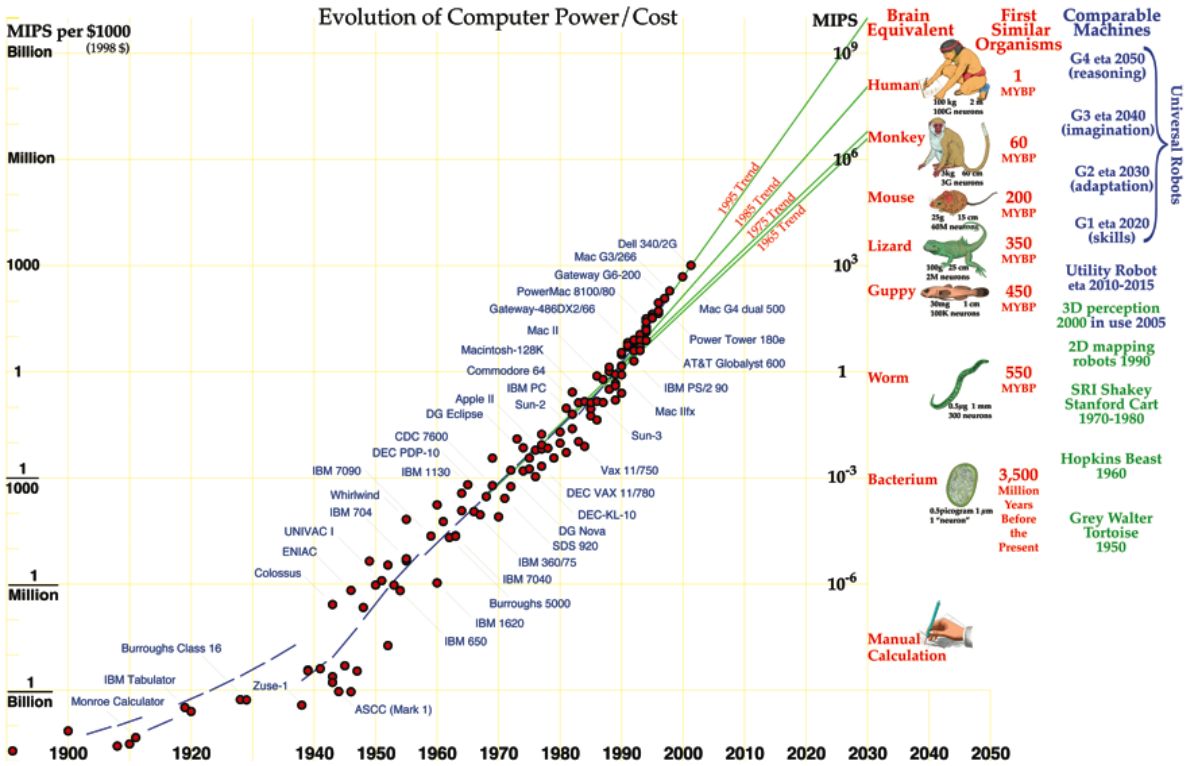


Figure 1.1: Evolution of computer power/cost compared with brainpower equivalent. Courtesy of Hans Moravec (1998).

Click on the ad to read more

Download free eBooks at bookboon.com

Chris Malcolm (2008) provides convincing arguments in a series of papers why robots will not rule the world. He points out that the rate of increase in *intelligence* is much slower than the rate of increase in processing power. For example, Moravec (2008) predicts that we will have fully intelligent robots by 2050 although we will have computers with greater processing power than the brain by 2020. Malcolm also highlights the dangers of "anthropomorphising and over-interpreting everything". For example, it is difficult to avoid not attributing emotions and feelings when observing Hiroshi Ishiguro's astonishingly life-like artificial clone of himself called Geminoid, or Hanson Robotics' androgynous android Jules (Brockway, 2008). Joseph Weizenbaum, who developed Eliza, a chatbot with an ability to simulate a Rogerian psychotherapist and one of the first attempts at passing the Turing Test, was so concerned about the uninformed responses of people who insisted on treating Eliza as a real person that he concluded that "the human race was simply not intellectually mature enough to meddle with such a seductive science as artificial intelligence" (Malcolm, 2008).

1.4 Conceptual Metaphor, Analogy and Thought Experiments

Much of language (as used in this textbook, for example) is made up of conceptual metaphor and analogy. For example, the analogy between AI research and physical exploration in Section 1.2 uses examples of a conceptual metaphor that links the concepts 'AI research' and 'exploration'. Lakoff and Johnson (1980) highlight the important role that conceptual metaphor plays in natural language and how they are linked with our physical experiences. They argue that metaphor is pervasive not just in everyday language, but in our thoughts and action, being a fundamental feature of the human conceptual system.

Recognizing the use of metaphor and analogy in language can aid understanding and facilitate learning. A conceptual metaphor framework, for example, has been devised for biology and for the teaching of mathematics. Analogy and conceptual metaphor are important linguistic devices for explaining relationships between concepts. A metaphor is understood by finding an analogy mapping between two domains – between a more abstract target conceptual domain that we are trying to understand and the source conceptual domain that is the source of the metaphorical expressions. Lakoff and Johnson closely examined commonly used conceptual metaphors such as "LIFE IS A JOURNEY", "ARGUMENT IS WAR" and "TIME IS MONEY" that appear in everyday phrases we use in language. Some examples are "I have my life *ahead* of me", "He *attacked* my argument" and "I've *invested* a lot of time in that". Understanding of these sentences requires the reader or listener to apply features from the more understood concepts such as JOURNEY, WAR and MONEY to the less understood, more abstract concepts such as LIFE, ARGUMENT and TIME. In many cases, the more understood or more 'concrete' concept is taken from a domain that relates to our physically embodied human experience (such as the "UP IS GOOD" metaphor used in the phrase "Things are lookup *up*"). Another example is the cartographic metaphor (MacroVu, 2008b) that is the basis behind the 'maps' of Robert Horn mentioned above in Section 1.2.

Download free eBooks at bookboon.com

Analogy, like metaphor, draws a similarity between things that initially might seem different. In some respects, we can consider analogy a form of argument whose purpose is to bring to the forefront the relationship between the pairs of concepts being compared, highlight further similarities, and help provide insight by comparing an unknown subject to a more familiar one. Analogy seems similar to metaphor in the role it plays, so how are they different? According to the Merriam-Webster's Online Dictionary, a metaphor is "a figure of speech in which a word or phrase literally denoting one kind of object or idea is used in place of another to suggest a likeness or analogy between them (as in *drowning in money*)". Analogy is defined as the "inference that if two or more things agree with one another in some respects they will probably agree in others" and also "resemblance in some particulars between things otherwise unlike". The essential difference is that metaphor is a figure of speech where one thing is used to *mean* another, whereas analogy is not just a figure of speech – it can be a logical argument that if two things are alike in some ways, they will be alike in other ways as well.

The language used to describe computer science and AI is often rich in the use of conceptual metaphor and analogy. However, they are seldom stated explicitly, and instead the reader is often left to infer the implicit relationship being made from the words used. We will use analogy (and conceptual metaphor where appropriate) in these textbooks to highlight explicitly how two concepts are related to each other, as shown below:

- A 'computer virus' in computer science is analogous to a 'virus' in real life.
- A 'computer worm' in computer science is analogous to a 'worm' in real life.
- A 'Web spider' in computer science is analogous to a 'spider' in real life.
- The 'Internet' in computer science is analogous to a 'spider's web in real life.
- A 'Web site' in computer science is analogous to an 'environment' in real life.

In these examples, an analogy has been explicitly stated between the computer science concepts 'computer virus, 'computer worm, 'Web spider and 'Internet' and their real life equivalents. Many features (but not all of them) of the related concept (such as a virus in real life) are often used to describe features of the abstract concept (a computer virus) being explained. These analogies need to be kept in mind in order to understand the language that is being used to describe the concepts.

For example, when we use the phrase "crawling the web", we can only understand its implicit meaning in the context of the third and fourth analogies above. Alternative analogies (e.g. the fifth analogy) lay behind the meaning of different metaphors used in phrases such as "getting lost while searching the Web" and "surfing the Web". When a person says they got lost while exploring the Web, they are not physically lost. In addition, it would feel strange to talk about a real spider 'surfing' its web, but we can talk about a person surfing the Web because we are making an analogy that the Web is like a wave in real life. Sample metaphors related to this analogy are phrases such as 'flood of information' and 'swamped by information overload'. The analogy is one of trying to maintain balance on top of a wave of information over which you have no control.

Download free eBooks at bookboon.com

Two important analogies used in AI concerning genetic algorithms and neural networks have a biological basis:

- A 'genetic algorithm' in Artificial Intelligence is analogous to genetic evolution in biology.
- A 'neural network' in Artificial Intelligence is analogous to the neural processing in the brain.

These are examples of 'natural computation' – computing that is inspired by nature.

In some cases, there are competing analogies being used in the language, and in this case we need to clarify each analogy further by specifying points of similarity and dissimilarity (where each analogy is strong or breaks down, respectively) and by providing examples of metaphors used in the text that draw out the analogy. For example, an analogy can be made between the target concept 'research' and the competing source concepts 'exploration' and 'construction' as follows:

Click on the ad to read more

Download free eBooks at bookboon.com

Analogy 1 'Research' in science is analogous to 'exploration' in real life.

Points of similarity: The word 'research' itself also uses the exploration analogy: we can think of it as a process of going back over (repeating) search we have already done.

Points of dissimilarity: Inventing new ideas is more complicated than just exploring a new path. You have to *build* on existing ideas, *create* or *construct* something new from *existing parts*.

Examples of metaphor used in this chapter: "We *set sail on this new sea* because there is new knowledge to be gained", "*Paths* to Artificial Intelligence", "Most of the *terrain to be explored* is still unknown".

Analogy 2 'Research' in science is analogous to 'construction' in real life.

Points of similarity: We often say that new ideas are *made* or *built* from existing ideas; we also talk about *frameworks* that provide *support* or *structure* for a particular idea.

Points of dissimilarity: Inventing new ideas is more complicated than just constructing or building something new. Sometimes you have to *go* where you have *never gone before*; sometimes you *get lost* along the way (something that seems strange to say if you are constructing a building).

Examples of metaphor used in this chapter: "how to *build* better search algorithms", "Let us *make* an analogy", "*build* on existing ideas", "little justification is usually offered to *support* these arguments".

Thought experiments (see examples in this chapter and subsequent chapters) provide an alternative method for describing a new idea, or for elaborating on problems with an existing idea. The analogy behind the term 'thought experiment' is that we are conducting some sort of experiment (like a scientist would in a laboratory), but this experiment is being conducted only inside our mind. As with all experiments, we try out different things to see what might happen as a result, the only difference is that the things we try out are to the most part only done inside our own thoughts. There is no actual experimentation done – it is just a reasoning process that is being used by the person proposing the experiments.

In a thought experiment, we are essentially posing "What if" questions in our own minds. For example, 'What if *X*?' or 'What happens if *X*?' where *X* might be "we can be fooled into believing a computer is a human" for the Turing Test thought experiment. Further, the person who proposes the thought experiment is asking other people to conduct the same thought process in their own minds by imagining a particular situation, and the likely consequences. Often the thought experiment involves putting oneself into the situation (in your mind), and then imagining what would happen. The purpose of the thought experiment is to make arguments for or against a particular point of view by highlighting important issues.

The German term for a thought experiment is *Gedankenexperiment* – there are many examples used in physics, for example. One of the most famous posed by Albert Einstein was that of chasing a light beam and led to the development of Special Relativity. Artificial Intelligence also has many examples of thought experiments, and several of these are described throughout these textbooks to illustrate important ideas and concepts.

Download free eBooks at bookboon.com

1.5 Design Principles for Autonomous Agents

Pfeifer and Scheier (1999, page 303) propose several design principles for autonomous agents:

Design 1.1 Pfeifer and Scheier's design principles for autonomous agents.

Design Meta-Principle: The 'three constituents principle'.
This first principle is classed as a meta-principle as it defines the context governing the other principles. It states that the design of autonomous agents involves three constituents: (1) the ecological niche; (2) the desired behaviours and tasks; and (3) the agent itself. The 'task environment' covers (1) and (2) together.

Design Principle 1: The 'complete-agent principle'.
Agents must be complete: autonomous; self-sufficient; embodied; and situated.

Design Principle 2: The 'principle of parallel, loosely coupled processes'.
Intelligence is emergent from agent-environment interaction through parallel, loosely coupled processes connected to the sensory-motor mechanisms.

Design Principle 3: The 'principle of sensory-motor co-ordination'.
All intelligent behaviour (e.g. perception, categorization, memory) is a result of sensory-motor co-ordination that structures the sensory input.

Design Principle 4: The 'principle of cheap designs'.
Designs are parsimonious and exploit the physics of the ecological niche.

Design Principle 5: The 'redundancy principle'.
Redundancy is incorporated into the agent's design with information overlap occurring across different sensory channels.

Design Principle 6: The 'principle of ecological balance'.
The complexity of the agent matches the complexity of the task environment. There must be a match in the complexity of sensors, motor system and neural substrate.

Design Principle 7: The 'value principle'.
The agent has a value system that relies on mechanisms of self-supervised learning and self-organisation.

These well-crafted principles have significant implications for the design of autonomous agents. To the most part, we will try to adhere to these principles when designing our own agents in these books. We will also be revisiting aspects of these principles several times throughout these books, where we will explore specific concepts such as emergence and self-organization in more depth.

However, we will slightly modify some aspects of these principles to more closely match the terminology and approach adopted in these books. Rather than make the distinction of three constituents as in the Design Meta-Principle and refer to an 'ecological niche', we will prefer to use just two: agents and environments. Environments are important for agents, as agent-environment interaction is necessary for complex agent behaviour. The next part of the book will explore what we mean by environments, and have a look at some environments that mirror the complexity of the real world.

Download free eBooks at bookboon.com

In presenting solutions to problems in these books, we will stick mostly to the design principles outlined above, but with the following further design principles:

Further design principles for the design of agents and environments in NetLogo for these books:

Design Principle 8:	The design should be simple, and concise (the 'Keep It Simple Stupid' or KISS principle).
Design Principle 9:	The design should be computationally efficient.
Design Principle 10:	The design should be able to model as wide a range of complex agent behaviour and complex environments as possible.

The main reason for making the design simple and concise is for pedagogical reasons. However, as we will see in latter chapters, simplicity in design does not necessarily preclude complexity of agent behaviour or complexity in the environment. For example, the NetLogo programming language has a rich set of models despite most of them being restricted to a simple 2D environment used for simulation and visualisation.

Click on the ad to read more

Download free eBooks at bookboon.com

1.6 Summary and Discussion

The quote at the beginning of this chapter relates to the time when humanity had yet to conquer the "final frontier" of space. Half a century of space exploration latter, perhaps we can consider that space is no longer the "final" frontier. We have many more frontiers to explore, although not of the physical kind as space is. These are frontiers in science and engineering, and frontiers of the mind. We can either choose to confront these challenging frontiers head on or ignore them by keeping our "heads in the sand".

This chapter provides an introduction to the field of Artificial Intelligence (AI), and positions AI as an emerging but potentially disruptive technology for the future. It makes an analogy between the study of AI and exploration of uncharted territory, and describes several paths that have been taken in the past for exploring that territory, some of them in conflict with each other. There have been many objections raised to Artificial Intelligence, many of which have been made from people who are ill-informed. This chapter also highlights the use of conceptual metaphor and analogy in natural language and AI.

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

- There are many paths to Artificial Intelligence. There are also many objections.
- The Turing Test is a contentious test for Artificial Intelligence.
- Searle's Chinese Room argument says a computer will never be able to think and understand like we do. AI researchers usually ignore this, and keep on building useful AI systems.
- Computers will most likely have human processing capabilities by 2020, but computers with intelligence will probably take longer.
- AI Winter – not at the moment.
- Conceptual metaphor and analogy – these are important linguistic devices we need to be aware of in order to understand natural language.
- Pfeifer and Scheier have proposed several important design principles for autonomous agents.

Download free eBooks at bookboon.com

2 Agents and Environments

Agents represent the most important new paradigm for software development since object-orientation. *McBurney et al.* (2004).

The environment that influences an agent's behavior can itself be influenced by the agent. We tend to think of the environment as what influences an agent but in this case the influence is bidirectional: the ant can alter its environment which in turn can alter the behavior of the ant. *Paul Grobstein* (2005).

The purpose of this chapter is to introduce agent-oriented systems, and highlight how agents are inextricably intertwined with the environment within which they are found. The chapter is organised as follows. Section 2.1 defines what agents are. Section 2.2 contrasts agent-oriented systems with object-oriented systems and Section 2.3 provides a taxonomy of agent-oriented systems. Section 2.4 lists desirable properties of agents. Section 2.5 defines what environments are and lists several of their attributes. Section 2.6 shows how environments can be considered to be *n*-dimensional spaces. Section 2.7 looks at what virtual environments are. And Section 2.8 highlights how we can use virtual environments to test out our AI systems.

2.1 What is an Agent?

Agent-oriented systems have developed into one of the most vibrant and important areas of computer science. Historically, one of the primary focus areas in AI has been on building intelligent systems. A standard textbook in AI written by Russell and Norvig (2002) adopts the concept of rational agents as central to their approach to AI. The emphasis is on developing agent systems "that can reasonably be called *intelligent*" (Russell & Norvig, 2003; page 32). Agent-oriented systems are also an important research area that underpins many other research areas in information technology. For example, the proposers of Agentlink III, which is a Network of Excellence for agent-based systems, state that agents underpin many aspects of the broader European research programme, and that "*agents represent the most important new paradigm for software development since object-orientation*" (McBurney et al., 2004).

Download free eBooks at bookboon.com

However, there is much confusion over what people mean by an "agent". Table 2.1 lists several perspectives for the meaning of the term 'agent'. From the AI perspective, a key idea is that an agent is embodied (i.e. situated) in an environment. Franklin and Graesser (1997) define an autonomous agent as "a system situated within and a part of an environment that senses that environment and acts on it, over time, in pursuit of its own agenda and so as to effect what it senses in the future". For example, a game-based agent is situated in a virtual game environment, whereas robotic agents are situated in a real (or possibly simulated) environment. The agent perceives the environment using sensors (either real or virtual) and acts upon it using actuators (again, either real or virtual).

The meaning of the term 'agent', however, can change emphasis when an alternative perspective is applied and this can lead to confusion. People will also often tend to use the definition they are familiar with from their own background and understanding. For example, distributed computing, Internet-based computing and simulation and modelling provide three further perspectives for defining what an 'agent' is. In the distributed computing sense, agents are autonomous software processes or threads, where the attributes of mobility and autonomy are important. In the Internet-based agents sense, the notion of agency is an over-riding criteria i.e. the agents are acting on behalf of someone (like a travel agent does when providing help in travel arrangements on our behalf when we do not have the expertise, or the inclination, or the time to do it ourselves). In simulation and modelling, an agent-based model (ABM) is a computational model whose purpose is to simulate the actions and interactions of autonomous individuals in a network or environment, thereby assessing their effects on the system as a whole.

Click on the ad to read more

Download free eBooks at bookboon.com

Perspective	Key Ideas	Some Application Areas
Artificial Intelligence	An agent is embodied (i.e. situated) in an environment and makes it own decisions. It perceives the environment through sensors and acts on the environment through actuators.	Intelligent Agents. Intelligent Systems. Robotics.
Distributed Computing	An agent is an autonomous software process or thread.	3-Tier model (using agents). Peer-to-peer networks. Parallel and Grid Computing.
Internet-based Computing	The agent performs a task on behalf of a user. i.e. The agent acts as a proxy; the user cannot perform (or chooses not to perform) the task themselves.	Web spiders and crawlers. Web scrapers. Information Gathering, Filtering and Retrieval.
Simulation and Modelling	An agent provides a model for simulating the actions and interactions of autonomous individuals in a network.	Game Theory. Complex Systems. Multi-agent systems. Evolutionary Programming.

Table 2.1 Various perspectives on the meaning of the term 'Agent'.

The term 'bot' – an abbreviation for ro*bot* – has become common as a substitute for the term 'agent'. In academic publications, the latter is usually preferred – for example, conversational agent rather than chatbot or chatterbot – although they are synonymous. A list of bots is shown in Table 2.2 named according to the task(s) they perform. The list is based on a longer list provided in Murch and Johnson (1999; pages 46–47).

Bot name(s)	Description
Chatterbots	Agents that are used for chatting on the Web.
Annoybots	Agents that are used to disrupt chat rooms and newsgroups.
Spambots	Agents that generate junk email ('spam') after collecting Web email addresses.
Mailbots	Agents that manage and filter e-mail (e.g. to remove spam).
Spiderbots	Agents that crawl the Web to scrape content into a database (e.g. Googlebot). For search engines (e.g. Google) this is then indexed in some manner.
Infobots	Agents that collect information. e.g. 'Newsbots' collect news; 'Hotbots' find the hottest or latest site for information; 'Jobbots' collect job information.
Knowbots or Knowledgebots	Agents that seek specific knowledge. e.g. 'Shopbots' locate the best prices; 'Musicbots' locate pieces of music, or audio files that contain music.

Table 2.2 Some bots and their applications.

Other names for agents and bots include: software agents, wizards, spiders, intelligent software robots, softbots and various further combinations of the words 'software', 'intelligent', 'bot' and 'agent'.

Download free eBooks at bookboon.com

Confusion can also arise because people will often adopt terms from other areas, and transfer the meaning into their own areas of interest. In the process, the original meaning of the term can often become changed or confused. For example, 'robot' is another term like 'agent' where the precise meaning is difficult to pin down. The term 'robot' is now being confused with the term 'bot' – many people now consider a 'robot' as not necessarily a physical machine, because included in their definition are such things as Web spiders (as used for search engines), and conversational bots (such as one might encounter these days when ringing up a helpline). Even more confusion arises because, for some people, both of these could also be considered to be agents.

What can also cause confusion with the use of the term agent is that it is often related to the concept of "agency", which itself can have multiple meanings. One meaning of the term agency is the capacity of an agent to act in a world – for humans, it is related to their ability to make their own choices which will then affect the world that they live in. This meaning is closely related to the meaning of agent that we adopt in these books. However, another meaning of agency is authorization to act on another's behalf – for example, a travel agency is authorized to act on behalf of its customers to find the most competitive travel options.

Merriam-Webster's Online Dictionary lists the following meanings for the term agent:

1. *One that acts or exerts power;*
2. *a: something that produces or is capable of producing an effect : an active or efficient cause*
   b: a chemically, physically, or biologically active principle;
3. *a means or instrument by which a guiding intelligence achieves a result;*
4. *one who is authorized to act for or in the place of another: as*
   a: a representative, emissary, or official of a government <crown agent> <federal agent>
   b: one engaged in undercover activities (as espionage) : spy <secret agent>
   c: a business representative (as of an athlete or entertainer) <a theatrical agent>;
5. *a computer application designed to automate certain tasks (such as gathering information online).*

The fourth meaning of agent relates to the meaning of agency often used in general English, such as used in the common phrases 'insurance agent', 'modeling agent', 'advertising agent', 'secret agent', and 'sports agent'. (See Murch and Johnson (1999; page 6) for a longer list of such phrases). This can cause the most confusion as it differs with the meaning of agent adopted in these books (which is more related to the fifth meaning).

Download free eBooks at bookboon.com

All of these similar, but slightly different, meanings spring from the underlying concept of an 'agent'. This is perhaps best understood by noting that an agent or agent-oriented system is analogous to a human in real life. Considering this analogy, we can make comparisons between the agent-oriented systems we design with attributes of people in real life. People make their own decisions, and exist within, interact with, and effect the environment that surrounds them. Similarly, the goal of agent designers is to endow their agent-oriented systems with similar decision-making capabilities and similar capacity for interacting and effecting their environment. In this light, the different meanings listed in the dictionary definition are related to each other by the underlying analogy of an entity that has the ability to act for itself, or on behalf of another, or with the ability to produce an effect, with some of the capabilities of a human. The agent exists (is situated) within an environment and is able to sense, move around and affect that environment, by making its own decisions so as to affect future events. The agent is analogous to a person in real life having some of a person's abilities.

Click on the ad to read more

Download free eBooks at bookboon.com

2.2 Agent-oriented Design Versus Object-oriented Design

How does agent-oriented design differ from object-oriented design? To answer this question, first we must explore what it means for a system design to be object-oriented. Object-oriented programming (OOP) is now the mainstream programming paradigm supported by most programming languages. An 'object' is a software entity that is an abstraction of a person, place or thing in the real world. Objects are usually associated with nouns that appear in the system requirements and are generally defined using a *class*. The purpose of the class is to encapsulate all the data and routines (called '*methods*') together in one place. An object consists of: *identity*, which allows the object to be uniquely identified – for example, attributes such as name, date of birth, place of birth can uniquely identify a person; *states*, such as 'door = open' or 'switch = on'; and *behaviour*, such as 'send message' or 'open door' (these are associated with the verbs + nouns in the system requirements).

What properties does a system need for it to be object-oriented? Some definitions state that only the properties *abstraction* and *encapsulation* are needed. Other definitions state that further properties are also required: *inheritance*, *polymorphism*, *dynamic binding* and *persistence*. (See Table 2.3).

Property	Description
Abstraction	Software objects are virtual representations of real world objects. For example, a class *HumanClass* might be an abstraction of humans in the real world. We can think of the class as defining an analogous relationship between itself and with humans in the real world.
Encapsulation	Objects encompass all the data and methods associated with the class, and access to these is allowed only through a strictly enforced interface that defines what is visible to other classes, with the rest remaining hidden (called 'information hiding'). For example, the class *HumanClass* might have a *talk*() method, the code for which defines exactly what and how the talking is achieved. However, anyone wanting to execute the *talk*() method are not interested in how the talking is achieved.
Inheritance	The developer is able to define subclasses that are specialisations of parent classes. Subclasses inherit the attributes and behaviour of their parent classes, but have additional functionality. For example, *HumanClass* inherits properties of its parent *MammalClass* which in turn inherits properties from its parent *AnimalClass*.
Polymorphism	This literally means "many forms". A method with the same name defined by the parent class can take different forms during execution depending on its subclass definition. For example, *MammalClass* might have a *talk()* method – this will execute very different routines for an object that belongs to the *HumanClass* compared to an objects belonging to the *DogClass* or the *LambClass* (the former might start chatting, whereas the latter might start barking or bleating).
Dynamic Binding	This determines which method is invoked at runtime. For example, if *d* is an object of *DogClass*, then the corresponding method to its actual class will be invoked at runtime, when *d.talk()* is executed. (Barking will be produced instead of chatting or bleating).
Persistence	Objects and classes of objects remain until they are explicitly deleted, even after they have finished execution.

Table 2.3 Properties that define object-oriented design.

Download free eBooks at bookboon.com

Figure 2.1 illustrates how objects are abstractions to entities in real life. Three objects are depicted in the diagram – `Bill` who is an instance of the `HumanClass`, `Tooty` who is an instance of the `DogClass` and `Timothy` who is an instance of the `LambClass`. (Objects are also called 'instances' of a particular class).

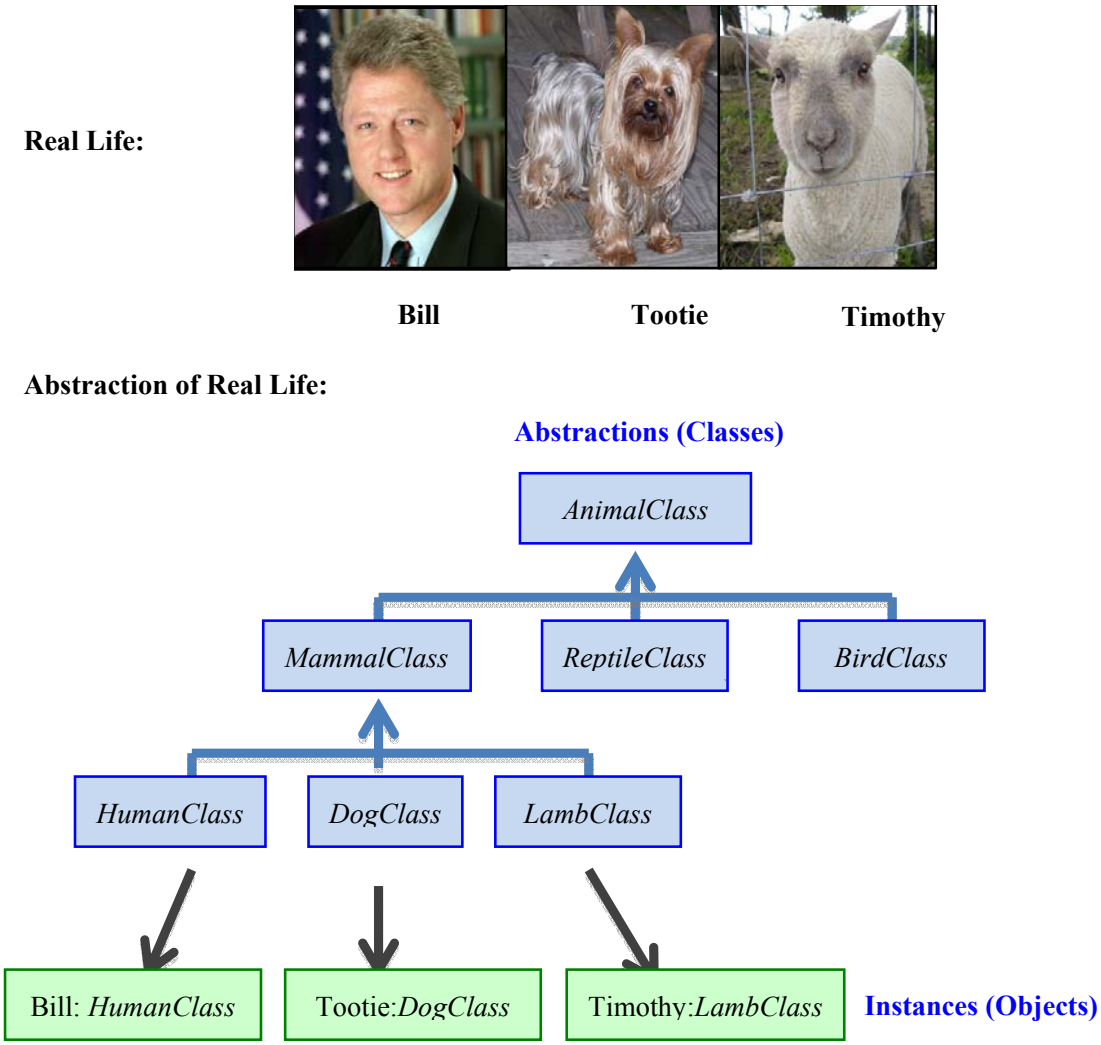


Figure 2.1 Object-oriented design: How objects are abstractions of entities in real life.

How do agents differ from objects? Wooldridge (2002; pages 25–27) provides the following answer:

- Agents have a stronger degree of autonomy than objects.
- Objects have no control over when they are executed whereas agents decide for themselves whether to perform some action. In other words, objects *invoke* other objects' methods, whereas agents *request* other agents to perform some action.
- Agents are capable of flexible (reactive, proactive, social) behaviour whereas objects do not specify such types of behaviour.
- A multi-agent system is inherently multi-threaded – each agent is assumed to have at least one thread of control.

Download free eBooks at bookboon.com

We can look at two sample tasks to further illustrate the distinction between agents and objects: task 1, cleaning the kitchen floor; and task 2, washing clothes. What are the agent-oriented versus object-oriented solutions to these two tasks? The short answer is that no completely object-oriented solution exists for either task, as we need to use an agent to get the tasks done.

For task 1, the person cleaning the floor can be considered to be the agent. For an object-oriented solution, this person can be considered as analogous to the software developer – he will make the decisions about when to start cleaning, and pick the appropriate tools (objects), and how to use them. He will choose for himself the most appropriate settings – these correspond to the parameters the developer chooses when writing the code (such as state, methods, and arguments passed to methods). Some example objects are: a broom – its direction of use, velocity and frequency of sweeps; a bucket – how much water, its temperature, the size of the bucket; or a vacuum cleaner – the power setting, the carpet or hard floor setting, its direction of use, and so on. In contrast, one possible agent-oriented solution is to have a robot do the task – for example, a robotic vacuum cleaner (see picture on the right). In contrast to the object-oriented solution, the robotic agent is doing the task itself, not someone else. It decides for itself where, when and how to do it. At the risk of anthropomorphising what is obviously a machine, we can take the perspective of the robot itself as it is making its decisions, such as: "*I* don't need to do it now – *I*'ll do it tomorrow"; "*I*'m running out of power – *I* better re-charge *myself*"; "The floor is a bit dirty – *I* better wet it"; "*I*'m stuck – *I* better ask for help".

For task 2, the object-oriented solution is for the human agent to use a washing machine to wash the clothes. The washing machine has many settings that the human can select. In most cases, he will literally not know how the washing machine works – its workings are hidden from him. Once the start button has been pressed (i.e. analogous to the program code starting executing), there is very little control over what happens after that. The washing machine object has methods – for example, fast cycle, spin cycle and so on. It also has state – for example, the time to finish, the temperature and so on. In contrast, human agents are the only agent-oriented solution presently available for this problem. In the future, a domestic robot might perform this task for humans. In this case, from its perspective, it might make the following decisions: "*I* will now do the washing for you"; "*I* will fetch the dirty clothes *myself*"; "*I* will now recharge *myself*".

Download free eBooks at bookboon.com

Note that the uses of the words "*I*" and "*myself*" have been highlighted in italics font for the two tasks above. This is to emphasize the first-person design perspective of the agent – it makes decisions from its own personal viewpoint. Objects, in contrast, are designed from a third-person perspective, being invoked externally – internally, there is no concept of "self".

Object-oriented programming and design is the predominant software engineering paradigm; few mainstream programming languages currently support agented-oriented programming. Python, for example, is said to be multi-paradigm, but it does not support agent-oriented programming. Presently, a developer is forced to resort to a hybrid design – using agent-oriented frameworks implemented on an object-oriented platform. Various agent frameworks are discussed in the next chapter. In the meantime, we will look at different kinds of agents, the properties they can have, and the kinds of environments that they can exist in.

2.3 A Taxonomy of Autonomous Agents

A common method used in scientific investigation is to classify concepts into a taxonomy. Often they are useful in providing structure to help organise a topic of discussion. We will see in Chapter 9 that such an exercise is fraught with difficulties, however, with the distinction between parent and child concepts, and between sibling concepts unclear. Many examples fall into multiple categories, or exist between the boundary of two related concepts. Therefore, taxonomical classification should be used with care.

Click on the ad to read more

Download free eBooks at bookboon.com

Figure 2.2 displays a taxonomy of autonomous agents based on the taxonomy proposed by Franklin and Graesser (1998). Noting the comments above about the limitations of taxonomical classification, it should be stressed that this is not a definitive or exhaustive taxonomy. The bottom row in Figure 2.1 differs from Franklin and Graesser's taxonomy that consists of just three sub-categories under the 'Software Agents' category – these are 'Task-specific Agents', 'Entertainment Agents' and 'Viruses'. The latter is not covered in these books – although computer viruses are a form of agent, they are not benevolent (i.e. usually harmful) and are better covered as a separate subject. 'Task-specific Agents' has been expanded to more closely examine individual tasks such as human language processing, and information gathering. Note that the last row is by no means exhaustive – further categories of agents include virtual humans, human agent interaction and mobile and ubiquitous agents.

Another agent category often considered is 'Intelligent Agents'. Unfortunately, this term is often misused. Often a system is labelled as being an intelligent agent with little justification as to why it is 'intelligent'. Chapter 10 will highlight the philosophical pitfalls in trying to define what intelligence is, so rather than being overly presumptuous in our taxonomical classification, we will instead separate agents by the primary task they perform as in Figure 2.2. Evaluation can then involve measuring how well the agent performs at the task for which it is designed. Whether that performance is sufficient for the agent then to be classed as being 'intelligent' is up to the observer who is watching the task being performed.

We will now explore each of the types of agents listed in Figure 2.2 a bit further to help clarify the definition. 'Real Life Agents' means animals that are alive such as mammals, reptiles, fish, birds, insects and so on. Franklin and Graesser used the term 'Biological Agents' instead for this category, but this could be confused with the biological agents that are toxins, infectious diseases (such as real-life viruses; for example, Dengue Fever and Ebola) and bacteria (such as Anthrax and the Plague). 'Artificial Life Agents' are agents that create an artificial life form or simulate a real life entity. Robotic agents of the mechanical kind (rather than software robots) are also agents from the AI perspective – for example, the robot rovers used for the Mars Rover missions, such as Spirit and Opportunity, act as "agents" for NASA on Mars, but also have some degree of autonomy to act by themselves. 'Software Agents' cover agents that exist purely in a virtual or software-based environment. These can be classified into many different categories – for example, agents that process human language, agents that gather information, agents that are knowledgeable in some way, agents that learn, and agents designed for entertainment purposes such as used in computer gaming and for special effects in movies.

Download free eBooks at bookboon.com

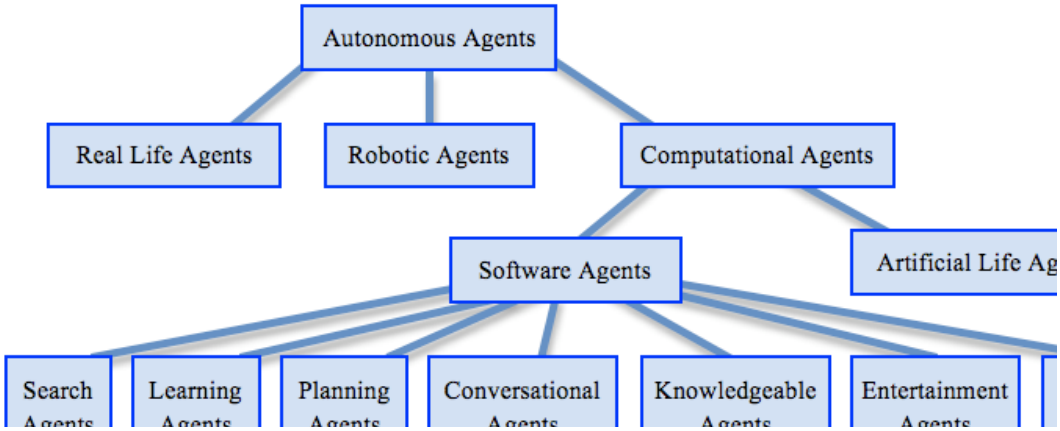


Figure 2.2: A Taxonomy of Autonomous Agents (based on Franklin and Graesser, 1997).

'Human Agents' naturally fall into the 'Real Life Agents' category. Murch and Johnson (1999) point out that currently humans are the agents that are the finest at performing most complex tasks in the world, and will continue to be so for quite a while. Humans agents with specialist skills (such as a travel agent, or an agent for a football player or movie star) provide a service on behalf of other humans who would not be able to get that service any other way, or who do not have the time or skills to do it themselves. They have the contacts to provide that service, have access to relevant information, and often they can provide that service at a fraction of the cost. However, humans are limited by the number of hours they can work in the week; with 12-hour days, they can only work a maximum of 84 hours in the week and at that rate they would burn out quickly! Therefore, there is an opportunity for computer-based agents to help us overcome these limitations.

In attempting to classify what an agent is, we can also ask the opposite question – "What are *not* agents?" Nwana (1996) noted that Minsky in his book *Society of the Mind* used the term to formulate his theory of human intelligence:

"…to explain the mind, we have to show how minds are built from mindless stuff, from parts that are much smaller and simpler than anything we'd consider smart… But what could those simpler particles be – the 'agents' that compose our minds? This is the subject of our book…" (Minsky, 1985; page 18).

Nwana defines agents in such a way that Minsky's notion of an agent does not satisfy her criteria. She uses three minimal characteristics to derive four types of agents based on the typology shown in Figure 2.3: *collaborative agents, collaborative learning agents, interface agents* and truly *smart agents*. She latter expands this list to include three further types: *Information/Internet agents*, *reactive agents*, and *hybrid agents*.

Download free eBooks at bookboon.com

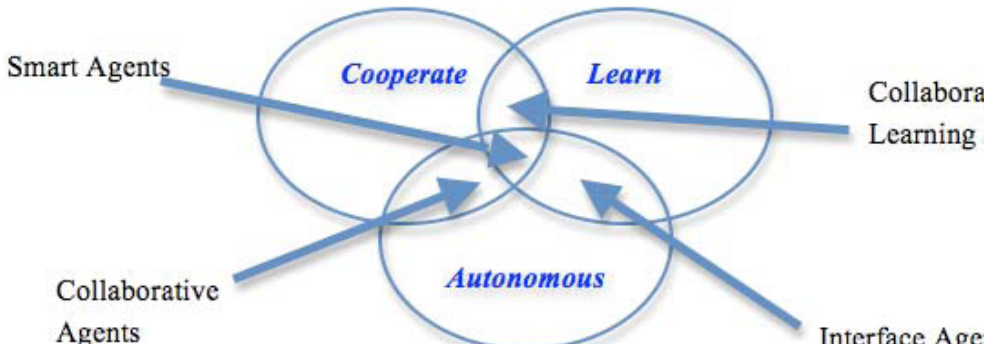


Figure 2.3 An Agent Topology (Nwana, 1996).

Her definition considers that agents operate more at the knowledge level rather than the symbol level, and require 'high-level messaging' [her words] (as opposed to 'low-level messaging' used in distributed systems). Therefore Minsky's agents, expert systems, most knowledge-based system applications, and modules in distributed computing applications do not qualify. Neither would turtle agents used in the programming language NetLogo (see Chapter 3 and subsequent chapters), a language that was designed after her publication.

INNOVATIVE
LIKE YOU.

If you're hoping for a truly modern education, one where you're encouraged to speak your mind and to think long-term, both when it comes to your own future and the future of the planet. Then the University of Gothenburg is the place for you.

Study a Master's programme in Gothenburg, Sweden | www.gu.se/education

UNIVERSITY OF GOTHENBURG

Click on the ad to read more

Download free eBooks at bookboon.com

The term *proto-agent* (North and Macal, 2007) is often used in agent modelling and simulation to cover 'lower-level' agents such as turtle agents in NetLogo to distinguish them from agents that adhere to stronger definitions such as Nwana's. North and Macal define a proto-agent as an entity used in modelling and simulation that maintains a set of properties and behaviours that need not exhibit learning behaviour. If they gain learning behaviour, they become agents.

For the purposes of these books, rather than making an arbitrary distinction between a proto-agent and agent, we consider all the examples above as having some degree of agent-hood as defined in Section 2.1. We therefore will use the term agent throughout rather than proto-agent, since in reality no agent-oriented system currently exists that has yet achieved the full set of properties as per Nwana's definition, or the set of desirable properties as described in more detail in the next section.

2.4 Desirable Properties of Agents

The concept of an agent can be defined by listing the desirable properties that we wish the agents to exhibit (see Tables 2.4 to 2.6). Russell & Norvig (2004) identified the first four properties in Table 2.4 as key attributes of an agent from the AI perspective: *autonomy* (acting on one's own behalf without intervention); *reactivity* (reacting to stimuli); *proactivity* (being proactive); and *social-ability* (able to communicate in some manner with other agents). Autonomy, in particular, is an important key attribute – in fact, the term 'autonomous agents' is often used in the literature as a synonym for agent-oriented systems to emphasize this point. The six properties in Table 2.4 are often designated as belonging to a *weak* agent, adding the ability to set goals and temporal continuity as two further key attributes (Wooldridge and Jennings (1995)). The properties in Table 2.5 are associated with a *strong* notion of an agent as they are properties usually applied to humans (Wooldridge and Jennings (1995); Etzioni and Weld (1995)). Taskin *et al.* (2006) list three further properties in Table 2.6 that are combinations of the basic properties: coordination, cooperative ability and planning ability.

Property	Description
Autonomy	The agent exercises control over its own actions; it runs asynchronously.
Reactivity	The agent responds in a timely fashion to changes in the environment and decides for itself when to act.
Proactivity	The agent responds in the best possible way to possible future actions that are anticipated to happen.
Social ability (Ability to communicate)	The agent has the ability to communicate in a complex manner with other agents, including people, in order to obtain information or elicit help in achieving its goals.
Ability to set goals	The agent has a purpose.
Temporal continuity	The agent is a continually running process.

Table 2.4 Properties associated with the weak notion of an agent. Based on Russell and Norvig (2004) and Wooldridge and Jennings (1995).

Download free eBooks at bookboon.com

Property	Description
Mobility	The agent is able to transport itself around its environment.
Adaptivity	The agent has the ability to learn. It is able to change its behaviour based on the basis of its previous experience.
Benevolence	The agent performs its actions for the benefit of others.
Rationality	The agent makes rational, informed decisions.
Collaborative ability	The agent collaborates with other agents or humans to perform its tasks.
Flexibility	The agent is able to dynamically respond to the external state of the environment by choosing its own actions.
Personality	The agent has a well-defined, believable personality and emotional state.
Cognitive ability	The agent is able to explicitly reason about its own intentions or the state and plans of other agents.
Versatility	The agent is able to have multiple goals at the same time.
Veracity	The agent will not knowingly communicate false information.
Persistency	The agent will continue steadfastly in pursuit of any plan.

Table 2.5 Properties associated with the strong notion of an agent. Based on Wooldridge and Jennings (1995) and Etzioni and Weld (1995).

Property	Description
Coordination	The agent has the ability to manage resources when they need to be distributed or synchronised.
Cooperation	The agent makes use of interaction protocols beyond simple dialogues, for example negotiations on finding a common position, solving conflicts or distributing tasks
Ability to plan	The agent has the ability to pro-actively plan and coordinating its reactive behavior in the presence of the dynamical environment formed by the other acting agents.

Table 2.6 Further properties associated with an agent. Based on Taskin et al. (2006).

These definitions are interesting from a philosophical point of view, but their meaning is often vague and imprecise. For example, if one were to attempt to classify existing agent-based systems using these labels, one would find the task is fraught with difficulties and inconsistencies.

Download free eBooks at bookboon.com

A simple exercise in the application of these properties to classifying examples of agent-oriented systems will demonstrate some of the shortcomings of such a classification system. For example, Googlebot, the Web crawler used by Google to construct its index of the Web, has the properties autonomy, reactivity, temporal continuity, mobility and benevolence, but whether it exhibits the other properties is unclear – for example, it does not have the rationality property (the informed decisions it makes are not its own). Therefore it exhibits both weak and strong properties, but could be construed to be neither. A chatbot on the other hand exhibits all of these properties in various strengths. Perhaps the strangest of the properties is benevolence. It is not clear why this is a necessary property of an agent-oriented system – computer viruses are clearly not benevolent; and interaction between competing multiple agents may also not be benevolent (such as the wolf-sheep predation model that comes with the NetLogo Models Library described in Chapter 4), but can lead to stability in the overall system. Also, underlying many of these properties is the implicit assumption that the agent has some degree of consciousness – for example, that it *consciously* makes rational decisions, that it *consciously* sets goals and make plans to achieve them, and that it does not *consciously* communicate false information and so on. Therefore, it may not be possible to build a computational agent with these properties without first having the capabilities that a human agent with full consciousness has.

MASTER OF SCIENCE PROGRAMMES

"The faculty at BI are always ready to guide and motivate students to think outside the box. The opportunity to interact with students from all over the globe contributes to global learning and thinking. BI's infrastructure caters to every student's perceivable need – the state-of-art canteen facilities, gymnasium, study rooms, IT facilities and more".

Althea Pereira, India, MSc in Leadership and Organisational Psychology, 2009

BI Norwegian School of Management (BI) offers a wide range of Master of Science (MSc) programmes in:

- Business & Economics
- Strategic Marketing Management
- International Marketing & Management
- Leadership & Organisational Psychology
- Political Economy
- Financial Economics
- Innovation & Entrepreneurship

For more information, visit www.bi.edu/msc
BI also offers programmes at Bachelor, Masters, Executive MBA and PhD levels.
Visit www.bi.edu for more information.

APPLY NOW

BI NORWEGIAN BUSINESS SCHOOL

EFMD EQUIS ACCREDITED

Click on the ad to read more

Download free eBooks at bookboon.com

The other failing is that these are qualitative attributes, rather than quantitative. An engineer would prefer to have attributes that were defined more precisely – for example, what does it mean for an agent to be rational? However, the classification does have merit in the sense that it highlights some of the attributes that we may wish to design into our systems. For example, we can use properties 1 to 3 as a starting point to suggest some minimal design principles that a system must adhere to before it can be deemed to be an agent oriented system as follows:

An agent-oriented system should adhere to the following agent design objectives – it is autonomous; it is reactive; it is proactive (at least):

Design Principle 2.1: An agent-oriented system should be autonomous.
Design Principle 2.2: An agent-oriented system should be reactive.
Design Principle 2.3: An agent-oriented system should be proactive.

However, rather than going further and suggest a list of ill-defined properties as defining degrees of intelligence (i.e. whether weak or strong), these books adopts a different design-oriented approach. In Chapter 10, various desirable design objectives are described to provide the "*strongest*" notion of an agent: knowledgeability, intelligence, rationality, self-awareness, consciousness and thoughtfulness (i.e. an agent that thinks as we do). Rather than say an AI must have these properties for it to be deemed 'intelligent', we instead propose several design objectives – properties that we as designers wish our system to have. We maintain that for an agent to think, it must first have knowledge of the environment it finds itself in as well as knowledge of how to act within it to maintain its competitive edge (in terms of fitness to survive compared to other agents). An agent must also be intelligent i.e. be able to understand the meaning of its knowledge, be able to make further inferences to add to its knowledge, and to act in an 'intelligent' manner in order to react to whatever is happening in its environment or whatever is likely to happen (again in order to maintain or improve its fitness). Self-awareness, consciousness and thoughtfulness correspond to the human traits we are all familiar with, but there is a lack of real understanding of how they happen, or of how we might go about developing artificial systems that have these properties. The intervening chapters will set the scene as an explanation for this design-based perspective on AI.

The remaining part of this chapter will look at what environments are, and highlight their importance for the design of AI systems.

2.5 What is an Environment?

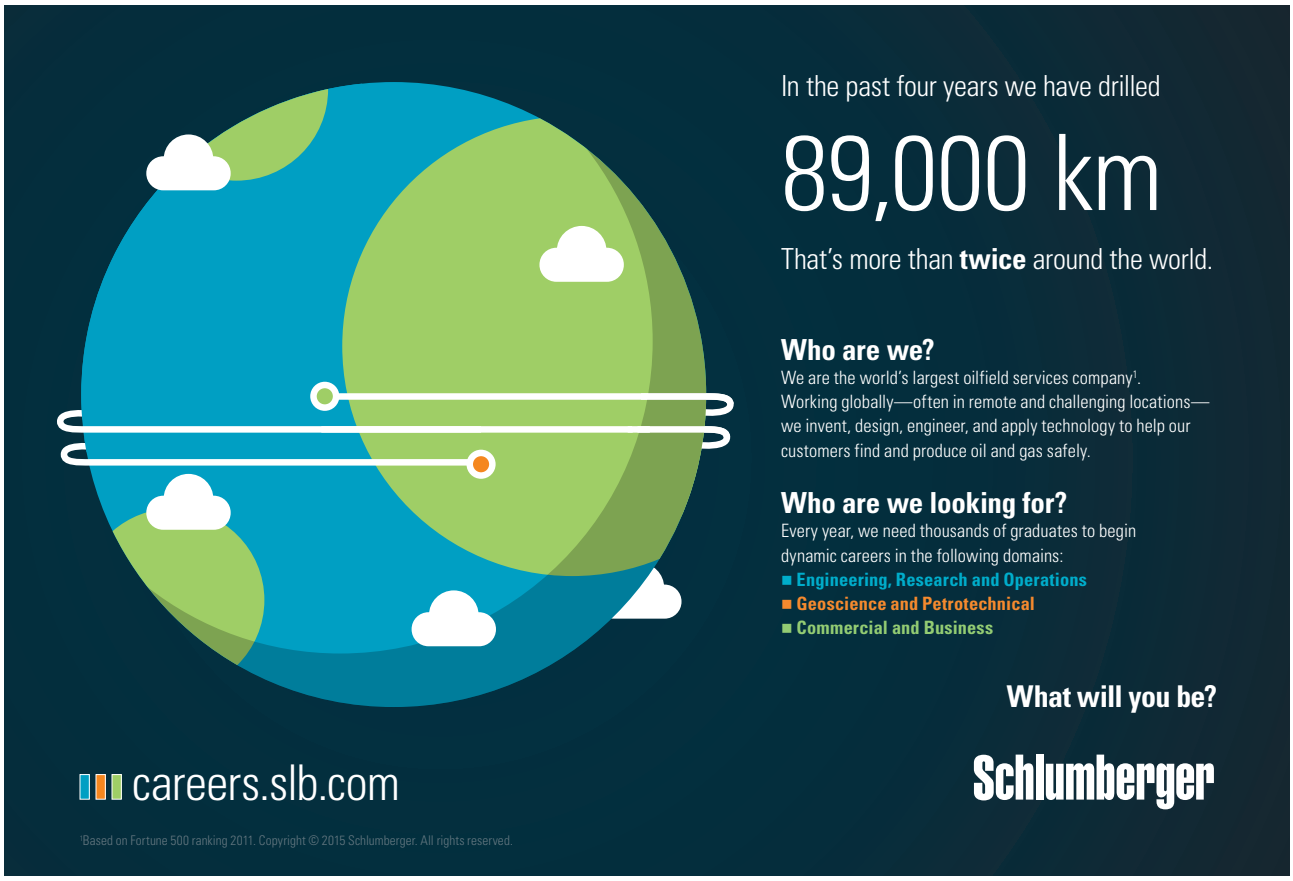
We can think of the environment as being everything that surrounds the agent, but which is distinct from the agent and its behaviour. An environment is everything in the world that surrounds the agent that is not part of the agent itself. This is where the agent 'lives' or operates, and provides the agent with something to sense and somewhere for it to move around. An environment is analogous to the world in real life having some of its properties.

Download free eBooks at bookboon.com

An environment is not the same as the term 'ecological niche' which is used to describe how an organism or population responds to the distribution of resources and competitors and depends not only on where the organism lives and what surrounds it but also on what it does. Odum (1959) uses the analogy that the habitat of an organism is its 'address' or location, whereas the niche is its 'profession'. For example, an oak tree might have oak woodlands as its habitat – the address might be "Oak Tree, New Forest" whereas what the oak tree does, and how it makes a living, by responding to distribution of resources and competitors, is its niche.

We will prefer to use the term 'environment' instead, as this more closely matches concepts that are familiar in computer science (such as the term 'virtual environment' – see below). In addition, the distinction between an agent and its ecological niche is related to its behaviour, with the two being intertwined – that is, the niche within which an agent is found dictates it's behaviour, and it's behavior to some extent determines its niche. On the other hand, an environment is clearly distinguishable from the agent as being everything in the immediate world or habitat of the agent that is not part of the agent itself. We also would like to adopt a first-person design perspective, to describe the behaviour of the agent directly based on the point of view of the agent itself, as opposed to a third-person perspective of an observer. In other words, we wish to design behaviour as a function of the agent alone as it interacts with its environment (which might include other agents), rather than have to design it in relation to its niche.

An environment can have various attributes from the point of view of the agent ('Intelligent Agents', 2008). These are listed in order of increasing complexity in Table 2.7.



Click on the ad to read more

Download free eBooks at bookboon.com

Attributes	Description
Observable and partially observable.	An agent can be considered to be an agent only if it has the ability to observe its environment (and conversely, the environment itself must therefore be observable). In some cases, usually simple environments, or software-generated environments, all of the environment may be observable. Usually, however, the environment may only be partially observable.
Deterministic, stochastic and strategic.	A fully deterministic environment is one where any future state of the environment can be completely determined from a preceding state and the actions of the agent. An environment is stochastic if there is some element of uncertainty or outside influence involved. Note that if a deterministic environment is only partially observable to the agent, it will appear to be stochastic from the agent's point of view. A strategic environment is fully determined by the preceding state combined with the actions of multiple agents.
Episodic and sequential.	The task environment is episodic if each of the agent's tasks do not rely on past performance, or cannot affect future performance. If not, then it is sequential.
Static and dynamic.	A static environment does not change. In a dynamic environment, if an agent does not respond to the change, this is considered as a choice to do nothing.
Discrete and continuous.	A discrete environment has a finite number of possible states, whereas the number of states in a continuous environment is infinite.
Single-agent and multiple-agent.	A multiple-agent environment the agent that acts cooperatively or competitively with another agent. If this is not the case, then from the perspective of the agent, the other agents can be viewed as part of the environment that is behaving stochastically.

Table 2.7 Attributes of environments (based on Wikipedia entry for 'Intelligent Agents').

The analogy of an environment being like the world we live in is often implicitly used when the term 'environment' is used in computer science and AI in particular. An agent can explore, get lost in, and map a virtual computer environment just the same as a human in the real world environment – the ability to observe/sense and move around the environment are key properties of both.

However, virtual computer environments do not have to be limited to a geographical interpretation where physical location is associated with a specific position in the environment. The physics of the virtual environment can be altered to suit the designer's purposes. For example, it is possible for an avatar (the computer-generated agent that represents the human operator) to fly around in virtual environments such as Second Life, and in some computer games, avatars and bots are able to teleport, both of which are impossible for humans in real life. Alternatively, the environment might be a representation of something that may not be easily or conveniently represented geographically such as a computer network, the Internet or even the London Underground network. The environment may also be a simulation of a real environment, where the goal is to simulate specifically chosen real physical properties as closely as possible. A problem with simulated environments, however, is that it is often difficult to achieve realism in the simulation, as the simulation may diverge from reality in unpredictable ways.

Download free eBooks at bookboon.com

2.6 Environments as *n*-dimensional spaces

There are many cases where we may wish to represent an environment which has no geographical equivalent at all, such as a collection of text documents in an Information Retrieval system, or data in a database, in which case, an abstract representation of the environment is required. In this case, the textual documents, or a row in a database table, can be represented by tuples corresponding to points in an *n*-dimensional space, each dimension representing one particular attribute (where *n* represents the total number of attributes used in all the tuples that are plotted in the space).

For example, the following two tuples, A and B, can be represented in a 3-dimensional space as shown in Figure 2.4, since there are three attributes – game, player and height:

Tuple A: (game: "Rugby", player: "Jonah Lomu", height: 196)
Tuple B: (game: "Rugby", player: "David Kirk", height: 173)

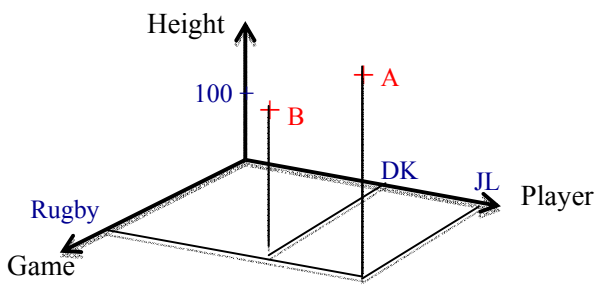


Figure 2.4: A three dimensional (3D) environment containing Tuples A and B represented as points.

On the "Player" axis, "DK" stands for the location representing "David Kirk"; similarly, "JL" stands for "Jonah Lomu". On the "Game" axis, "Rugby" stands for the location representing the game of rugby. The "Height" axis is a continuous dimension representing the height in centimeters ≥ 0. The other two dimensions are discrete. The two points representing the tuples A and B are shown by red crosses.

An *n*-dimensional space, as illustrated by this simple example, can clearly be considered as an environment: there is a space that can be explored and mapped, where objects such as Tuples A and B can have locations; movement from one location to another can have meaning and be plotted using a path; and locations can be considered to be near or far away from each other. In fact, *all* environments can be readily represented as an *n*-dimensional space – we move around in a real world environment that can be represented in 3D – length, height and width. Further dimensions are required to describe the attributes of objects and agents to be found at locations in the 3D real world environment – but essentially, these attributes can simply be considered as further dimensions in an *n*-dimensional space.

Download free eBooks at bookboon.com

As a further example, we can look at a larger set of data such as that listed in Table 2.8. The table lists a set of tuples containing information about New Zealand rugby players, called All Blacks. The data is 5-dimensional as each tuple (a row in the table) contains five attributes about each player – the position they played, their name, their height in cm, their weight in kg, and the year that they first played for the All Blacks.

Position	Name	Height (cm)	Weight (kg)	Debut year
Wing	Grant Batty	165	70	1972
Wing	Doug Howlett	185	93	2000
Wing	John Kirwan	191	97	1984
Wing	Jonah Lomu	196	119	1994
Wing	Joe Rokocoko	189	98	2003
Half-back	Sid Going	170	81	1967
Half-back	David Kirk	173	73	1983
Half-back	Justin Marshall	179	95	1995
Half-back	Piri Weepu	178	94	2004
Prop	Richard Loe	188	116	1986
Prop	Jamie Mackintosh	193	130	2008
Prop	Kees Meeuws	183	121	1998
Prop	Neemiah Tialata	187	127	2005
Lock	Mark Cooksley	205	125	1992
Lock	Andy Haden	199	112	1977
Lock	Chris Jack	202	115	2001
Lock	Ian Jones	198	104	1990

Table 2.8 An example of 5-dimensional data: Some New Zealand All Blacks rugby players.

Visualising n-dimensional data can often be difficult even when the number of dimensions is relatively small – most real-life data, however, is usually highly multi-dimensional. If we use a Cartesian co-ordinate system, then one solution we can resort to is to use multiple plots (as shown in the left plots of Figure 2.5 below). An alternative solution is to use a parallel co-ordinate system where the axes are placed parallel to each other, and a single tuple is plotted as a polyline that connects the points on each axis that represent each attribute value. Therefore, the tuple in this environment is represented as a line rather than as a single point (as shown in the right plot of Figure 2.5).

Download free eBooks at bookboon.com

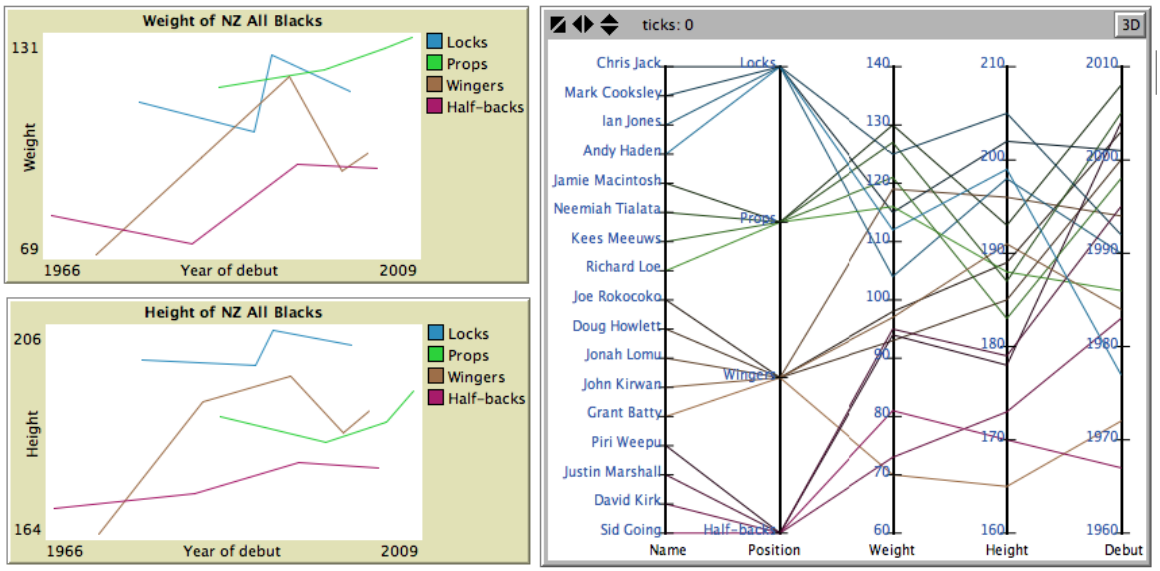
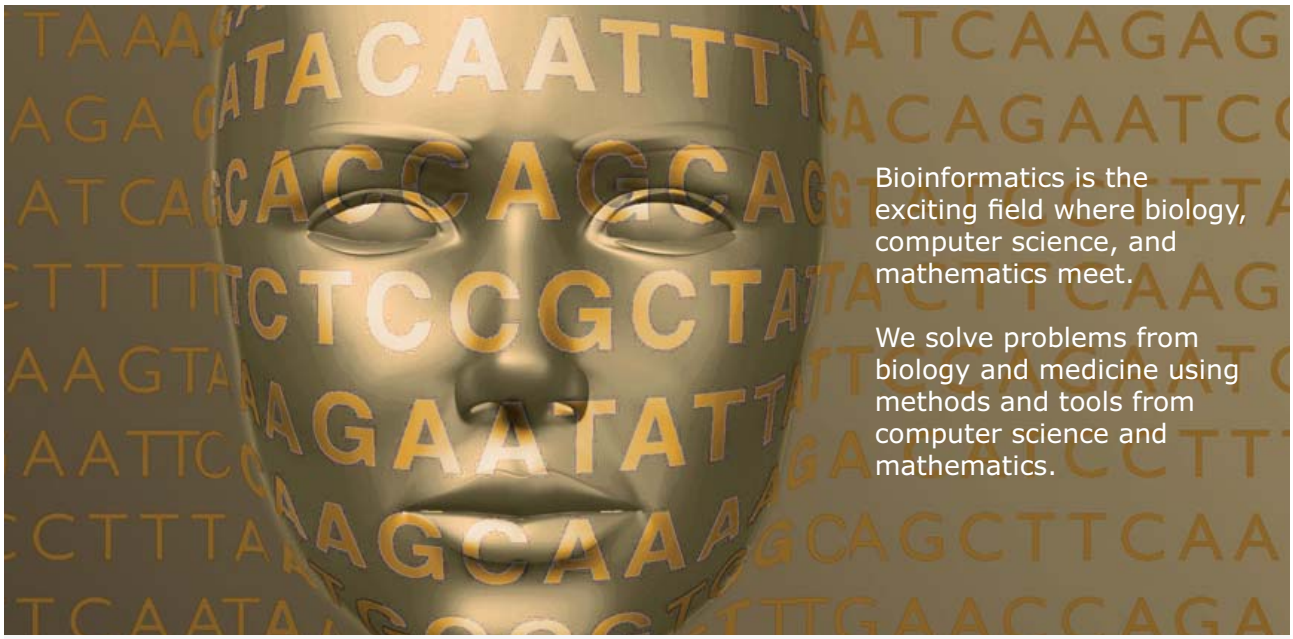


Figure 2.5 The data in Table 2.8 plotted using Cartesian co-ordinates (left plots) and parallel co-ordinates (right plot).

UPPSALA UNIVERSITET

Develop the tools we need for Life Science
Masters Degree in Bioinformatics



Read more about this and our other international masters degree programmes at www.uu.se/master

Click on the ad to read more

Download free eBooks at bookboon.com

Both co-ordinate systems for visualizing *n*-dimensional data have their advantages and disadvantages as is apparent in the plots shown in Figure 2.5. For example, all the plots are useful for showing tendencies and highlighting properties of the data such as All Black half-backs tending to be shorter than locks. However, the Cartesian co-ordinate plots tend to be limited in the amount of information that can be conveyed (adding names in the left plots, for example, would add significant clutter). Similarly, the parallel co-ordinate plot on the right is constrained to the particular order that the parallel axes are chosen, and it therefore may be more difficult to spot a particular trend if two particular axes are far apart. It does have the advantage though of making it easier to include data for all dimensions on the same plot.

The plots shown in Figure 2.5 were created using NetLogo. This language will be described in more detail latter in the book, starting in Chapter 3. A set of exercises that look at the code to create these plots can be found in the accompanying text books *Exercises for Artificial Intelligence*.

The purpose of showing these plots is to highlight the link between data, information and environments, and also highlight that there are different ways of visualizing information. The plots are made up of a series of points and paths, and can be considered to be maps that are representing aspects of the real world environment in some manner (like a topographical map attempts to represent the real world such as one might see in a car navigation system, or use when hiking or orienteering). Points, paths and their relationships (such as nearness and farness) and maps, are important concepts for environments and their visualizations and will be used as fundamental design patterns in code samples elsewhere in these books and can be considered as analogous to their topographical equivalents in real life. We can define a point as a location in an environment or an *n*-dimensional space. A path can be defined as one possible way an agent can move between two or more points. A map is a schematic representation of an environment or an *n*-dimensional space.

These will form the basic building blocks that we will use in subsequent chapters to demonstrate various aspects of agent-oriented design for AI. We can also consider points, paths and maps as analogous to the same terms used in mathematical topology.

2.7 Virtual Environments

Virtual environments are immersive online worlds that people using avatars visit and modify, or they can be computer-generated worlds such as those depicted in computer-games, or in simulations (for example, for medical or military simulations). Other names include 'virtual worlds', 'collaborative virtual environments' (CVEs), 'multi-user virtual environments' (MUVEs), and 'massively multi-player online game' (MMOGs).

Download free eBooks at bookboon.com

A 'virtual tour' is a computer-generated tour that provides the sensation of movement through the viewed space. There are numerous practical applications for virtual touring in sectors as diverse as: business, education, architecture, science, medicine, robotics, the military, art, entertainment, and sport. Virtual environments are set up for various purposes, such as:

- *commercial* – for example, Second Life (http://www.secondlife.com) which has fully customizable avatars and a built-in scripting and building language, where users socialize and sell real-world products;
- *education* – for example, Forterra Systems (http://www.forterrainc.com) which has domains such as e-learning, military and homeland security training, and healthcare;
- *social* – for example Kaneva (http://www.kaneva.com) that combines a virtual environment with social networking;
- *entertainment* – for example, computer games such as Everquest and World of Warcraft (http://everquest.station.sony.com/ and http://www.worldofwarcraft.com/index.xml);
- *sport* – for example, virtual orienteering (http://www.catchingfeatures.com);
- *simulation* – for situations where testing in real-life may be too expensive or dangerous, such as flight training, medical operations, and military war-games.

They can also have different target audiences, such as:

- *military* – such as for war-games and homeland security training;
- *academic* – for educational purposes;
- *architecture* – for virtual viewing of the built environment;
- *adults* – for example, Second Life, and The Sims Online (http://www.thesimsonline.com);
- *teens* – for example, Barbie Girls (http://www.barbiegirls.com), a Barbie-themed world where girls shop for fashion, play music, chat, design rooms, and play games; and
- *kids* – for example, Neopets (http://www.neopets.com) where kids play games, socialize and buy things for pets.

Implementation of virtual environments can be classified under the following three headings:

1. Pseudo-realistic;
2. Photo-realistic;
3. Non-photo-realistic.

Download free eBooks at bookboon.com

Pseudo-realistic virtual environments are the most common. They are rendered entirely by the computer, and often do not have a real-life equivalent. Some examples are shown in Figure 2.6. The screenshot shown top left, created using a game engine, shows the sophistication in terms of detail possible. The top right image is a screenshot from an orienteering sports simulation game called Catching Features. Another place where virtual environments are found is in 3D virtual worlds such as Second Life designed by Linden Lab and created by its residents (as shown in the bottom two screenshots). These are the humans who visit the virtual world using avatars they have chosen to represent themselves.

Pseudo-realistic environments are only limited by the imagination of the environment's designer(s) and the effort that has gone into creating them. They require substantial time to create realism that is reflective of real-world realism since an extremely high level of painstaking detail is often necessary to create a believable pseudo-realistic virtual environment. One method used to avoid this effort is to duplicate common objects, but this can detract from the realism and believability. Perhaps the most common examples of pseudo-realistic environments are found in modern computer games. Increasingly, they are also being used to replicate real world environments. For example, they could be used to provide a virtual view of a nature trail. However, it would require substantial effort to exactly represent all the trees and plants that could be seen on the trail to produce photo-realistic results (i.e. closely match what is seen in real-life), and usually such realism is not required for such applications. Other applications may require the exact rendering of real-life scenes, however, such as criminal investigations, or simulation of stakeouts where lines of fire need to be determined in advance.

Click on the ad to read more

Download free eBooks at bookboon.com

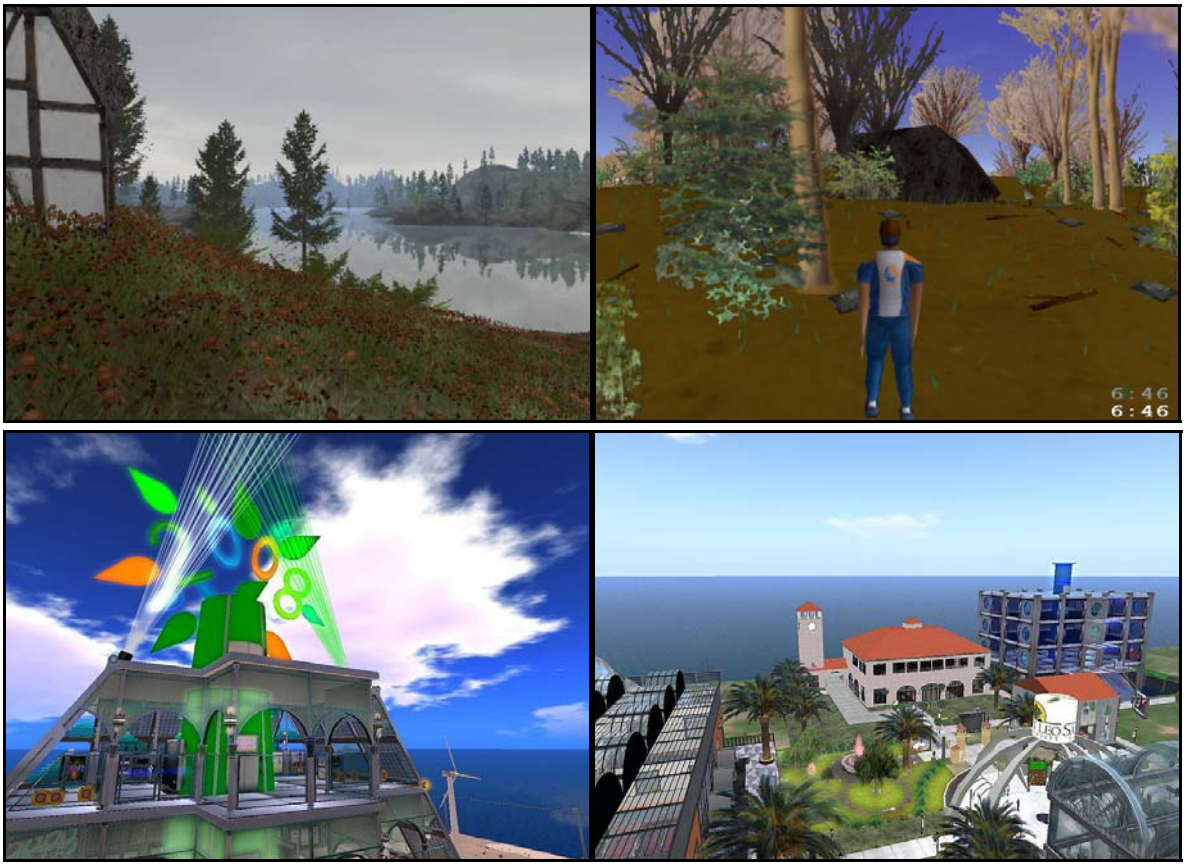


Figure 2.6: Pseudo-realistic virtual environments. (Top left) An imaginary world. (Top right) Virtual orienteering. (Bottom left & right) Screenshots from Second Life.

Photo-realistically rendered virtual environments provide an alternative where exact duplication of the real-life environment is necessary. Indeed, a long-term goal of computer graphics is to produce photo-realistic results using pseudo-realistic techniques (Roussou and Drettakis, 2003). One method is to use static perspective panoramas that provide a 360° view of the environment, and these are frequently being used on the Internet to allow a user to view how a scene might look like in real-life. Sometimes these static fixed-point representations are linked together to provide a crude virtual tour, but to the most part, the ability to dynamically move around all parts of the scene from multiple points is not possible. Being only able to move from one panorama to the next in a pre-determined manner restricts the user, and it is often difficult for the user to perceive where he or she is moving to in relation to where he or she has come from. Essentially, the virtual tour is a collection of independent viewpoints, along a pre-determined path and the panoramas are contextually independent of each other. In addition to the typically cumbersome interface, the panoramas can be expensive to create, requiring specialist equipment with an extreme wide-angle lens or 'fish-eye' lens. An alternative is to use specialist software to stitch together multiple overlapping images to create a panoramic view. However, this process is error-bound because usually overlapping images do not match where they overlap due to the different angles the images are taken from, and different lighting.

Download free eBooks at bookboon.com

Google Street View, as seen in Google Maps and Google Earth, is an alternative approach that displays images of scenes taken from cameras mounted on a fleet of cars. It allows users to view parts of selected cities at ground level and provides 360° horizontal and 290° vertical panoramic street-level views. Users can navigate around the scenes using mouse clicks and keyboard arrow keys, and the images can be viewed in different sizes, in any direction and from a variety of angles. Lines are also displayed to indicate the direction that was taken by the street view camera car when the images were captured.

Although these photorealistic environments are the closest we have for accurate representations of the world around us, they do not provide a software mechanism for easily determining what is actually depicted in the virtual environment itself, such as where the objects that can be seen are located with respect to each other. Therefore, these types of virtual environments are of limited worth when considering the task of developing intelligent agents that can sense, move around and interact within it, unless the environment is augmented with additional information about what is contained within it. The difficulty of providing this additional information, however, is equivalent in complexity to creating a pseudo-realistic copy of a real-life environment.

A third method is to render the environments non-photorealistically, for example as seen in etchings, sketches or cartoon-like landscapes (Reynolds, 2008). Objects are rendered using simplified representation of detail, concentrating on style rather than realism with detail is omitted or emphasized to communicate information more effectively (Gooch and Willemson, 2002). Often the environment is built up from just the outlines of the objects using stylized texturing. Consequently, fast rendering algorithms and extremely high frame rates are possible that allows very fluid movement allowing the environment to be navigated at high speeds. However, like both the pseudo-realistic and photo-realistic environments, some effort is needed to design environments with enough complexity for intelligent agents to exist in.

2.8 How can we develop and test an Artificial Intelligence system?

When building any system, an engineer must face the question of how best to evaluate how effective it is. If engineers ignore the evaluation phase, then they are in danger of make the same mistakes that early AI researchers did when they often omitted a full evaluation of their system. (This is often a mistake made with current AI systems as well). Evaluation is more than just testing the system to see if it is working correctly. Testing is a necessary step in the software engineering process that should not be omitted. However, evaluation goes further, and analyzes the results produced by the system, and compares the results with those produced by other systems. Without this analysis, then it is impossible to ascertain whether true progress has been made. That is, whether there have been improvements in whatever criteria are being used to evaluate the systems, or whether the design objectives have been met (such as the design objectives proposed in Chapter 10).

Download free eBooks at bookboon.com

How, then, can we develop and evaluate an AI system? One way is to create a virtual environment with as much detail and complexity that the real world has, not unlike the environments that the Catching Features software is able to create, and have the AI physically grounded within it. There are two advantages to this:

1. **The environment is computer generated.** That is, the computer already knows everything about it, since it has already been generated, and there are data structures that represent every aspect of it. In a real sense, we can neatly sidestep the issue of knowledge representation (how to represent the knowledge about the physical aspects of the environment) that are outlined in Chapter 9.
2. **If the virtual environment can be visualized, then both the human and AI can interact together within the same environment.** This could be useful if we wish to take the role of teachers to teach the AI agents to aid them in learning about the world and how best to interact with it. The role that the teacher takes could be non-competitive – more as a benevolent tutor who oversees the learning process – or it could be competitive as in the orienteering simulation where the goal is to get the computer agent to reproduce the behaviour of the human players and attain the same level of performance or perhaps exceed it. If the latter is possible, then the computer agent can take over the role of teaching or helping the human players perform better themselves (this may simply happen by the human trying to compete at the same level as the computer agents).

ericsson.
com

Shaping tomorrow's world – today
Our business is at the heart of a connected world – a world where communication is empowering people, business and society. Our networks, telecom services and multimedia solutions are shaping tomorrow. And this might just be your chance to shape your own future.

It's a people thing
We are looking for high-caliber people who can see the opportunities, people who can bring knowledge, energy and vision to our organization. In return we offer the chance to work with cutting-edge technology, personal and professional development, and the opportunity to make a difference in a truly global company.

We are currently recruiting both new graduates and experienced professionals in four areas: **Software, Hardware, Systems** and **Integration & Verification**.

Are you ready to shape **your** future? Begin by exploring a career with Ericsson. Visit www.ericsson.com/join-ericsson

ERICSSON

Click on the ad to read more

Download free eBooks at bookboon.com

2.9 Summary and Discussion

Agent-oriented design provides an alternative to the more well known object-oriented design when building computer systems. During design, we can conceptualise computer systems using the analogy of agents existing in an environment. This characterisation allows us to focus our design on specific desirable properties, such as autonomy (the agents in the system must make their own decisions), reactivity (they respond to events occurring in the environment) and proactivity (they respond to likely future events).

Agents do not exist apart from their environment. The environment is fundamental in determining the behaviour of an agent. The complexity of an agent-oriented system is a result not only of the interactions the agents have with other agents, but also with the interactions between the agents and the environment. Virtual environments are an excellent means for testing out AI systems.

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

- Agents are autonomous, reactive and proactive.
- Agents also have other properties that determine whether they are weak or strong agents.
- Agents make the decision themselves about what to do, whereas objects are invoked externally.
- Agents cannot be considered independently of their environments.
- Environments, whether real or abstract, can be represented as *n*-dimensional spaces.
- Virtual environments can be used for testing out AI systems.

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

Model	URL
N Dimensional Space	http://files.bookboon.com/ai/N-Dimensional-Space.nlogo

Download free eBooks at bookboon.com

3 Frameworks for Agents and Environments

NetLogo is a programmable modeling environment for simulating natural and social phenomena… NetLogo is particularly well suited for modeling complex systems developing over time. Modelers can give instructions to hundreds or thousands of "agents" all operating independently. This makes it possible to explore the connection between the micro-level behavior of individuals and the macro-level patterns that emerge from the interaction of many individuals. *Uri Wilensky*, 1999.

The purpose of this chapter is to explore alternative frameworks that we might use to implement an agent-oriented system. The chapter is organised as follows. Section 3.1 discusses some architectures and frameworks for agents and environments. Section 3.2 describes the FIPA standards for agent-based technologies. Section 3.3 looks at alternative agent-oriented programming language platforms. The remainder of the chapter provides an introduction to agent directed simulation in the NetLogo programming language in particular: Section 3.4 provides an introduction to NetLogo; Section 3.5 describes the NetLogo development environment; Section 3.6 looks ate how agents and environments are implemented ion NetLogo; and Section 3.7 provides some sample code that shows how mazes can be drawn in NetLogo. This will be used in latter chapters on behaviour and searching.

3.1 Architectures and Frameworks for Agents and Environments

If we are to build agent-oriented systems, we need a computer platform and some software frameworks on which to build them. When we talk about platforms and frameworks, we are using a construction analogy that relates the concept of creation of an computer system with building in real life. Words such as 'architecture', 'framework', 'platform', 'toolkit' and 'structure' are used to draw attention to the underlying construction analogy. Their precise meaning is difficult to pin down as the terms are often used interchangeably due to the implicit underlying analogy. For this book, we can refer to the architecture of an agent-oriented system as the overall conceptual design and operational structure of the system. The purpose of a framework is to simplify the design development of the system by providing higher level abstraction of the necessary components and functions of the system so that less time is spent on the lower level details necessary to get the system to work. The platform is the hardware or software framework that allows the software to run on. These include the architecture, operating system, programming languages, runtime libraries and graphical user interfaces.

Download free eBooks at bookboon.com

In this chapter, we will explore various frameworks that are available for building agent-oriented systems. Such frameworks have various design issues to overcome first, such as ('Software Agent', 2008): scheduling, synchronization, and prioritization of agent tasks; collaboration between agents; resource allocation and recruitment; re-instantiation of agents in different environments; storage of agents' internal states; the probing of the environment; messaging and communication; agent-environment interaction in the face of a changing, dynamic environment; and agent taxonomy – which hierarchy is appropriate for the task e.g. task execution agents, scheduling agents, resource providers and so on.

3.2 Standards for Agent-based Technologies

Another important issue is standardisation. If we are to develop autonomous agents with the ability to communicate with each other, for example, then it is important to agree to a set of standards on how that communication should take place. FIPA (for the Foundation for Intelligent Physical Agents) is an IEEE Computer Society standards organization concerned with agent-based technology and its interoperability with other technologies (FIPA, 2008). FIPA specifications provide a collection of standards whose purpose is to promote the interoperation of heterogeneous agents and the services they represent. In 2002, FIPA completed standardising a subset of 25 of all of its specifications. These are listed in Table 3.1. The subset as well as the complete set of specifications covers various categories such as: agent communication; agent transport; agent management; abstract architecture; and applications.

Download free eBooks at bookboon.com

Identifier	Title
SC00001	FIPA Abstract Architecture Specification
SC00008	FIPA SL Content Language Specification
SI00014	FIPA Nomadic Application Support Specification
SC00023	FIPA Agent Management Specification
SC00026	FIPA Request Interaction Protocol Specification
SC00027	FIPA Query Interaction Protocol Specification
SC00028	FIPA Request When Interaction Protocol Specification
SC00029	FIPA Contract Net Interaction Protocol Specification
SC00030	FIPA Iterated Contract Net Interaction Protocol Specification
SC00033	FIPA Brokering Interaction Protocol Specification
SC00034	FIPA Recruiting Interaction Protocol Specification
SC00035	FIPA Subscribe Interaction Protocol Specification
SC00036	FIPA Propose Interaction Protocol Specification
SC00037	FIPA Communicative Act Library Specification
SC00061	FIPA ACL Message Structure Specification
SC00067	FIPA Agent Message Transport Service Specification
SC00069	FIPA ACL Message Representation in Bit-Efficient Specification
SC00070	FIPA ACL Message Representation in String Specification
SC00071	FIPA ACL Message Representation in XML Specification
SC00075	FIPA Agent Message Transport Protocol for IIOP Specification
SC00084	FIPA Agent Message Transport Protocol for HTTP Specification
SC00085	FIPA Agent Message Transport Envelope Representation in XML Specification
SC00088	FIPA Agent Message Transport Envelope Representation in Bit Efficient Specification
SI00091	FIPA Device Ontology Specification
SC00094	FIPA Quality of Service Specification

Table 3.1 FIPA specifications for heterogeneous and interacting agents and agent-based systems.

The core FIPA specifications concern agent communication that defines a language called ACL (for Agent Communication Language). It deals with communications messages, protocols for exchanging the messages between agents, and representations for the message content.

Download free eBooks at bookboon.com

The specifications are well suited to agent applications where communication is important and increasingly, many of the agent architectures and frameworks are now becoming FIPA compliant. Communication is only one aspect of intelligence, however, and the specifications overlook important agent issues such as embodiment and situatedness – traits that have been identified in Chapter 1 (and expanded upon in Chapter 5) as important for intelligence. For example, it is surprising that there is no separate category in the specifications for environments, and how the agents are able to sense and move around within them. Therefore these specifications are of less relevance to applications where believable agent interaction with the environment is crucial – such as in computer animation, computer gaming and environmental simulation and modeling.

3.3 Agent-Oriented Programming Languages

One immediate problem for building agent-oriented systems is finding an appropriate programming language platform. Currently, comparatively few programming languages have built-in support for agented-oriented programming and no completely agent-oriented mainstream programming language exists. Various agent-oriented programming languages have been proposed, such as:

- 3APL, for implementing cognitive agents and high-level control of cognitive robots with beliefs, observations, actions, goals, communication, and reasoning rules (Dastani *et al.*, 2003).
- AgentSpeak, allows BDI (for Belief, Desires and Intentions) agent programs to be written and interpreted in a restricted first-order logic program with events and actions (Rao, 1996), and can be viewed as a simplified text language of the Procedural Reasoning System (PRS) (Ingrand *et al.*, 1992) and the Distributed Multi-Agent Reasoning System (dMARS) (D'Inverno *et al.*, 2004). Jason (Bordini *et al.*, 2007) is an open-source interpreter for AgentSpeak written in Java that allows developers to build multi-agent systems in complex environments.
- SPADES, a middleware system for the creation of agent-based simulations (SPADES FAQ, 2008).
- SPLAW, based on KQML, the standard inter-agent communication language (Xiaocong, 1998).
- STAPLE, based on joint intention theory (Kumar and Cohen, 2004).

This list is by no means exhaustive. Its purpose is to illustrate the variety of solutions that have been devised. Figure 3.1 provides a diagrammatic classification of a selection of the agent-based modelling platforms and languages.

Download free eBooks at bookboon.com

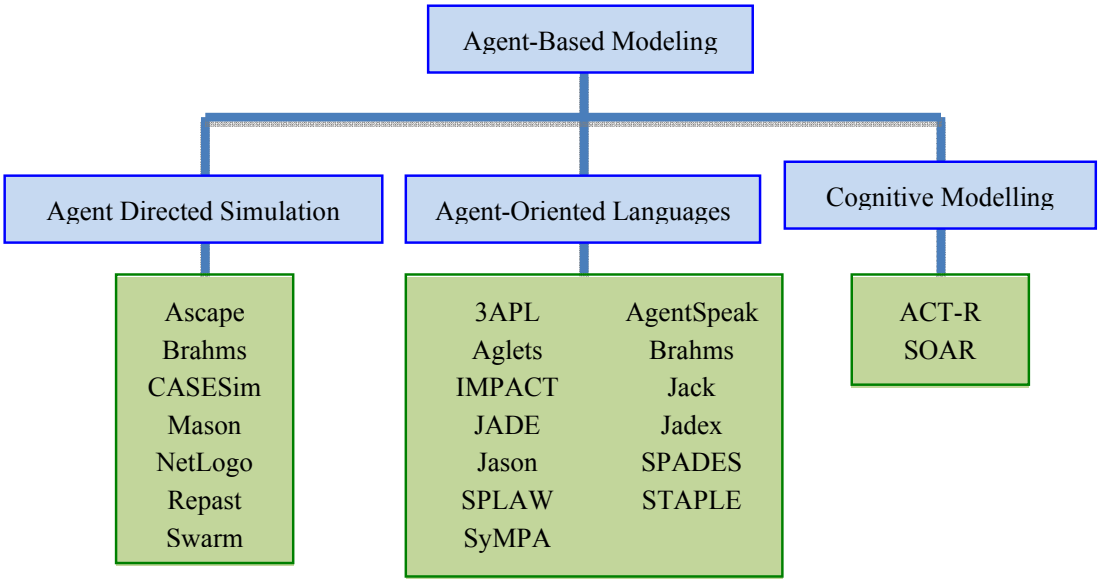


Figure 3.1. A selection of agent-oriented programming languages (based on Zhang, Lewis and Sierhuis).

Click on the ad to read more

Download free eBooks at bookboon.com

The classification in the diagram splits agent-based modelling into three categories – agent-directed simulation, agent-oriented languages, and cognitive modelling. The first category covers languages that combine agent modelling with simulation. In the remaining sections of this chapter, we will look at one particular example in this category – NetLogo. According to Zhang, Lewis and Sierhuus (2009), Netlogo's advantages are that it has a small learning curve for the beginning programmer, it is easy to debug, and provides easy access to basic visualization, as it was "designed with the inexperienced programmer in mind". Another advantage is that NetLogo programs are very compact and easy-to-read, and therefore are much easier to convert to other programming languages if required. Hence, it is an ideal tool for pedagogical purposes, and will therefore be used throughout the rest of these books.

An obvious feature of Figure 3.1 is the predominance of instances in the second category, with very few in the third category. As stated above, no agent-oriented programming language has yet to become mainstream in the way that C, Java and Python have become mainstream. Perhaps their common failing is the same failing that Prolog has – too much emphasis on a logic-based paradigm – which can be difficult to understand and debug for novice programmers. For example, Huget (2002) has listed desiderata for an agent-oriented programming language that includes features for handling logic (such as BDI for rational agents); but he has also listed other desirable features such as: knowledge and communication; definitions of organisations and environments; abstraction in order to avoid complexity in design; conformation to standards; ability to mix formalisms such as automata and Petri nets; and an event driven architecture (as used by reactive agents who react to events as they occur). Although logic-based languages are interesting and powerful in their own right, they do not cater well to what programmers like doing best – getting useful things to work well with a minimum of fuss.

Rather than develop a whole new agent-oriented programming language, an alternative approach is to develop a hybrid system on a non-agent-oriented programming language. As stated, most programming languages do not have support for agent-oriented programming. The usual solution is to develop an agent framework in an object-oriented programming language since that is the current predominant programming paradigm in vogue. There have been numerous agent frameworks developed, and most concentrate on issues to do with reasoning using a logic-based solution (such as BDI) as well as other capabilities such as agent-to-agent communication, usually using the standard KQML communication language.

What would an object-oriented agent framework look like? Objects can be used to represent agents in the system or application (Knapik and Johnson, 1999) such as scheduling agents, human interface agents, search agents, and so on. Object-oriented agents are defined with a class ('`AgentClass`' or '`RootAgentClass`' say), and all agents have various things in common such as a name or ID, other global attributes, and a basic set of communication and error-handling protocols. Subclasses are used to define more specific types of agents – for example, '`TextSearchAgent`' – that will have specific attributes and protocols. Each object-oriented agent has attributes that define the agent's state and operations/methods that define the agent's behaviour.

Download free eBooks at bookboon.com

What are the benefits of developing agents using object-oriented technologies? Knapik and Johnson (1999) list the same benefits that accrue to the object-oriented programming paradigm: re-usable code; reduced agent development costs; flexible structuring of agent design; maintainability; extensibility; understandability; support for interconnected hierarchies of agents and domains; system knowledge is intrinsic; and a readily available development environment for modeling and simulation.

An increasing number of object-oriented agent frameworks have been developed using Java. (NetLogo is also implemented in Java). Java offers an object-oriented solution that is already integrated into the universal client – Web browsers – and supports packages such as `java.net` that can be used by agents to access and extract information from Web pages. Java has tremendous potential as an agent development language due to its widespread use on the Web. Similarly, Python is another language that is increasingly being used on the Web, and has support for the object-oriented programming paradigm, but to date, there has been less agent frameworks developed for it than Java.

We will now have a brief look at a few other agent frameworks. Again, the purpose is not to be exhaustive and document the frameworks that are available as there are too many. The purpose is mainly to illustrate the variety of solutions that are possible, and to emphasize again that no single solution has captured the imaginations and interests of developers world-wide to become mainstream. In Java, applets and servelets can be considered as somewhat akin to agents (Knapik and Johnson, 1999). Applets migrate from the server to the client, so execute on the client and thereby freeing up the server, so have some degree of mobility, an important trait of agents. Java servelets also let a user upload an executable program to the network, so a client user or application could launch a servelet-based agent to search the network for information or respond automatically or give periodic updates.

IBM's aglets (for agile agents) is a Java mobile agent platform that adds further functionality to Java that is specifically focused on agent tasks (Aglets, 2008). Originally developed at the IBM Tokio Research Laboratory, the Aglets technology is now hosted at sourceforge.net. An aglet is a Java agent able to autonomously and spontaneously move from one host to another roaming the Internet. A complete Java mobile agent platform is available, along with a stand-alone server called Tahiti, and there is a library that allows developer to build mobile agents or to have aglet code embedded in an application. Aglets can be halted, be packaged with their current state in another object dispatched to another environment, and have their execution resumed. The Java API has various classes such as: the class `Aglet`, an abstract class which is used to define the aglets; the `AgletContext` which is an interface an aglet uses to gain knowledge about its environment; the `AgletProxy` that is a class that encapsulates the real aglet, protecting against direct access to the aglet's public interface; the `AgletIdentifier` that is a class that sets up a unique identifier for an aglet; the `Itinerary` class represents the routing to travel plans for an aglet; and the Message class that enables communication between agents.

Download free eBooks at bookboon.com

JADE (for Java Agent DEvelopment Framework) is another framework fully implemented in Java (JADE, 2008) and is free software distributed by Telecom Italia. It supports the development of multi-agent systems that comply to the FIPA specifications. There are graphical tools that aid the debugging and deployment phases, and remote GUI can be used to control the agent configuration, which can be distributed across machines, and which can be altered at run-time by moving agents from one machine to another.

LEAP (for Light Extensible Agent Platform) is an extension of JADE that allows a FIPA-compliant platform with reduced footprint compatible with mobile Java environments to run on wireless devices and PDA's such as cell phones and Palm computers. WADE (Workflows and Agents Development Environment) is an extension to JADE that allows agents to execute tasks defined according to the workflow metaphor. A WADE workflow is a Java class with a well defined structure that allows the developer to define a process in terms of the activities to be executed, their activation and termination criteria, and the relationships between them. Further information can be specified such as the participants in the workflow, the software tools to be invoked, the required inputs and expected outputs and internal data that will be manipulated during the execution. The advantage of the workflow approach is that the execution steps as well as their sequencing are all made explicit. WADE comes with a development platform called WOLF that is an Eclipse plug-in.

Click on the ad to read more

Download free eBooks at bookboon.com

As stated above, mainly for pedagogical reasons, an agent directed simulation approach will be adopted for these books rather than the alternative approaches of using an agent programming language or using cognitive modelling.

3.4 Agent Directed Simulation in NetLogo

NetLogo uses an approach that is different to the classical logic-based approach that agent-oriented programming languages and frameworks usually take. It illustrates the power of the agent-oriented approach at tackling problems in a non-traditional way, and has the potential for overcoming some of the barriers to developing useful multi-agent systems.

NetLogo, first designed by Uri Wilensky in 1999, is a cross-platform multi-agent programmable modelling environment under continuous development at the CCL (Centre for Connected Learning and Computer-based Modelling) at Northwestern University. NetLogo can be used for the rapid prototyping of simulations of natural and social phenomena. It is based on an earlier graphics-oriented language called Logo developed by Seymour Papert in the 1960s and is distinguished from the more traditional agent-oriented programming languages in that it does not support logic-based formalisms. NetLogo adopts an event-driven architecture where agents with simple reactive behaviours can yield astonishingly complex simulations despite being situated in a limited two-dimensional (2D) grid environment (a version with a 3D grid environment is currently under development).

NetLogo describes the environment it uses for visualizing its simulations as a 'world' (in other words, it making an analogy between its environment and the real world). This world is made up of 'agents' that perform activities by following instructions specified by methods programmed in the NetLogo programming language. These activities are carried out simultaneously for all agents in the world. NetLogo has four types of agents: *turtles*, *patches*, *links* and the *observer*. Turtles are agents that move around and interact with the world. The reason they are called turtles has been inherited from the Logo programming language – the analogy is of an imaginary robotic turtle that has the ability to move around and to lift or drop a virtual coloured drawing pen it has on its back onto a drawing canvas. The 2D grid is also made up of patches that are square pieces of land that the turtles can move over, and move around (see right box in Figure 3.2 below).

Links are agents that connect two turtles together. The observer looks out over the world of turtles and patches but does not have a specific location (in some sense it is not embodied like the other agents).

Download free eBooks at bookboon.com

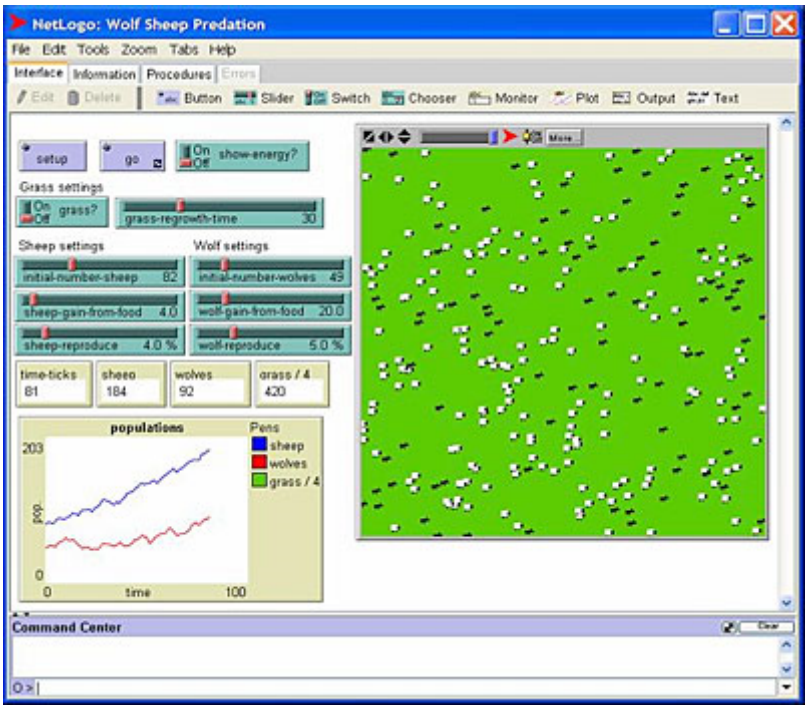


Figure 3.2: Screenshot of the Wolf Sheep Predation model in NetLogo.

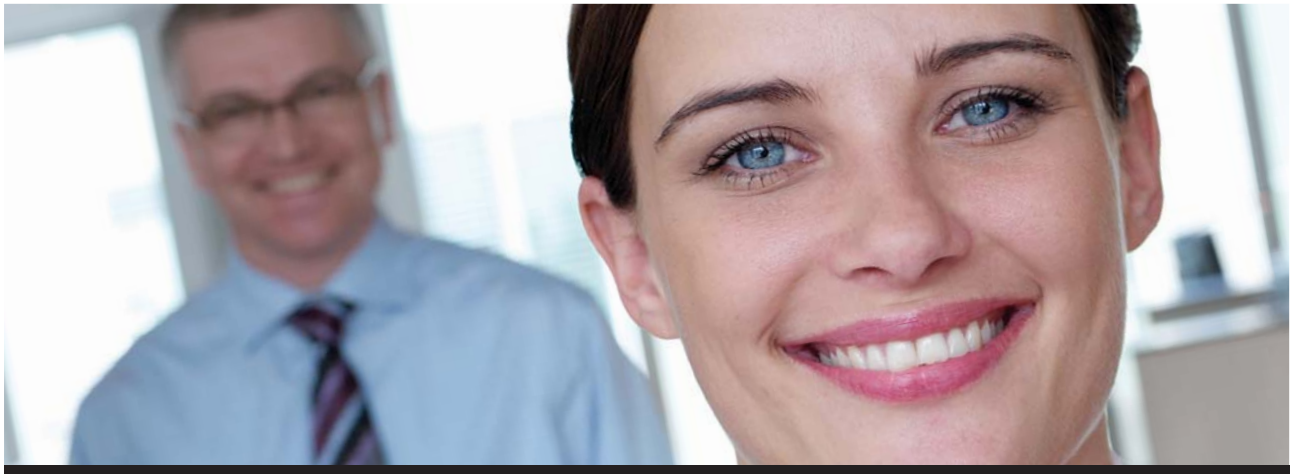
NetLogo comes with a rich set of sample 'models' (or programs) that are simulations of natural or social phenomena in a number of areas such as: art, biology, chemistry & physics, computer science, earth science, mathematics, and social science. For example, the Wolf Sheep Predation model explores the stability of predator-prey ecosystems. In the simulation, there are two types of breeds (distinct types of turtle agents) – wolves and sheep, as well as grass patches. The wolves eat the sheep and the sheep eat the grass. Depending on the start-up conditions (number of wolves, number of sheep, reproduction-rates of wolves and sheep, how much energy they gain from eating, where they are randomly located in the environment, and grass re-growth time), the simulation will result in an unstable system where either the wolves or both the wolves and sheep become extinct, or it will result in a stable system where it will tend to maintain itself despite fluctuations in population sizes over time. A screenshot of the model is shown in Figure 3.2.

NetLogo code is very readable, and compact. Listed below is the code for the main go procedure that specifies what happens when the Wolf Sheep Predation simulation is run.

Download free eBooks at bookboon.com

```
to go
  if not any? turtles [ stop ]
  ask sheep [
    move
    if grass? [
       set energy energy- 1 ;; deduct energy for sheep only if grass
                              ;; switch is on
      eat-grass
    ]
    reproduce-sheep
    death
  ]
  ask wolves [
    move
    set energy energy- 1 ;; wolves lose energy as they move
    catch-sheep
    reproduce-wolves
    death
  ]
  if grass? [ ask patches [ grow-grass ] ]
  tick
  update-plot
  display-labels
end
```

NetLogo Code 3.1: Main go procedure for the Wolf Sheep Predation model in NetLogo.



We have ambitions. Also for you.

SimCorp is a global leader in financial software. At SimCorp, you will be part of a large network of competent and skilled colleagues who all aspire to reach common goals with dedication and team spirit. We invest in our employees to ensure that you can meet your ambitions on a personal as well as on a professional level. SimCorp employs the best qualified people within economics, finance and IT, and the majority of our colleagues have a university or business degree within these fields.

Ambitious? Look for opportunities at www.simcorp.com/careers

SimCorp

www.simcorp.com

Click on the ad to read more

Download free eBooks at bookboon.com

The ask command is used to specify the behaviour of the turtle agents, patches and links. In the code above, the behaviour of the sheep is first to move, then to eat grass, reproduce and die. The behaviour of the wolves is to move, catch sheep (and eat them), reproduce then die.

The many varied models that come with NetLogo illustrate that the agent-oriented programming paradigm it adopts is adept at modelling a surprising range of phenomena in a non-complicated way. For example, Figure 3.3 shows the Tumor model that simulates the growth of a tumour and how it resists chemical treatment. A tumour consists of two kinds of cells: stem cells represented by blue turtle agents; and transitory cells represented by all other turtles. The figure illustrates how the cell advances into distant sites and how it creates another tumour colony, a process called metastasis shown in red.

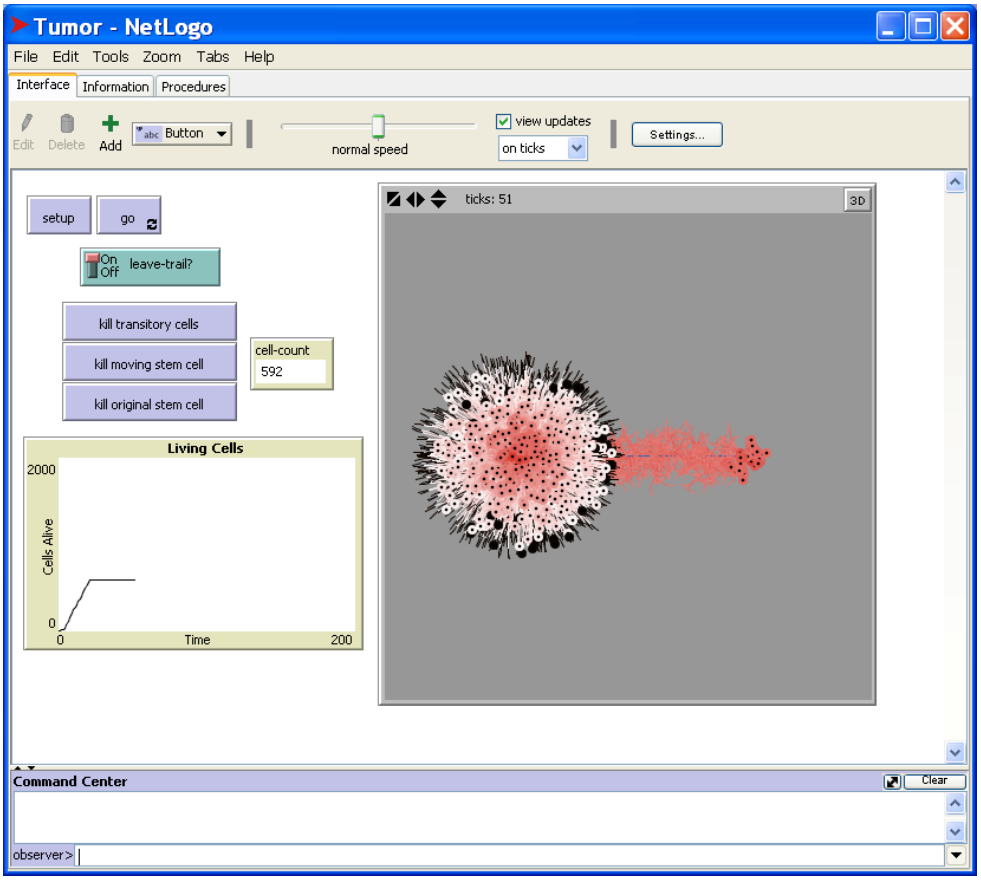


Figure 3.3: Screenshot of Tumor model in NetLogo.

NetLogo is an excellent tool for illustrating the principle that complexity can arise from the interaction of agents who individually apply simple reactive behaviours, but collectively, exhibit much more – that is, the system as a whole is greater than the sum of its parts.

Throughout these books, we will look at sample code in NetLogo to show how various aspects of agent-oriented systems can be implemented.

Download free eBooks at bookboon.com

3.5 The NetLogo development environment

The reader should familiarise herself or himself with the documentation that comes with the NetLogo development environment. You will need to refer to this often if you wish to develop your own programs. Once you have started the NetLogo application, select the Help Menu, then select "NetLogo User Manual". You can follow the following tutorials to learn a bit more about NetLogo:

- Sample Model: Party
- Tutorial #1: Models
- Tutorial #2: Commands
- Tutorial #3: Procedures

For language references, you have the following choices:

- Interface Guide
- Programming Guide
- Transition Guide
- NetLogo Dictionary

The Interface Guide provides a summary of the NetLogo user interface and how to navigate around it. Most of the user interface is fairly obvious, so we will not repeat that material here. If you are having difficulties in finding things in the user interface, then this is the place to find about the user menus and user options available.

Perhaps the single most useful thing to know concerning the user interface is how to load the Models Library and select a specific model. In the File tab menu, select "Models Library" and this will load a large library of models separated into different subject areas such as Art, Biology, Computer Science, Chemistry and Physics and so on. To familiarize yourself with the interface, select the Biology subject area, and then select the Termites model. After clicking on the Setup button, you should have the following appear on your screen:

Download free eBooks at bookboon.com

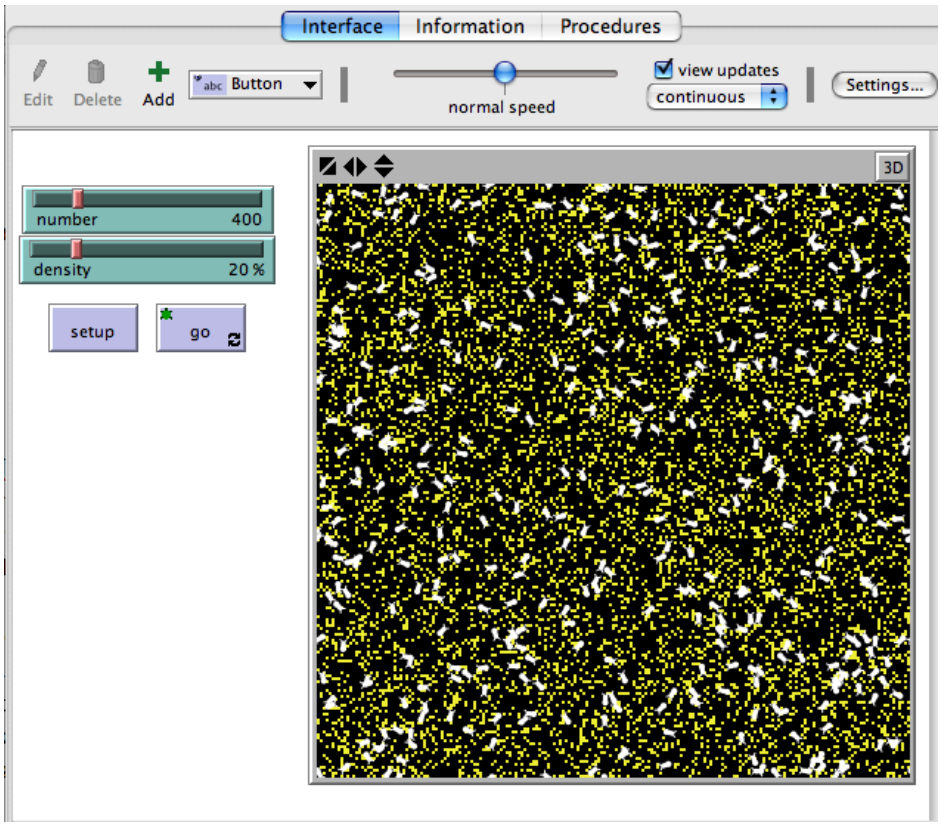
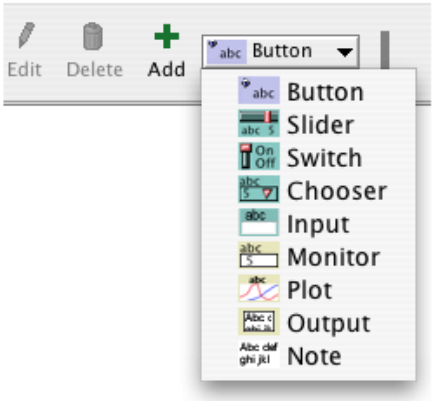


Figure 3.4 A screenshot of the Interface for the NetLogo Termites model.

Click on the ad to read more

Download free eBooks at bookboon.com

Note the three menu buttons at the top – `Interface, Information and Procedures.` When you have selected the `Interface` button, then you will be provided with an interface to your program and a visualization of the current state of the (usually 2D) NetLogo environment as shown in Figure 3.4. This will vary depending on the model or application currently being executed. Usually there is a `setup` button to set up the initial state of the environment, and a `go` button to start the simulation. Sometimes a `go-once` button might be provided – this will execute the model through a single simulation step or tick. At other times, a `go-forever` button will be provided which will execute the model indefinitely. These interface elements can be added by clicking on the `Button` menu and selecting from a number of items such as buttons, sliders and choosers:



Various of these interface elements define global variables or require a specific command (i.e. procedure) to be defined in the program somewhere. For example, the `go` button requires a command called `go` to be defined in the code, but the label displayed on the button can be overridden if required.

The `Information` button will switch to a screen where information is displayed about the model that the developer has provided. The `Procedures` button will switch to a screen mode where the model's NetLogo code is displayed and can be edited. For the Termites model, the first two commands specify the code to be executed when the `setup` and `go` buttons are clicked:

Download free eBooks at bookboon.com

```
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
  create-turtles number [
    set color white
    setxy random-xcor random-ycor
    set size 5 ;; easier to see
  ]
end
to go ;; turtle procedure
  search-for-chip
  find-new-pile
  put-down-chip
end
```

NetLogo Code 3.2 Code at the beginning of the Termites model.

NetLogo is very concise and easy to read. The `setup` procedure randomly distributes the wood chips and termites (seen as yellow patches and white ant-like icons respectively in the environment after the `setup` button has been clicked). The `go` procedure executes three further procedures – searching for a wood chip, finding a new pile and putting down a wood chip – that define the behaviour of all the termite agents each tick. The code for these three procedures can be found in the remainder of the code that is provided with the model.

The Programming Guide provides a summary to key elements of the NetLogo programming language. It provides an overview of the language's important features, and can be a useful source for programming examples. It is worth a read, especially when first starting out with the language. However, the NetLogo Dictionary is where NetLogo developers will spend most of their time and it contains links to all the commands that are available in the language. The Transition Guide describes earlier versions of NetLogo and what has changed in latter versions, so is therefore of less benefit for someone learning how to program in NetLogo.

Download free eBooks at bookboon.com

3.6 Agents and Environments in NetLogo

We will now look at how agents and environments are implemented in NetLogo. Technically, the turtles implemented in NetLogo could be classified as 'proto-agents' rather than full-blown 'agents' as discussed in Section 2.3, as they are not autonomous and they do not have explicit sensing capabilities (these have to be programmed), although they do have characteristic behaviour in a limited sense, and are situated in a rudimentary 2D environment. However, we find the proto-agent/agent distinction arbitrary, and we will prefer instead to refer to the turtles as agents as they are described in NetLogo documentation, and also because they have the design potential to achieve more elaborate agent properties if programmed in a more sophisticated way, such as by using threads for each turtle, and/or by providing the turtles with greater sensing capabilities.

One way of thinking about them is as a set of pre-programmed software 'robots' that we have control over as the developer. We can explicitly program the set of instructions or commands that control what they do when they are executed, so there is nothing really different to a standard object-oriented approach. The power of NetLogo, however, comes from the ease with which it allows multiple agents (in many cases, hundreds or even thousands) to be created and executed simultaneously.

Click on the ad to read more

Download free eBooks at bookboon.com

As stated, there are four types of agents in NetLogo:

1. turtles;
2. patches;
3. links;
4. the observer.

Turtle agents are agents that are situated in the environment and move around within it. The 2D environment consists of a grid of patches over which the agents can move. Using an analogy with a real-life environment, we can think of the patches as being square pieces of ground laid out in a grid pattern much like Manhattan city blocks. Link agents link together two turtle agents. The observer agent is a single agent that oversees (or 'observes') the environment and the agents, but does not have a specific location like the other three types of agents.

In NetLogo, a set of agents is called an *agentset*. The language has a rich set of commands that allow us to easily set attributes for an entire set of agents (such as their colour as in NetLogo code example 3.4 below). The built-in variables `turtles`, `patches` and `links` hold the sets of all the turtle, patch and link agents that currently exist in the environment.

The developer also has the ability to define turtle and link agents as new breeds of agents. For example, the following command will define two breeds of agents, foxes and rabbits:

```
breed [foxes fox]
breed [rabbits rabbit]
turtles-own [age gender]
```

NetLogo Code 3.3 Code to define two breeds of agents – foxes and rabbits.

The `turtles-own` command specifies that all the turtles (including foxes and rabbits) own the attributes `age` and `gender`. Attributes associated with a specific breed can be defined using the breed specific command `<breed>-owns` – for example, `foxes-own` and `rabbits-own` become legitimate commands that can be used in the program once the above `breed` commands have been defined. New turtle and link agents can be created using the `create-turtles` command, or by breed specific commands such as `create-foxes` and `create-rabbits`. The following code will create 100 foxes and 1000 rabbits at random locations throughout the NetLogo environment all of age zero but with random gender:

Download free eBooks at bookboon.com

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

turtles-own [age gender]

to setup
  clear-all ;; clear everything

  create-foxes 100 [
     set age 0
     set size 2
     set color brown
     ifelse random 2 = 0
       [set gender "Male"]
       [set gender "Female"]
     setxy random-xcor random-ycor
]
  create-rabbits 1000 [
     set age 0
     set size 2
     set color white
     ifelse random 2 = 0
       [set gender "Male"]
       [set gender "Female"]
     setxy random-xcor random-ycor
]
end
```

NetLogo Code 3.4 Code to create some foxes and rabbits.

An example screenshot of the resulting environment is shown in Figure 3.5.

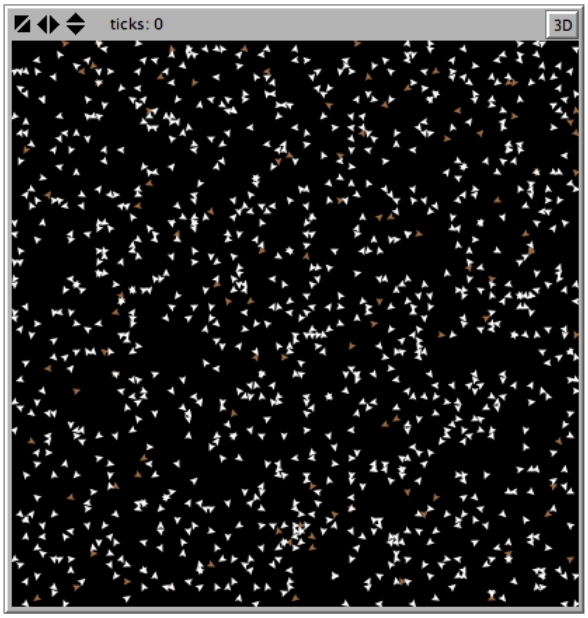


Figure 3.5 A screenshot of the environment created by the Foxes and Rabbits model whose code is listed in NetLogo Code 3.4.

Download free eBooks at bookboon.com

The following code provides a further example of how to define a turtle agent breed called `all-blacks` with four attributes – `name, pos, weight, height` and `debut` – and initialize these attributes using a selection of the data provided in Table 2.7.

```
breed [all-blacks all-black]
all-blacks-own [name pos weight height debut]

to initialise-all-blacks
    create-all-blacks 1 [set name "Sid Going" set pos "Half-back"
                         set weight 81 set height 170 set debut 1967]
    create-all-blacks 1 [set name "David Kirk" set pos "Half-back"
                         set weight 73 set height 173 set debut 1983]
    create-all-blacks 1 [set name "Justin Marshall" set pos "Half-back"
                         set weight 95 set height 179 set debut 1995]
    create-all-blacks 1 [set name "Piri Weepu" set pos "Half-back"
                          set weight 94 set height 178 set debut 2004]
    create-all-blacks 1 [set name "Grant Batty" set pos "Winger"
                         set weight 70 set height 165 set debut 1972]
    create-all-blacks 1 [set name "John Kirwan" set pos "Winger"
                         set weight 97 set height 191 set debut 1984]
    create-all-blacks 1 [set name "Jonah Lomu" set pos "Winger"
                         set weight 119 set height 196 set debut 1994]
    create-all-blacks 1 [set name "Doug Howlett" set pos "Winger"
                         set weight 93 set height 185 set debut 2000]
    create-all-blacks 1 [set name "Joe Rokocoko" set pos "Winger"
                         set weight 98 set height 189 set debut 2003]
end
```

NetLogo Code 3.5 Code to create and initialize attributes for some All Blacks agents (see N Dimensional Space Model).

Attributes of agents can be altered easily. For example, one can change the background by setting the patch colour for all patches to white (rather than the default black) in the following way:

```
ask patches
 [set pcolor white] ;; make background full of white patches
```

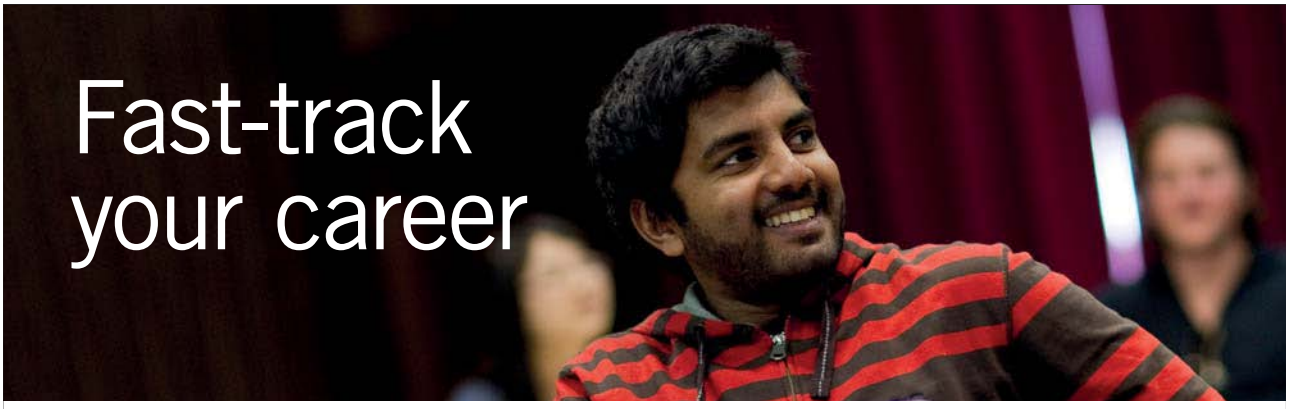
NetLogo Code 3.6 Code to set all patches in the environment to white.

Some built-in attributes of patches are its co-ordinates specified by `pxcor` and `pycor`, and its color `pcolor`. Similarly, turtles and links have the built-in variables `xcor, ycor` and `color.` Note that although the patch agents cannot move around like the turtle agents, patches have user-defined attributes and also have the ability to sprout agents (an analogy might be of new rabbits popping out of a hole in the ground). For this reason, the patches are considered to be a type of agent that is static (stays in one place) rather than dynamic (has the ability to move around like turtle agents do).

Download free eBooks at bookboon.com

Note also that the size of the patches can be altered. A developer can change the size by clicking on the `Settings` button in the `Interface` screen (see top right Figure 3.4). This will come up with a list of model settings that specify the size (in number of patches) of the environment and its co-ordinate system, whether the environment wraps horizontally and/or vertically, the patch size in pixels, turtle shapes, the font size for labels on agents, and whether the tick counter is shown or not.

As a further example, the following code draws the outside of an empty 2D maze that is centred at co-ordinates (0,0) and has an entrance in the middle of the bottom of the maze and an exit in the middle top. The code uses five global variables – `left-cols`, `right-cols`, `above-rows`, `below-rows` and `entrance-cols` – that specify the number of columns (horizontal patches) to the left and right of the line $x = 0$, the number of rows (vertical patches) above and below the line $y = 0$ and the width in patches of the entrance. The values of these variables may be altered in the Interface screen as shown in Figure 3.6.



Masters in Management

Stand out from the crowd

Designed for graduates with less than one year of full-time postgraduate work experience, London Business School's Masters in Management will expand your thinking and provide you with the foundations for a successful career in business.

The programme is developed in consultation with recruiters to provide you with the key skills that top employers demand. Through 11 months of full-time study, you will gain the business knowledge and capabilities to increase your career choices and stand out from the crowd.

Applications are now open for entry in September 2011.

London Business School

London Business School
Regent's Park
London NW1 4SA
United Kingdom
Tel +44 (0)20 7000 7573
Email mim@london.edu
www.london.edu/mim/

For more information visit www.london.edu/mim/
email mim@london.edu or call +44 (0)20 7000 7573

Click on the ad to read more

Download free eBooks at bookboon.com

```
; Empty Maze Demo
;
; Draws an empty maze with entrance at middle bottom and exit
; at middle top
; setup empty maze at (0,0)
to setup-empty-maze
  ca ;; clear everything

  ask patches
  [
    set pcolor white ;; make background full of white patches
    if (pxcor >= (- left-cols - entrance-cols) and
        pxcor <= (- entrance-cols) and pycor = (- below-rows))
      [set pcolor blue] ;; draws bottom left horizontal wall in blue

    if (pxcor >= (entrance-cols) and
        pxcor <= (right-cols + entrance-cols) and pycor = (- below-rows))
      [set pcolor blue] ;; draws bottom right horizontal wall in blue

    if (pxcor >= (- left-cols - entrance-cols) and
        pxcor <= (- entrance-cols) and pycor = (above-rows))
      [set pcolor blue] ;; draws top left horizontal wall in blue

    if (pxcor >= (entrance-cols) and
        pxcor <= (right-cols + entrance-cols) and pycor = (above-rows))
      [set pcolor blue] ;; draws top left horizontal wall in blue

    if (pxcor = (- left-cols - entrance-cols) and
        pycor >= (- below-rows) and pycor <= (above-rows))
      [set pcolor blue] ;; draws left vertical wall in blue

    if (pxcor = (right-cols + entrance-cols) and
        pycor >= (- below-rows) and pycor <= (above-rows))
      [set pcolor blue] ;; draws right vertical wall in blue
  ]
end
```

NetLogo Code 3.7: Code to create an empty maze (see Empty Maze model).

The output produced by the program is shown in Figure 3.6. The settings are `max-pxcor 50`, `max-pycor 50` and `Patch size 3`.

Download free eBooks at bookboon.com

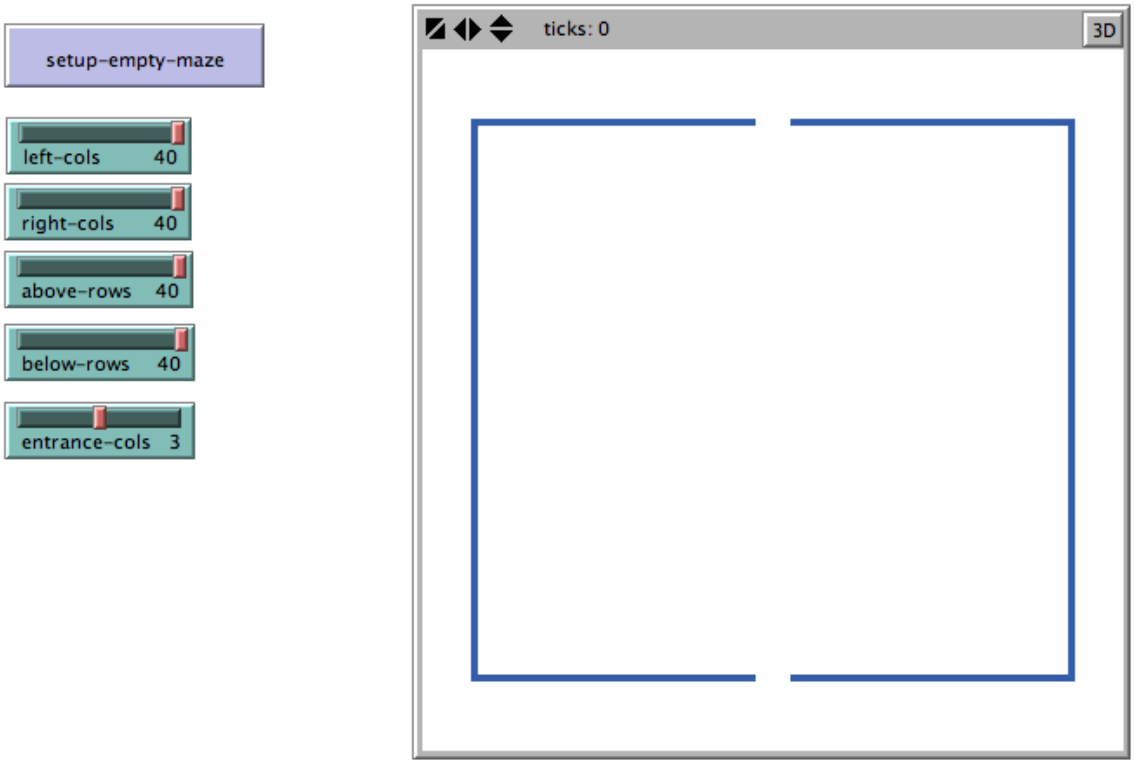


Figure 3.6 A screenshot of the empty maze created by the Empty Maze NetLogo model listed in NetLogo Code 3.7.

3.7 Drawing Mazes using Patch Agents in NetLogo

A maze is a deliberately confusing series of paths or passages leading to a specific point, and designed to make life difficult, with lots of false turnings, dead ends and traps. Mazes are very ancient – in Greek mythology the Minotaur was imprisoned in a maze – and they can still be found in the gardens of great houses, like the famous maze at Hampton Court Palace.

Juliet and Charles Snape, *The Fantastic Maze Book*.

The hedge maze at Chevening House, England, c.1820, was one of the first consciously designed to provide a more complex puzzle and thwart the "hand-on-wall" rule for solving mazes.

"Maze Typology", URL http://www.labyrinthos.net/typomaze01.html

Download free eBooks at bookboon.com

Drawing mazes usually requires more effort than the straightforward code provided in NetLogo Code 3.7 for drawing an empty maze. The quote at the top of this section refers to the Hampton Court Palace Maze, a famous garden maze in England. A stylized aerial view of the maze closer to its real shape is shown on the right in Figure 3.7. Rather than drawing the outside of this maze first, and possibly drawing inwards in some fashion, we can adopt a different solution. In NetLogo we can use patch agents to draw the walls and specify each wall back and forth across the maze in a zigzag fashion, first defining all the horizontal walls followed by all the vertical walls. NetLogo code that uses this solution to draw the schematic map of the Hampton Court Place Maze shown on the left of Figure 3.7 is listed in NetLogo Code 3.8.

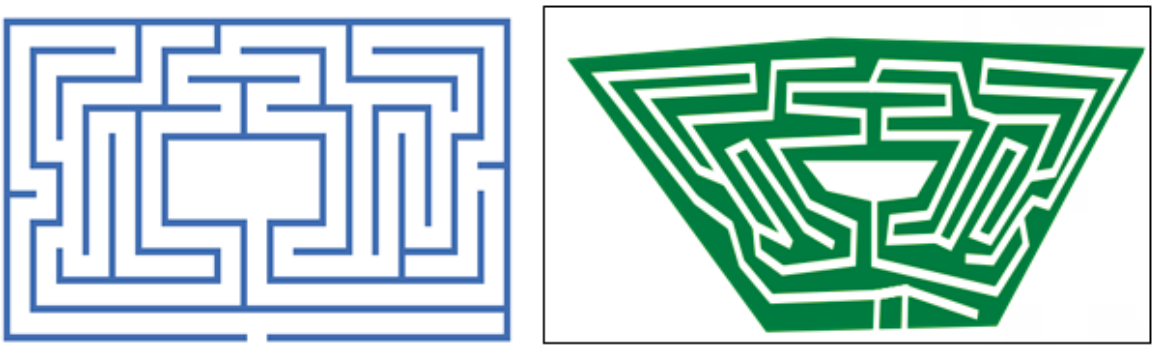


Figure 3.7 A screenshot of the schematic map (on the left) of the Hampton Court Palace Maze created by the program listed in NetLogo Code 3.8. A stylised map of the maze closer to its real shape is shown on the right.

Click on the ad to read more

Download free eBooks at bookboon.com

Looking at the map and image in Figure 3.7, it is clear that the map does not exactly represent the real life maze. (The reader can readily verify this using Google Earth). The map is a 'schematic' that distorts the relationships between objects that exist in the real life environment that it represents, in a similar way that the London Underground Map does not exactly represent the true distances and geographical relationships in the London Underground railway system. The distortions are clearly visible, but it is important to realize that distortions are characteristic of *all* maps whose purpose is to represent a real life environment. Even precise topographical or orienteering maps will have distortions due to the scale, as a result of the width required to draw each of the symbols on the map. For example, motorways drawn on topographical maps are not drawn to scale – if they were, then the motorways would have to be hundreds of meters across in real life. In addition, *every* map will have errors (and usually many). For example, the map on the left in Figure 3.7 has a short wall jutting out at right angles half way up the leftmost wall of the maze, whereas in real life the hedge is continuous with no sharp angles. However, despite these distortions and errors, the map can be readily used as an aid to navigation – people using the map have the knowledge of how to quickly get to the centre of the maze (the goal) without making a mistake. Note that we will be using maps as a basic building block in Chapter 9 for describing knowledge and concepts – two important aspects for building an agent with artificial intelligence.

Let us now examine the code in NetLogo Code 3.8 that draws the map in Figure 3.7. We can simplify by first defining general procedures to draw horizontal or vertical lines, with gaps in them. The procedures `setup_row` and `setup_col` take four parameters as input – the number of the row (`row`) or column (`col`) where the line of patch segments is drawn, `row-width` and `col_width` that is the width in number of patches that defines the width of both the rows and columns of the maze (we will keep them the same for this example), and `segments`, a list of start and finish positions that specify where each segment of patches begin and end on the same line. The rest of the code draws each row and column in the maze (the horizontal and vertical lines that represent the hedges in the maze).

```
; Hampton Court Maze Demo
;
; Draws a schematic map of the Hampton Court Maze
; setup maze at (0,0)
to setup-row [row colour segments]
   foreach segments
   [
     if pycor = row * row-patches-width and
       (pxcor >= col-patches-width * (item 0 ?)) and
       (pxcor <= col-patches-width * (item 1 ?))
         [set pcolor colour]
   ]
end
to setup-col [col colour segments]
   foreach segments
   [
     if pxcor = col * col-patches-width and
       (pycor >= row-patches-width * (item 0 ?)) and
       (pycor <= row-patches-width * (item 1 ?))
         [set pcolor colour]
   ]
end
```


Download free eBooks at bookboon.com

```
to setup-hampton-maze
   ca ;; clear everything

   ask patches
   [
      if (pxcor >= min-pxcor and pxcor <= max-pxcor and
          pycor >= min-pycor and pycor <= max-pycor)
          [set pcolor white] ;; make background full of white patches

      setup-row 5 blue [[-9 10]]
      setup-row 4 blue [[-8 -5] [-3 -1] [0 3] [5 9]]
      setup-row 3 blue [[-7 -4] [-2 2] [4 8]]
      setup-row 2 blue [[-6 -1] [1 4] [5 7]]
      setup-row 1 blue [[-3 3] [8 9]]
      setup-row 0 blue [[-8 -7] [9 10]]
      setup-row -1 blue [[-9 -8]]
      setup-row -2 blue [[-8 -7] [-3 0] [1 3]]
      setup-row -3 blue [[-4 -1] [2 4] [6 8]]
      setup-row -4 blue [[-7 -1] [1 9]]
      setup-row -5 blue [[-8 10]]
      setup-row -6 blue [[-9 0] [1 10]]

      setup-col 10 blue [[-6 5]]
      setup-col 9 blue [[-4 -1] [1 4]]
      setup-col 8 blue [[-3 1] [2 3]]
      setup-col 7 blue [[-2 2]]
      setup-col 6 blue [[-4 1]]
      setup-col 5 blue [[-3 2]]
      setup-col 4 blue [[-3 2] [3 5]]
      setup-col 3 blue [[-2 1] [2 4]]
      setup-col 1 blue [[-4 -2]]
      setup-col 0 blue [[-5 -2] [1 3]]
      setup-col -1 blue [[-4 -3] [4 5]]
      setup-col -3 blue [[-2 1] [2 4]]
      setup-col -4 blue [[-3 2] [3 5]]
      setup-col -5 blue [[-4 1]]
      setup-col -6 blue [[-3 2]]
      setup-col -7 blue [[-4 -3] [-2 0] [1 3]]
      setup-col -8 blue [[-5 -2] [0 4]]
      setup-col -9 blue [[-6 5]]
   ]
end
```

NetLogo Code 3.8: Code to generate the schematic map of the Hampton Court Maze shown in Figure 3.7 (see Hampton Court Maze model).

Note that there are many ways to draw a maze. In NetLogo, we do not have to use patch agents to do the drawing – we could use a turtle agent to do the drawing instead by having the turtle move directly over the hedge lines drawing them as it goes. However, a limitation of NetLogo is that anything drawn by a turtle agent cannot be 'seen' by other turtle and patch agents – they do not have the ability to sense or react to the drawings.

Download free eBooks at bookboon.com

Since we wish to use these mazes in future to demonstrate how you might program an embodied agent to search them, we require the ability that other agents can sense where the hedges are in the virtual environment. Also, the method adopted here of drawing the horizontal hedges followed by the vertical hedges is not necessarily the most efficient in distance covered as the turtle travels over the 2D environment twice. However, if the turtle agent is visible (that is, the turtle is animated), the behaviour of the turtle agent is easily discernible to the observer – a useful feature that helps make the solution easier to understand and debug. An extension might be to use multiple agents – for example, one turtle could be used to draw the horizontal walls, and another to draw the vertical ones. If each agent is implemented as a thread, then both can be executed simultaneously, thus halving the execution time if the number of horizontal paths equals the number of vertical paths. However, NetLogo does not implement the turtle agents as separate threads, so there is no execution advantage to using multiple agents in this case.

The Hampton Court maze is a simply-connected maze. A human agent who adopts the behaviour of keeping a hand on the wall will always be able to get to the centre of the maze (the goal). The Chevening House maze in England built in the 1820s was one of the first mazes that was multiply-connected (that is, there is one or more "islands" in the maze that are isolated from each other, totally disconnected from the outer wall). A schematic map of the maze is shown in Figure 3.8. The maze was drawn using the NetLogo code listed in NetLogo Code 3.9 using the same approach as above of drawing horizontal hedges followed by the vertical hedges. The only difference is that `row-patches-width` and the `column-patches-width` have been chosen to be twice the global patch size so that the map more closely mirrors the schematic maps of the real life maze that can be found online.

STUDY FOR YOUR MASTER'S DEGREE
IN THE CRADLE OF SWEDISH ENGINEERING

Chalmers University of Technology conducts research and education in engineering and natural sciences, architecture, technology-related mathematical sciences and nautical sciences. Behind all that Chalmers accomplishes, the aim persists for contributing to a sustainable future – both nationally and globally.

Visit us on **Chalmers.se** or **Next Stop Chalmers** on facebook.

Click on the ad to read more

Download free eBooks at bookboon.com

If one follows all the walls radiating out from the centre of the maze, one will find that these walls are not connected in any way to the outer wall. Consequently, the "hand-on-the-wall" behaviour will no longer be good enough for solving the maze. The purpose of this second maze example is to illustrate how behaviour appropriate for one environment may no longer be appropriate for a different environment (such as a new environment or when the original environment has changed). In order for an agent to continue to achieve a particular goal, the agent has to adapt (or learn) to cope with the altered environment. We will examine behaviour and the important role it has to play for autonomous agents later in Chapter 6.

Figure 3.8 A screenshot of the schematic map of the Chevening House Maze created by the Chevening House Maze model listed in NetLogo Code 3.9.

Download free eBooks at bookboon.com

```
to setup-chevening-house-maze
   ca ;; clear everything

   ask patches
   [
      if (pxcor >= min-pxcor and pxcor <= max-pxcor and
         pycor >= min-pycor and pycor <= max-pycor)
            [set pcolor white] ;; make background full of white patches

      setup-row 12 blue[[-11 12]]
      setup-row 11 blue [[-10 11]]
      setup-row 10 blue [[-9 10]]
      setup-row 9 blue [[-8 0] [1 9]]
      setup-row 8 blue [[-7 -1] [2 8]]
      setup-row 7 blue [[-6 0] [1 7]]
      setup-row 6 blue [[-5 0] [1 6]]
      setup-row 5 blue [[-4 -1] [2 5]]
      setup-row 4 blue [[-3 0] [1 4]]
      setup-row 3 blue [[-2 3]]
      setup-row 2 blue [[-1 2]]
      setup-row 1 blue [[-9 -5] [-4 -2] [3 5] [6 8] [10 11]]
      setup-row -0 blue [[-9 -7] [3 5] [6 10]]
      setup-row -1 blue [[-1 0] [1 2] [7 9]]
      setup-row -2 blue [[-2 0] [1 3]]
      setup-row -3 blue [[-3 -1] [1 4]]
      setup-row -4 blue [[-4 -2] [2 5]]
      setup-row -5 blue [[-5 -3] [2 6]]
      setup-row -6 blue [[-6 -3] [1 7]]
      setup-row -7 blue [[-7 -2] [0 8]]
      setup-row -8 blue [[-8 1] [3 9]]
      setup-row -9 blue [[-9 0] [3 10]]
      setup-row -10 blue [[-10 -1] [2 11]]
      setup-row -11 blue [[-11 0] [1 12]]
      setup-col 12 blue [[-11 12]]

      setup-col 11 blue [[-10 1] [2 11]]
      setup-col 10 blue [[-9 0] [1 10]]
      setup-col 9 blue [[-8 -1] [0 9]]
      setup-col 8 blue [[-7 -2] [1 8]]
      setup-col 7 blue [[-6 -1] [2 7]]
      setup-col 6 blue [[-5 0] [1 6]]
      setup-col 5 blue [[-4 0] [1 5]]
      setup-col 4 blue [[-3 -1] [2 4]]
      setup-col 3 blue [[-2 0] [1 3]]
      setup-col 2 blue [[-10 -7] [-1 2]]
      setup-col 1 blue [[-11 -8] [-6 -1] [4 6] [7 9]]
      setup-col 0 blue [[-11 -9] [-7 -1] [4 6] [7 9]]
      setup-col -1 blue [[-8 -3] [-1 2]]

      setup-col -2 blue [[-7 -4] [-2 0] [1 3]]
      setup-col -3 blue [[-3 1] [2 4]]
      setup-col -4 blue [[-4 0] [1 5]]
      setup-col -5 blue [[-5 6]]
      setup-col -6 blue [[-6 0] [2 7]]
      setup-col -7 blue [[-7 0] [1 8]]
      setup-col -8 blue [[-8 -1] [2 9]]
      setup-col -9 blue [[-9 0] [1 10]]
      setup-col -10 blue [[-10 11]]
      setup-col -11 blue [[-11 12]]
   ]
end-
```

NetLogo Code 3.9: Code to generate the schematic map of the Chevening House Maze shown in Figure 3.8.

Download free eBooks at bookboon.com

3.8 Summary

This chapter has provided an introduction to the programming language NetLogo. NetLogo is an example of a type of agent-based modelling called agent directed simulation that combines agents with simulation. NetLogo is easy to learn and debug, designed with the inexperienced programmer in mind.

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

- In NetLogo, four types of agents are defined: turtles, patches, links and the observer.
- The Models Library in NetLogo provides a wide range of interesting models and code samples.
- URL links to additional models developed for these books can be found listed at the bottom of each chapter.

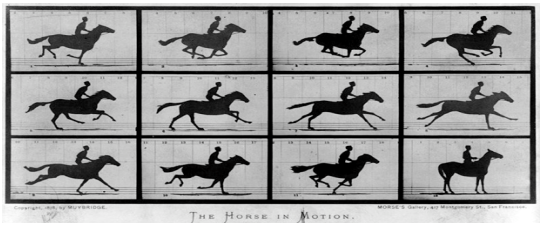
The code for the NetLogo models described in this chapter can be found as follows:

Model	**URL**
Foxes and Rabbits	http://files.bookboon.com/ai/Foxes-and-Rabbits.nlogo
N Dimensional Space	http://files.bookboon.com/ai/N-Dimensional-Space.nlogo
Empty Maze	http://files.bookboon.com/ai/Empty-Maze.nlogo
Hampton Court Maze	http://files.bookboon.com/ai/Hampton-Court-Maze.nlogo
Chevening House Maze	http://files.bookboon.com/ai/Chevening-House-Maze.nlogo

Model	**NetLogo Models Library (Wilensky, 1999) and URL**
Wolf Sheep Predation	Biology > Wolf Sheep Predation http://ccl.northwestern.edu/netlogo/models/WolfSheepPredation
Tumor	Biology > Tumor http://ccl.northwestern.edu/netlogo/models/Tumor
Termites	Biology > Termites http://ccl.northwestern.edu/netlogo/models/Termites

Download free eBooks at bookboon.com

4 Movement



The Horse in Motion by Eadweard Muybridge.

"Change means movement. Movement means friction. Only in the frictionless vacuum of a nonexistent abstract world can movement or change occur without that abrasive friction of conflict."

Saul Alinsky.

Click on the ad to read more

Download free eBooks at bookboon.com

The purpose of this chapter is to highlight how movement is a fundamental task of an autonomous agent situated in an environment, how motion of an agent from an observer's frame of reference can help determine much of its behaviour, and how decision-making behaviour in an agent occurs when an agent makes a choice between alternative paths it can take. The chapter is organised as follows. Section 4.1 defines movement and motion, and emphasizes their importance for agents situated in environments. Section 4.2 describes how to control the movement of turtle agents in NetLogo. Section 4.3 makes the link between movement and behaviour. Section 4.4 describes some methods for representing behaviour using NetLogo link agents to illustrate the importance of movement in defining behaviour. Section 4.5 describes how computer animation provides an illusion of movement. Section 4.6 looks at animated mapping and simulation as a means to represent movement and motion in dynamic environments.

4.1 Movement and Motion

Movement is a fundamental task performed by an autonomous agent situated in an environment. The way an agent moves its body, and the way it chooses to move around its environment, determines much of its behaviour. The types of agent-to-environment interaction and agent-to-agent interaction that are possible are also a result of the types of movement that are possible for the agent.

In physics, *motion* refers to a constant change in the location of a body as a result of some force or forces that are applied to it. A body in motion can be characterised by its velocity, acceleration, displacement and time. A body that does not move is said to be at rest or motionless (stationary). Motion is measured (observed) according to a *frame of reference* based on the motion of an observer. There is no absolute frame of reference – according to a given frame of reference, the body may appear to be motionless, but it is also moving relative to an infinite number of other frames of references. Therefore, everything in the universe can be considered to be in constant motion.

Movement forms the basis of many common metaphors used in human language. Our understanding as humans is derived from fundamental concepts of space, relative position, and movement and motion, including self-motion, which is related to our physical embodiment and situatedness in the real world. We often express and understand abstract concepts which we cannot touch or see based on these fundamental concepts. The list of sample phrases shown in Table 4.1 and the conceptual metaphor on which they are based illustrates how the notions of space and movement/motion are fundamental to the way we think.

Download free eBooks at bookboon.com

Category	Conceptual Metaphor	Sample Phrases
Distance.	AGREEMENT is DISTANCE. FRIENDSHIP is DISTANCE. TIME is DISTANCE.	*That's a long way from what I want.* *We're close friends. He seems distant.* *Long ago and far away.*
Containment (being 'inside' or 'outside' of something).	BELONGING is CONTAINMENT. BELONGING is CONTAINMENT. DIFFICULTIES is a CONTAINER. OBLIGATIONS are CONTAINERS. OBLIGATIONS are CONTAINERS.	*I'm in the army.* *She's an outcast.* *We're in hot water.* *Can you get out of doing the dishes?* *Do you know what you are getting into?*
Movement.	LIFE is a JOURNEY. LIFE is a JOURNEY. NEGOTIATION is a JOURNEY. AGING is MOVING SLOWER. EMOTION is MOTION. EMOTION is MOTION. CHANGE is MOTION (LOCATION). CHANGE is MOTION (LOCATION). CHANGE OF STATE is CHANGE OF DIRECTION. PROGRESS is MOVING FORWARD. PROGRESS is MOVING FORWARD. CREATING is MOVING TO A LOCATION.	*He is on the journey to adulthood.* *She has passed away.* *We're on the road to agreement.* *He's slowing down.* *I was moved.* *Driving me around the bend.* *Faced with the choice, she did a U-turn.* *He went back to polishing the silver.* *When he turned three, he turned into a little monster.* *We're just going around in circles here.* *That's a step in the right direction.* *The created object is brought into being (existence).*

Table 4.1: A selection of phrases to illustrate the importance of location and movement in human language. Examples from Metaphors and Space (2009), and Lakoff (2009).

We will see in Section 4.3 how movement is an important component in agent behaviour.

4.2 Movement of Turtle Agents in NetLogo

Programming the turtle in NetLogo to perform various commands is simple and very intuitive. The command `pen-up` is used to move the drawing pen up, so until the command `pen-down` is called, nothing is drawn on the screen. The methods `pen-size`, and `color` sets the width and colour of the drawing pen. The command `forward` makes the turtle move forward a specified number of steps in the direction it is facing, and `left` and `right` make the turtle turn a specified angle to the left or right respectively. Table 4.2 lists a selection of the methods available, many of which will be used in sample NetLogo code latter on in this book.

Download free eBooks at bookboon.com

Method	Description
	To alter the environment:
clear-all (ca)	Clears everything in the environment, including all turtles, patches, links, plots and drawings and resets variables to 0.
clear-drawing	Clears all drawings made by turtle agents.
	For moving the turtle:
forward d *or* fd d	Moves the turtle ahead the distance d steps based on current direction.
back d *or* bk d	Moves the turtle back the distance d steps based on current direction.
right a *or* rt a	Turns the turtle right the specified angle a in degrees.
left a *or* lt a	Turn the turtle left the specified angle a in degrees.
setxy x y	Moves the turtle from the current position to the absolute position at co-ordinates (*x*,*y*), and draws a line if the pen is down.
set xcor x	Sets the *x* co-ordinate of the turtle to x (keep *y* co-ordinate unchanged).
set ycor y	Sets the *y* co-ordinate of the turtle to y (keep *x* co-ordinate unchanged).
set heading a	Sets the direction a turtle is facing to angle a in degrees.
move-to agent	Sets the (*x*,*y*) co-ordinates to be the same as the agent's.
face agent	Set the heading towards the agent.
facexy x y	Set the heading to the (*x*,*y*) co-ordinates.
	To return or set the turtle's state:
xcor	Built-in turtle variable. It holds *x* co-ordinate of the turtle.
ycor	Built-in turtle variable. It holds *y* co-ordinate of the turtle.
heading	Built-in turtle variable. It holds the current heading of the turtle.
towards agent *or* towardsxy x y	Reports the heading from the current agent towards the given agent or the heading to the co-ordinates (*x*,*y*).
distance agent *or* distancexy x y	Returns the distance to the turtle or patch agent or the distance to the co-ordinates (*x*,*y*).
show-turtle *or* st	Makes the turtle visible.
hide-turtle *or* ht	Makes the turtle invisible.
hidden?	Returns True if the turtle is not visible, False otherwise.
	To control the drawing pen:
pen-down *or* pd	Push the pen down so that the turtle draws when it is moving.
pen-up *or* pu	Pull the pen up so that the turtle does not draw anything when it is moving.
pen-erase *or* pe	Push the eraser down so that the turtle erases lines when it is moving.
pen-mode	Built-in turtle variable. It holds the state of the turtle's pen – values are "up", "down" and "erase".
pen-size	Built-in turtle variable. It holds the width of the turtle's pen in pixels.
color	Built-in turtle variable. It holds the colour of the turtle or link.

Table 4.2: A selection of methods in NetLogo for controlling and visualising turtle movement.

Download free eBooks at bookboon.com

The methods have been grouped into methods related to the environment, and the movement of the turtle, its state, and its drawing pen. The latter three groups of methods affect or determine the behaviour of the turtle. Despite the simplicity of the design, intricate shapes and pictures are very easy to program. Programming with turtle graphics is intuitive because it uses an agent-oriented first-person design perspective – everything is drawn from the perspective of the turtle itself.

4.3 Behaviour and Decision-making in terms of movement

We can adopt a design perspective that any behaviour of an agent can be defined or characterised in terms of its movements it exhibits in an environment. Aspects of an agent's behaviour include how the agent moves, where the agent moves to, when the agent chooses to move, what the agent chooses to move to and why an agent chooses to make a particular move which is related to its decision-making.

Click on the ad to read more

Download free eBooks at bookboon.com

From a design perspective, we can adopt the analogy when designing our agent-orient systems that decision-making is movement from one location of an environment to another location, and is similar to movement in real life. The link between movement between locations or internal state and decision-making is apparent in the metaphors that underlie the meaning behind the phrases listed on the right of Table 4.1. Movement in real life environments clearly results in a change in geographical position. Movement in virtual environments will result in the internal state of the environment being altered in some manner (for example, a game bot's virtual position being altered). Based on this analogy, we can define decision-making behaviour as follows. An agent exhibits decision-making behaviour when it chooses to move along one particular path when alternative paths are available to it. This is analogous to movement in real life.

We can also use the n-dimensional space environment discussed in Section 2.6 to represent decision-making in agents in an abstract way. We can consider the process of making a decision for an agent as moving from one internal state to another where state is represented as a single point in an n-dimensional space. This is equivalent to representing behaviour using a finite state machine (FSM) (also called a finite state automaton, or FSA). A FSM is a model of behaviour with a limited internal memory that can be conceptualised as a directed graph that consists of a number of states (the vertices or nodes in the graph), transitions between those states (the edges in the graph), and actions that are performed upon entering or exiting the state, or during a transition.

A simple FSM for describing car driving behaviour is illustrated in Figure 4.1. There are two states – labelled 'Motor Running' and 'Motor Stopped' – representing when the car's motor is running or stopped respectively, and two transition actions – labelled 'Turn Key Off' and 'Turn Key On' – for moving the FSM from one state to the other.

We will restrict ourselves to actions that occur only during transitions in order to strengthen the analogy between decision-making and motion. We can think of a point in an agent's decision-making environment as analogous to a state in a finite-state machine. We can also think of movement along a path in an agent's decision-making environment as being analogous to making a transition and performing a transition action in a finite state machine.

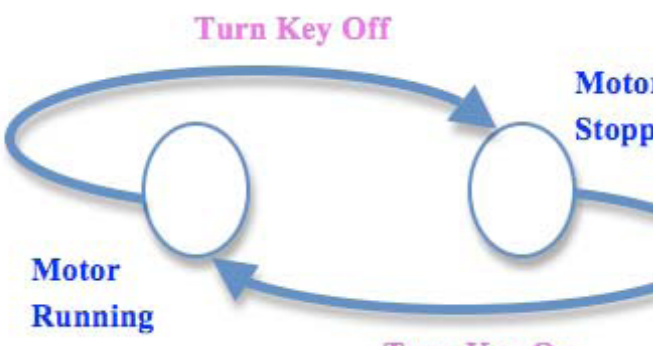


Figure 4.1 A finite state machine (FSM) for representing car driving behaviour.

Download free eBooks at bookboon.com

Decisions where the agent upon reaching a particular point or state has no choice but to perform a particular action are called *deterministic*. Decisions where the agent is able to make a selection between a set of alternative actions are called *non-deterministic*.

4.4 Drawing FSMs and Decision Trees using Link Agents in NetLogo

We can use link agents in NetLogo to draw FSMs. We will also use similar code to draw decision trees, a further method of representing behaviour. This will prove handy latter in this volume, particularly in describing how to use animation to demonstrate dynamic behaviour, and for representing knowledge.

Sample code for visualising a simple two-state FSM is shown in NetLogo Code 4.1. A screenshot is shown in Figure 4.2. The two-state FSM represents what happens when a light needs to be switched on and off. In the first state, the light is off, and to move to the second state, when the light is on, requires an agent to perform the action of moving the light switch down. When the light is on, the opposite action of moving the light switch up is required to move back to the first state again.

Click on the ad to read more

Download free eBooks at bookboon.com

```
breed [agents agent]
breed [points point]
directed-link-breed [curved-paths curved-path]

agents-own [location] ;; holds a point

to setup
   clear-all ;; clear everything

   set-default-shape points „circle”

   create-points 1 [setxy -15 0 set label "Light off "] ;; point 0
   create-points 1 [setxy 15 0 set label "Light on "] ;; point 1

   ask points [set size 5 set color blue]

   ask point 0 [create-curved-path-to point 1]
   ask point 1 [create-curved-path-to point 0]

   ask curved-paths [set thickness 0.5]
   set-default-shape curved-paths "curved path"

   ask patches [
      ;; don’t allow the viewer to see these patches; they are only for
      ;; displaying labels on separate lines
      set plabel-color brown
      if (pxcor = 11 and pycor = 8) [set plabel "Move light switch down"]
      if (pxcor = 11 and pycor = -8) [set plabel "Move light switch up"]
   ]
      create-agents 1 [
      set color lime
      set size 5
      set location point 0 ;; start at point 0
      move-to location
   ]
end
```

NetLogo Code 4.1: Code to set up a two state FSM.

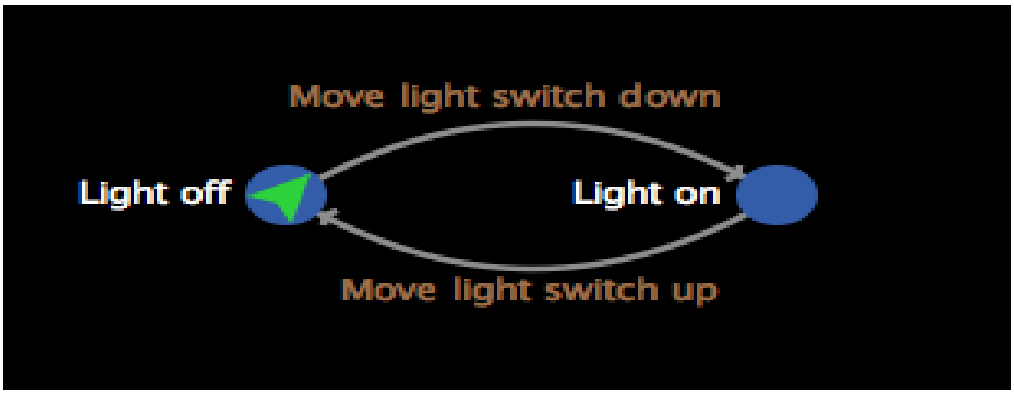


Figure 4.2 A screenshot of the two state FSM created by the Two States model listed in NetLogo Code 4.1.

Download free eBooks at bookboon.com

The FSM drawn by the code consists of two states labelled 'Light off' and 'Light on' respectively. The states are represented as turtle agents that do not move – that is, have a fixed point in the environment (hence these are defined as belonging to the turtle agent breed `points`. There are two links, represented as curved paths, from the first state `point 0` to the second state `point 1`, and then back again. These links are labelled as 'Move light switch down' and 'Move light switch up' respectively, and represents the action the agent needs to take when transitioning from one state to the next.

The code also defines a turtle agent breed called `agent`. We can use this agent to animate the FSM using the code listed in NetLogo Code 4.2, by showing how an agent can move back and forth between the two states. The code works by selecting one of the out links of the current state the turtle agent is visiting (in this example there will always be only one outgoing link). Then the program increases the thickness of the link it will soon move over, and the agent faces the new state's location, and then moves to it.

```
to go
   ask links [ set thickness 0.5 ]
   ask agents [
      let new-location one-of [out-link-neighbors] of location
      ;; change the thickness of the link I will cross over
      ask [link-with new-location] of location [ set thickness 1.1 ]
      face new-location
      move-to new-location
      set location new-location
   ]
   tick
end
```

NetLogo Code 4.2: Code to animate the FSM of Figure 4.2.

A more complex example of a FSM is shown in Figure 4.3. The FSM models the life cycle stages that people go through in their lives, and is based on a model proposed by Jobber (1998). (Jobber's model runs from left to right, whereas the model below runs from top to bottom in order to fit within the confines of the NetLogo 2D visual environment). In the model, a person starts out in the "Single, at home" stage, then may move on to the "Young, on own" stage or the "Young, couple, no children" stage. From there the person can move to a number of further stages, ultimately ending up at the "Solitary, retired" stage at the bottom. The purpose of the model is to illustrate that there are a number of stages that people go through in their lives, and that cycles occur – that is, life is not just a linear progression.

Download free eBooks at bookboon.com

The NetLogo model used to produce this FSM (see URL reference to actual code at bottom of this chapter) animates the model by creating a number of agents each tick (the slider shown on the top left of Figure 4.3 can be used to control the `births-each-tick` variable; in the figure this has been set at 2). Each agent then moves from one state to the other – the lime coloured arrowheads drawn at some of the states show their current location. The agents randomly make a choice when there is more than one outgoing transition available. The plot on the left shows how the overall population (labelled "All") varies with ticks. Also shown on the plot are the current number of agents located at the "Young, parents" stage and at the "Solitary, retired" stages.

Click on the ad to read more

Download free eBooks at bookboon.com

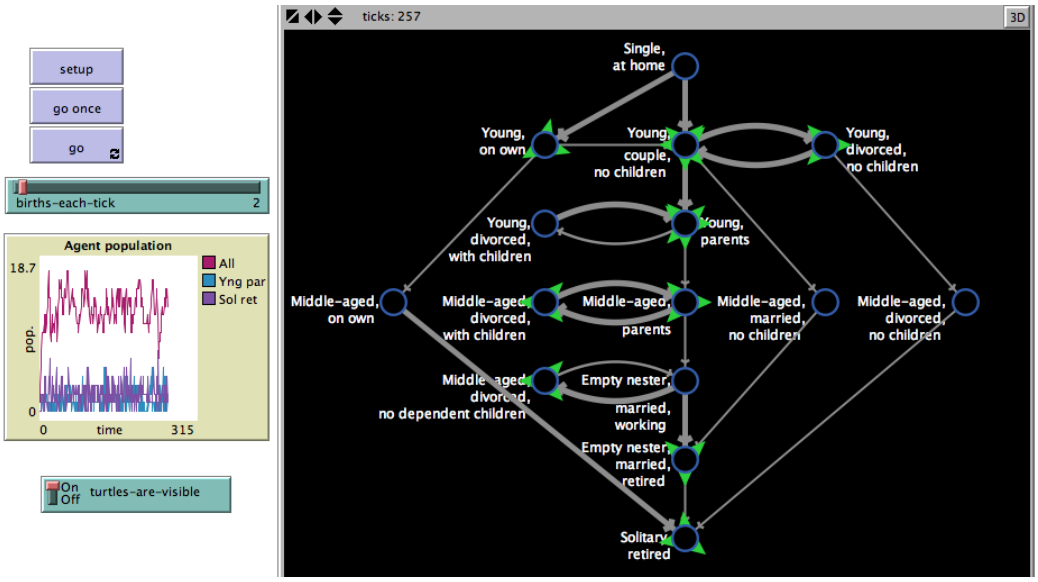


Figure 4.3 A screenshot of the Life Cycle Stages model showing an animated FSM that models life cycle stages (based on Jobber, 1998).

A *decision tree* is often used for representing decision-making behaviour. A *tree* is a directed graph with no loops in the network, with only one path between two nodes, and with each node having one or more children nodes but a single parent node, or if the root node, no parent at all. The leaves of the tree are the nodes with no outgoing paths. In a decision tree, the agent making the decisions starts from the root node of the tree, and chooses the outgoing paths based on a particular decision associated with the current node, and continues until the leaves of the tree are reached. The leaf that is reached usually represents in some manner the outcome of the decisions made along the way – for example, it can be used for classification, where the decision tree operates in a similar way to the popular parlour game Twenty Questions (such as played by characters in the Charles Dickens' novel *A Christmas Carol*). In this game, the first player has to guess an object – usually an animal, vegetable or mineral – that the second player is thinking of. The first player is allowed to ask up to twenty questions about the object in question, while the second player has to truthfully provide only 'Yes' or 'No' answers to the question. The first player has to devise suitable questions in order to narrow the search space sufficiently enough in order to guess what the object in question is before using up their twenty questions, otherwise the game is lost.

An example of a decision tree is shown in Figure 4.4. This decision tree can be used for guessing New Zealand birds. In the Twenty Questions game, this may have been preceded by a set of eight questions and answers as follows: '*Is it an animal*?' [Yes], '*Is it a mammal*?' [No] '*Is it a bird*?' [Yes], '*Is it a Northern Hemisphere bird*?' [No], '*Is it an African or South American bird*?' [No], '*Is it an Australasian bird*?' [Yes], '*Is it an Australian bird*?' [No], '*Is it a New Zealand bird*?' [Yes]. The decision tree asks a further four questions to guess between the eight New Zealand birds listed at the bottom of the tree.

Download free eBooks at bookboon.com

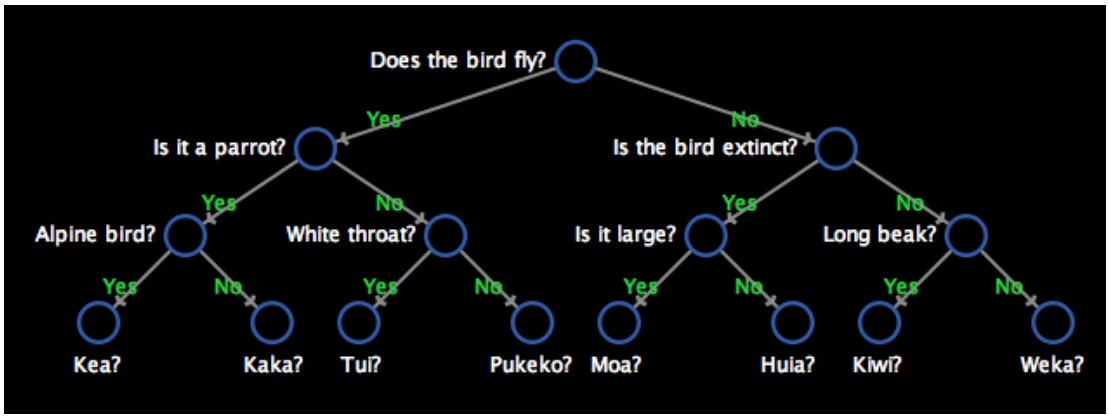


Figure 4.4 A screenshot of the decision tree for classifying New Zealand birds created by the NZ Birds model listed in NetLogo Code 4.3.

We can use NetLogo code similar to that devised for drawing FSMs to also draw decision trees. Each node in the tree can be represented in a similar way to a state in a FSM, with the paths between nodes represented by link transitions. In the code shown in NetLogo Code 4.3 below, there are fifteen states represented by the turtle agents point 0 to point 14 that define nodes in the tree, and fourteen link transitions that define paths between the nodes.

Click on the ad to read more

Download free eBooks at bookboon.com

```
breed [agents agent]
breed [points point]
directed-link-breed [straight-paths straight-path]

agents-own [location] ;; holds a point
points-own [question] ;; this is the question associated with the node
straight-paths-own [answer] ;; the transition paths define answers to the question
to setup
   clear-all ;; clear everything
   set-default-shape points „circle 2”
   create-points 1 [setxy 5 35 set question "Does the bird fly?"]
   ;; point 0, level 0 (root)
   create-points 1 [setxy -25 25 set question "Is it a parrot?"]
   ;; point 1, level 1
   create-points 1 [setxy 35 25 set question "Is the bird extinct?"]
   ;; point 2, level 1
   create-points 1 [setxy -40 15 set question "Alpine bird?"]
   ;; point 3, level 2
   create-points 1 [setxy -10 15 set question "White throat?"]
   ;; point 4, level 2
   create-points 1 [setxy 20 15 set question "Is it large?"]
   ;; point 5, level 2
   create-points 1 [setxy 50 15 set question "Long beak?"]
   ;; point 6, level 2
   create-points 1 [setxy -50 5 set question "Kea?"]
   ;; point 7, level 3
   create-points 1 [setxy -30 5 set question "Kaka?"]
   ;; point 8, level 3
   create-points 1 [setxy -20 5 set question "Tui?"]
   ;; point 9, level 3
   create-points 1 [setxy 0 5 set question "Pukeko?"]
   ;; point 10, level 3
   create-points 1 [setxy 10 5 set question "Moa?"]
   ;; point 11, level 3
   create-points 1 [setxy 30 5 set question "Huia?"]
   ;; point 12, level 3
   create-points 1 [setxy 40 5 set question "Kiwi?"]
   ;; point 13, level 3
   create-points 1 [setxy 60 5 set question "Weka?"]
   ;; point 14, level 3

   ask patches [
      ;; don’t allow the viewer to see these patches; they are only for
      ;; displaying labels on separate lines
      if (pxcor =   1 and pycor = 35) [set plabel [question] of point 0]
      if (pxcor = -29 and pycor = 25) [set plabel [question] of point 1]
      if (pxcor =  30 and pycor = 25) [set plabel [question] of point 2]
      if (pxcor = -44 and pycor = 15) [set plabel [question] of point 3]
      if (pxcor = -14 and pycor = 15) [set plabel [question] of point 4]
      if (pxcor = 16 and pycor = 15) [set plabel [question] of point 5]
      if (pxcor = 46 and pycor = 15) [set plabel [question] of point 6]
      if (pxcor = -48 and pycor = 0) [set plabel [question] of point 7]
      if (pxcor = -27 and pycor = 0) [set plabel [question] of point 8]
      if (pxcor = -18 and pycor = 0) [set plabel [question] of point 9]
      if (pxcor =   4 and pycor = 0) [set plabel [question] of point 10]
```


Download free eBooks at bookboon.com

```
      if (pxcor =  12 and pycor = 0) [set plabel [question] of point 11]
      if (pxcor =  32 and pycor = 0) [set plabel [question] of point 12]
      if (pxcor =  42 and pycor = 0) [set plabel [question] of point 13]
      if (pxcor =  63 and pycor = 0) [set plabel [question] of point 14]
   ]
   ask points [set size 5 set color blue]

   ask point 0 [create-straight-path-to point 1]
   ask point 0 [create-straight-path-to point 2]
   ask point 1 [create-straight-path-to point 3]
   ask point 1 [create-straight-path-to point 4]
   ask point 2 [create-straight-path-to point 5]
   ask point 2 [create-straight-path-to point 6]
   ask point 3 [create-straight-path-to point 7]
   ask point 3 [create-straight-path-to point 8]
   ask point 4 [create-straight-path-to point 9]
   ask point 4 [create-straight-path-to point 10]
   ask point 5 [create-straight-path-to point 11]
   ask point 5 [create-straight-path-to point 12]
   ask point 6 [create-straight-path-to point 13]
   ask point 6 [create-straight-path-to point 14]

   ask straight-paths [set label-color lime]
   ask straight-path 0 1 [set answer "Yes" set label answer]
   ask straight-path 0 2 [set answer "No" set label answer]
   ask straight-path 1 3 [set answer "Yes" set label answer]
   ask straight-path 1 4 [set answer "No" set label answer]
   ask straight-path 2 5 [set answer "Yes" set label answer]
   ask straight-path 2 6 [set answer "No" set label answer]
   ask straight-path 3 7 [set answer "Yes" set label answer]
   ask straight-path 3 8 [set answer "No" set label answer]
   ask straight-path 4 9 [set answer "Yes" set label answer]
   ask straight-path 4 10 [set answer "No" set label answer]
   ask straight-path 5 11 [set answer "Yes" set label answer]
   ask straight-path 5 12 [set answer "No" set label answer]
   ask straight-path 6 13 [set answer "Yes" set label answer]
   ask straight-path 6 14 [set answer "No" set label answer]

   ;; ask points [ create-path-with one-of other points ]
   ;; lay it out so links are not overlapping

   ask straight-paths [ set thickness 0.5 ]
   a
   create-agents 1 [
      set color lime
      set size 5
      set location point 0 ;; start at point 0
      move-to location
   ]
   ask links [ set thickness 0.5 ]
end
```

NetLogo Code 4.3: Code to define and visualise the decision tree in Figure 4.4.

Download free eBooks at bookboon.com

As before, the code defines a turtle agent breed called agent, and we can also use this agent to show how the decision tree is executed depending on the decisions taken using the code listed in NetLogo Code 4.4. The code works by selecting the out link that matches the response provided by the user's answer to the question associated with the current state that the turtle agent is visiting. If a leaf node is reached, and the guess is correct, then the program responds with the user message "Excellent! Let's try another bird." Otherwise it responds with the user messages "Sorry, I cannot guess what the bird is." and "Let's try another bird."

```
to go
   ask agents [
      let neighbours [out-link-neighbors] of location
      let user-question [question] of location
      let answers [] ;; all the answers for current node
      let answers-points []
      ;; (the points where the agent should move to depending on the answer)
      let parent-point location

   ask [out-link-neighbors] of location [
      let this-link in-link-from parent-point
      let this-answer [answer] of this-link
      set answers fput this-answer answers
      set answers-points fput self answers-points
   ]

   ifelse empty? answers
   ;; we are at a leaf node
    [ let user-answer user-one-of user-question ["No" "Yes"]
      ifelse user-answer = "Yes"
        [ user-message "Excellent! Let's try another bird."]
        [ user-message "Sorry, I cannot guess what the bird is."
          user-message "Let's try another bird." ]

      set location point 0 ;; Go back to the beginning
      ask links [ set thickness 0.5 ]
      ;; (reset transitions so it doesn't show path taken)
    ]
   ;;else we are at an internal node
      [ let user-answer user-one-of user-question answers
        let pos position user-answer answers
        let new-location item pos answers-points
        ;; change the thickness of the link I will cross over to show
        ;; the path taken
        ask [link-with new-location] of location [ set thickness 1.1 ]
        face new-location
        move-to new-location
        set location new-location
      ]
   ]
   tick
end
```

NetLogo Code 4.4: Code to animate the execution of the decision tree of Figure 4.4.

Download free eBooks at bookboon.com

Decision trees and FSMs are two ways of representing the behaviour of autonomous agents. If we view behaviour as movement carried out in an abstract environment, then the link between movement and behaviour is clear as has been demonstrated by the common use of basic animation techniques in the NetLogo code examples above. The frame of reference or viewpoint of the observer is also an important aspect of behaviour to consider. For example, if the agent has the ability to make decisions for itself (it has some degree of autonomy) and also has the ability to move, then lack of movement is usually seen to be significant.

There are different types of behaviour, such as communication, searching and reasoning, that we will explore in more detail in latter chapters. We will see that movement of agents in environments is a common component in all of these behaviours.

4.5 Computer Animation

Computer animation is the process of manipulating images being displayed in order to create an illusion of movement. The optical illusion is due to an eye phenomenon called 'persistence of vision' where the eye retains an impression of an image for a short period of time after it has disappeared. Computer animation uses a variety of techniques. A traditional approach is to show a succession of images with each image varying slightly from one image to the next.

Click on the ad to read more

Download free eBooks at bookboon.com

Computer animation is important for Artificial Intelligence research as it can be used to not only develop simulations of computer agents to test out the believability of their behaviour in virtual environments, but also to help build models to predict the behaviour of real-life agents such as humans and robots. Various examples of the use of computer animation for Artificial Life, virtual creatures and game and movie agents will be described in Volume 2 of this book series. We have already used basic computer animation techniques in the NetLogo examples above (see Figures 4.2 to 4.4) to emphasize the link between movement and behaviour.

Figure 4.5 A screenshot of the Shape Animation Example model listed in NetLogo Code 4.5.

The NetLogo Models Library provides a Shape Animation Example model to illustrate how to use shapes to create animations (see Figure 4.5). The Turtles Shapes Editor has been used to create nine different person shapes, and sixteen different flower shapes as shown in Figures 4.6. The model works by successively showing the different shapes from one frame to the next. This provides a primitive form of animation similar to what is produced when using a flick book (also called a flip book), often used in illustrated books for children which contain a series of pictures that vary gradually from one page to the next and which become animate when flicked through rapidly.

Figure 4.6 The nine different person shapes and seven of the sixteen different flower shapes used by the Shape Animation Example model in NetLogo Code 4.5.

Download free eBooks at bookboon.com

The code for the model is listed below in NetLogo Code 4.5. The model defines two breeds – flowers and persons. When the `go` button is pressed, the program calls the animate procedure repeatedly at the speed determined by the global variable `seconds-per-frame`. The animate procedure first calls the procedure `move-person` that simply selects the next person shape to be displayed and moves forward a small amount, then it calls the procedure `age-flower` which sets the shape of each flower turtle to the next shape in the sequence, and then finally it sprouts a new flower 20% of the time the procedure is called.

```
breed [ flowers flower ]
breed [ people person ] ;; putting people last ensures people appear
                        ;; in front of flowers in the view

people-own [frame] ;; ranges from 1 to 9
flowers-own [age] ;; ranges from 1 to 16


to setup                  ;; executed when we press the SETUP button
   clear-all              ;; clear all patches and turtles
   create-people 1 [
      set heading 90      ;; i.e. to the right
      set frame 1
      set shape "person-1"
   ]
end
to go
   every seconds-per-frame [
      animate
      tick
   ]
end
to animate
   move-person
   ask flowers [ age-flower ]
   if random 100 < 20
      [ make-flower ]
end

to move-person
   ask people [
      set shape (word "person-" frame)
      forward 1 / 20
      ;; The shapes editor has a grid divided into 20 squares, which when
      ;; drawing made for useful points to reference the leg and shoe, to
      ;; make it look like the foot would be moving one square backward
      ;; when in contact with the ground (relative to the person), but
      ;; with a relative velocity of 0, when moving forward 1/20th of
      ;; a patch each frame
      set frame frame + 1
      if frame > 9
         [ set frame 1 ] ;; go back to beginning of cycle of animation frames
   ]
end
```


Download free eBooks at bookboon.com

```
to age-flower ;; flower procedure
   set age (age + 1)
      ;; age is used to keep track of how old the flower is
      ;; each older plant is a little bit taller with a little bit
      ;; larger leaves and flower.
   if (age >= 16) [ set age 16 ]
       ;; we only have 16 frames of animation, so stop age at 16
   set shape (word "flowexr-" age)
end

to make-flower
   ;; if every patch has a flower on it, then kill all of them off
   ;; and start over
   if all? patches [any? flowers-here]
      [ ask flowers [ die ] ]
   ;; now that we're sure we have room, actually make a new flower
   ask one-of patches with [not any? flowers-here] [
      sprout 1 [
         set breed flowers
         set shape "flower-1"
         set age 0
         set color one-of [magenta sky yellow]
      ]
   ]
end
```

NetLogo Code 4.5: Code to animate a figure walking through a field of growing flowers.

As a further example, NetLogo Code 4.6 below illustrates how to provide an illusion of a human stick figure walking across the screen as shown in Figure 4.7. The program works by repeatedly drawing six slightly different stick figures in succession, with each figure varying only in the positions of the hands and legs.

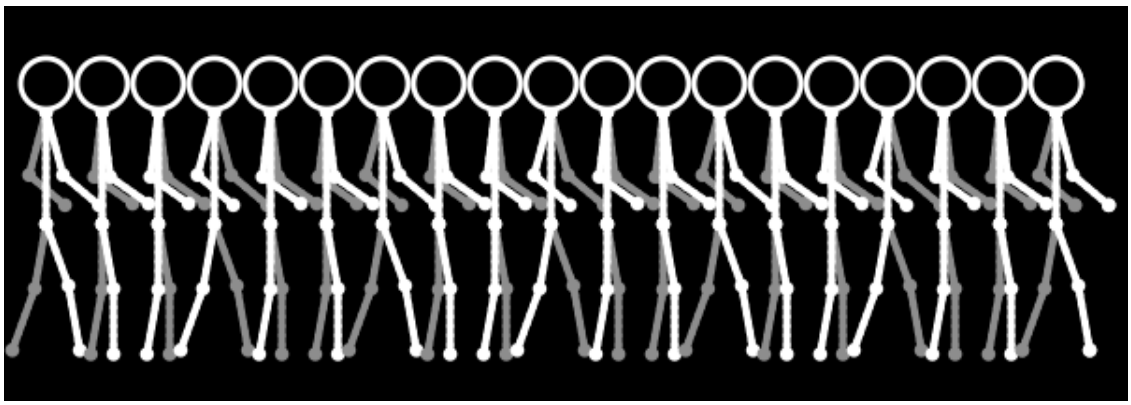


Figure 4.7 A screenshot of a stick figure moving progressively from left to right across the screen created by the Stick Figure Walking model listed in NetLogo Example 4.5.

Download free eBooks at bookboon.com

```
breed [paths path] ;; turtles that draw the limbs and body of the stick
                   ;; figure as a jointed path
breed [circle-paths circle-path] ;; for drawing the head

to setup
   clear-all

   create-paths 5 [initialize-paths] ;; agents to draw paths for four limbs
                                     ;; (two legs, two arms), plus body
   set-default-shape paths "circle" ;; this is the shape of the turtle for
                                    ;; drawing the path

   ifelse forward-movement-increment > 0
      [draw-stick-figure -56 0 [160 7 170 7] [190 7 200 7] [165 7 130 5]
                               [19 7 130 5] [0 0 0 12]]
;;else
      [draw-stick-figure 56 0 [160 7 170 7] [190 7 200 7] [165 7 130 5]
                              [195 7 130 5] [0 0 0 12]]
end

to initialize-paths
   pen-up
   set size 0
   set pen-size 4
end

to initialize-circle-paths
   pen-up
   set size 0
   set pen-size 8
end

to go
   let frame 1
   let xpos forward-movement-increment
   ifelse forward-movement-increment > 0
      [set xpos xpos - 56]
;;else
      [set xpos xpos + 56]

   while [xpos > -57 and xpos < 57]
   [
      if clear-display-in-between
         [clear-drawing]

      let front-leg item (frame mod 6) [[160 7 170 7] [170 7 180 7]
         [180 7 190 7] [190 7 200 7] [180 7 190 7] [170 7 180 7]]
      let back-leg item (frame mod 6) [[190 7 200 7] [180 7 190 7]
         [170 7 180 7] [160 7 170 7] [170 7 180 7] [180 7 190 7]]
      let front-arm item (frame mod 6) [[165 7 130 5] [173 7 125 5]
         [187 7 125 5] [195 7 130 5] [187 7 125 5] [173 7 125 5]]
      let back-arm item (frame mod 6) [[195 7 130 5] [187 7 125 5]
         [173 7 125 5] [165 7 130 5] [173 7 125 5] [187 7 125 5]]
      let body [0 0 0 12] ;; body does not change orientation
```


Download free eBooks at bookboon.com

```
      draw-stick-figure xpos 0 front-leg back-leg front-arm back-arm body

      set frame frame + 1
      set xpos xpos + forward-movement-increment
      tick
   ]
end

to draw-limb [limb-color top-x top-y specs]
   ;; draws a limb for the stick figure according to the specifications
   ;; top of limb is at co-ordinates top-x, top-y
   ;; middle joint is drawn at heading [item 0 specs] with length
   ;; [item 1 specs]
   ;; bottom of limb is drawn at heading [item 2 specs] from the joint with
   ;; length [item 3 specs]

   let dir-to-joint item 0 specs
   let joint-length item 1 specs
   let dir-to-bot item 2 specs
   let bot-length item 3 specs

   pen-up
   set color limb-color
   set size 1.5
   setxy top-x top-y
   pen-down
   stamp
   if joint-length > 0 [
      set heading dir-to-joint
      forward joint-length
      stamp
   ]
   set heading dir-to-bot
   forward bot-length
   stamp
   pen-up
end

to draw-stick-figure [xpos ypos front-leg back-leg front-arm back-arm body]
   ;; draw the stick figure in the following order so that the front leg
   ;; and arms appear in front of the back leg and arms
   ask path 0 [draw-limb gray xpos ypos back-leg] ;; back leg
   ask path 3 [draw-limb gray xpos ypos + 12 back-arm] ;; back arm
   ask path 1 [draw-limb white xpos ypos front-leg] ;; front leg
   ask path 2 [draw-limb white xpos ypos body] ;; body
   ask path 4 [draw-limb white xpos ypos + 12 front-arm] ;; front arm

   create-circle-paths 1 [initialize-circle-paths]
   set-default-shape circle-paths "circle 2"
   ask circle-path 5 [setxy xpos 15 set size 6 set color white stamp]
end
```

NetLogo Code 4.6: Code to make a stick figure 'walk' across the screen.

Download free eBooks at bookboon.com

The program uses two turtle breeds for drawing paths, one using straight lines for drawing the limbs and the body and another using a circular path for drawing the head. The code relies on two core procedures – `draw-limb` and `draw-stick-figure`. The `draw-limb` procedure is used for drawing a limb, either an arm or a leg, by drawing a path consisting of three points – start, middle and end points – with two straight lines connecting the middle point to the other two points. The parameter `limb-color` is used to specify what colour the points and lines are drawn. It can be used to provide a semblance of depth to the stick figure by using a brighter colour (for example, `white`) for the front limbs, and a lighter colour for the back limbs (`gray`). For convenience, the procedure is also used for drawing the body as a single line. The `draw-stick-figure` procedure draws the stick figure by first drawing the back leg and arm, followed by the front leg, then the body and front arm and finally the head. The main `go` procedure repeatedly calls the `draw-stick-figure` procedure with six different specifications that define slightly different positions of the limbs.

The Stick Figure Animation model goes further and allows the user to create multiple stick figures and move them around as shown in Figure 4.8. The program allows the user to record a snapshot of the screen as a 'frame', and then play all the frames back at once in order to create an animation sequence.

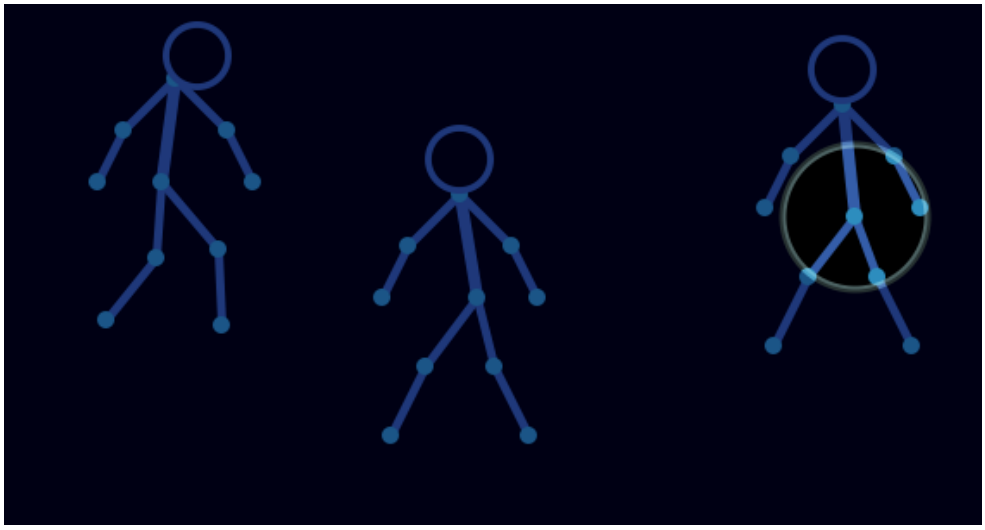


Figure 4.8 A screenshot of the Stick Figure Animation model showing multiple stick figures, with the right one being edited in order to change its body position the next frame.

The model works by recording the frames in a novel way. Rather than store the turtle and link information in each frame separately, new copies of the turtle and link agents are made, and the old copies are then hidden from the viewer. In other words, all the old frames still exist in the environment – the user just can't see them all. Associated with each agent is extra information for storing the frame number when it was recorded. This is then used to determine when each agent should be visualised during the animation sequence. How this is done is shown in NetLogo Code 4.7.

Download free eBooks at bookboon.com

```
breed [points point]       ; points associated with with the stick figure
breed [heads head]         ; for drawing the head
breed [sides side]         ; the four sides of the selection square

turtles-own
[ frame                    ; number of frame the turtle is associated with
  child ]                  ; copy of turtle for next frame
links-own [link-frame]     ; number of frame the link is associated with

globals
[ frames                   ; how many frames so far
  selected ]               ; agentset of currently selected objects
to start-recording
  clear-all
  set frames 0 ; no frames so far

  set-default-shape points "circle" ; this is the shape of the points
  set-default-shape heads "circle 2"
  ; this is the shape of the stick figure's heads
  set-default-shape sides "line"
  ; this is the shape of the selected region sides

  ;; initially, no turtles are selected
  set selected no-turtles
end
to record-this-frame
; records current "frame" by copying all the agents and links,
; making them all invisible, then incrementing their frame number

  let this-child nobody
  let that-child nobody
  let this-colour white
  let this-thickness 1
  let this-frame 0

  ask turtles with [frame = frames]
  [ ; record all turtles in this frame for posterity
    hatch 1 ; make a new copy of this turtle for next frame
    [ set frame frames + 1
      set this-child self ]
    set child this-child
    hide-turtle ; make old copy invisible
  ]

  ask turtles with [frame = frames]

  [ ; record all their links for posterity as well
    set this-child child
    ask my-links
    [ ; copy this link
      set that-child [child] of other-end
      set this-colour color
```


Download free eBooks at bookboon.com

```
        set this-thickness thickness
        ask this-child ; make a new copy of this link for next frame
        [ setup-link that-child this-colour this-thickness (frames + 1) ]
        hide-link ; make old copy invisible
      ]
   ]
   set frames frames + 1
end
to setup-link [ this-turtle colour this-thickness this-frame ]
; for setting up the link

   create-link-with this-turtle
   [
      set link-frame this-frame
      set thickness this-thickness
      set color colour
   ]
end
to play-frames
; creates an animation by playing all the frames that have been
; recorded as invisible turtles and links

   let this-frame 0

   while [this-frame <= frames]
   [ ; display all the frames in sequence
      ask turtles [ hide-turtle ] ; hide all turtles first
      ask links [ hide-link ] ; hide the links as well

      ask turtles with [frame = this-frame]
      [ show-turtle ]
      ask links with [link-frame = this-frame]
      [ show-link ]
      display

      wait frame-delay

      set this-frame this-frame + 1
   ]
end
```

NetLogo Code 4.7: Selected code for animating stick figures for the Stick Figure Animation model.

The code defines three agent breeds: `points`, for defining where the hands, elbows, feet, knees, neck and bottom are for the stick figures; `heads`, for defining where the head is; and `sides`, used for selecting the turtle agents when editing. It then defines the three procedures that are associated with the `start-recording`, `record-this-frame` and `play-frames` buttons at the top left of the model's Interface.

Download free eBooks at bookboon.com

All turtle agents (i.e. heads and points) own two variables: `frame`, which is the frame number when the object was created; and `child`, which is used by the `record-this-frame` procedure to help copy the links between turtle agents. In effect, this is a forward pointer from the parent agent to the child agent when the agents and links are copied during this procedure. It is used to ensure the links between child agents parallel the links between the parent agents once they have been copied. The parent agents are then made invisible.

The `play-frames` procedure then performs the animation by incrementing the frame number, and making visible only those agents and links that have the current frame number, while making everything else invisible. Although the code for doing this is relatively straightforward, this is not the most efficient way of performing the animation – it cycles through all the agents every frame, which may cause the animation to slow down if the number of agents and frames is large. A delay is also added to slow the animation down – this can be controlled by the `frame-delay` slider in the model's Interface.

Click on the ad to read more

Download free eBooks at bookboon.com

The `start-recording` procedure simply clears the environment, sets the default shapes for each of the agents, and sets the global variable `selected` to an empty turtles agentset. This variable is used during editing when the mouse is clicked or dragged in order to show which turtles have been selected for moving around or for deleting. Code for doing this can be found in the model by following the URL link below. Note that this model uses the same code as used for the Map Drawing model that will be described in Section 9.7. In this respect, we can consider each frame in an animation sequence as analogous to a map, with the process of animation being equivalent to animated mapping. This latter topic is discussed in the next section.

4.6 Animated Mapping and Simulation

The purpose of maps is to provide a method of representing what exists in an environment. Many maps, such as topographical maps, are static representations of the environment – they only represent the objects whose positions do not change. However, environments are usually dynamic from the frame of reference of the observer, where positions of agents and objects change constantly with time. Clearly, we require some method to represent agents and objects in motion if we wish to adequately represent the environment when we map it.

One solution developed since the 1950s is 'animated mapping' that combines computer animation with mapping. Animation can be defined as the rapid display of 2D or 3D images in order to create an illusion of scenes of motion. In animated mapping, a map is used to represent the static relationships that exist in the environment, and then this is overlaid with animation to represent the changes that are occurring in the environment over time. The visualisation provides an opportunity to view historic events, and to alter the timescale – either by speeding up the animation so that it is faster than real time, or by slowing it down.

By altering the sequence of events, and trying out different scenarios (that is, asking What-If questions such as "What would happen if this were to happen…?"), then the visualisation moves more into simulation rather than mapping. The role of simulation is to imitate or model natural or human systems, or to determine the possible consequences of a set of actions performed by agents or possible future positions of objects in motion. The main role of mapping is to represent as close as possible pre-existing environmental relationships, but a further role includes using it for predicting what might occur in the future via simulation.

Download free eBooks at bookboon.com

Some examples of animated mapping are the visualisation of the decrease in the Arctic Ice Sheet and the Greenland Ice Sheet in recent years, the spread of civilisation and population growth over the last two centuries, and the predicted paths of hurricanes as often shown on weather channels. Animation provides an opportunity to add the further dimension of time that is difficult to represent in static maps, and include variables such as duration, rate of change and its location, the order events occur, the time at which changes occur, frequency and synchronisation (Slocum, 2005). The use of animated maps on the Internet is increasing where the user has the ability to witness how changes occur over time and where the user can also manipulate various parameters such as the viewpoint, the rate of change, and the type of data that is visualised.

The Models Library in NetLogo also provides some examples of animated mapping. The Grand Canyon model simulates the rainfall on the eastern end of the Grand Canyon (where Crazy Jug Canyon and Saddle Canyon meet to form Tapeats Canyon). Each patch in the simulated environment corresponds to area approximately 32m on each side, and is based on data from the National Elevation Dataset available at http://seamless.usgs.gov. Figure 4.9 provides two screenshots after the model in separate simulations has been executing for 50 ticks and 100 ticks respectively, with the rainfall rate set at 10 drops per tick. The simulation represents higher elevations by lighter colours and lower elevations in darker colours, and shows how the raindrops flow from the lighter down to the darker patches, and start forming pools of water at locations where lower land is surrounded by higher land.

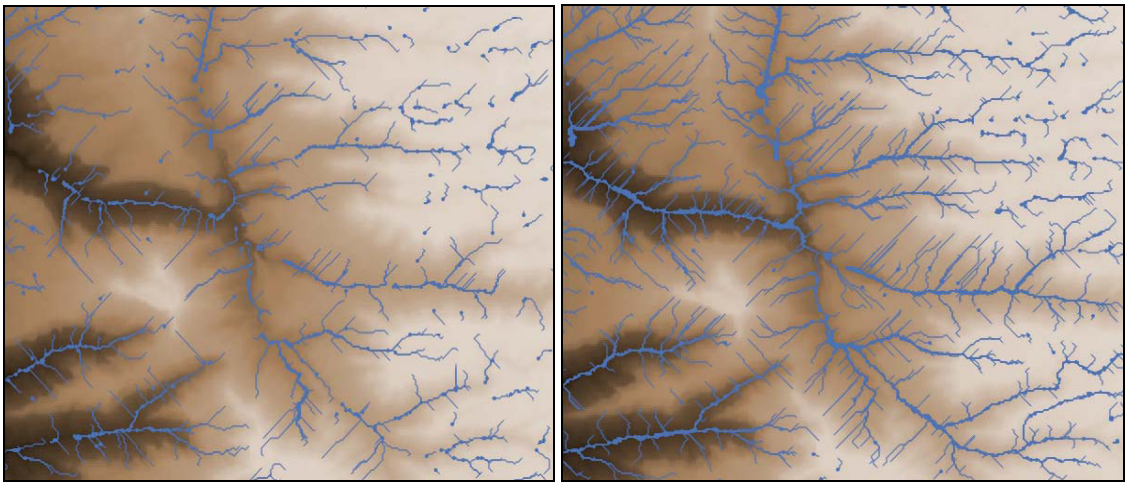


Figure 4.9: NetLogo simulation of rainfall in the Grand Canyon (using the Grand Canyon model).

Another NetLogo model that uses animated mapping is the Continental Divide model. The purpose of the model is to demonstrate how the continental divide is located which separates a continent based on how the rain in separate regions flows into two bodies of water. The example used in the simulation is the North American continent, and the two bodies of water are the Pacific and Atlantic oceans.

Download free eBooks at bookboon.com

Figure 4.10 show four screenshots at different ticks – at 0 ticks (upper left), 407 ticks (upper right), 791 ticks (bottom left) and 1535 ticks (bottom right). In the simulation, the model is initialised with an elevation map, and then both oceans are incrementally risen with the two flood plains gradually converging towards each other over the continent until they eventually crash into each other. Where they crash into each other is where the continental divide is defined to be.

Animated mapping and simulation also provide an opportunity for representing knowledge which is more akin to the situatedness and embodiment approach to Artificial Intelligence that is adopted in these books. Further details of this are provided in Chapter 9. In the next few chapters, we will also explore how we can get basic agents in NetLogo to perform animated movement and motion in simple 2D environments, and to create animated mazes. The maze examples will be used in latter chapters to help explain important types of agent behaviour such as searching. Also, all the NetLogo models described in these books and found in the Models Library can be considered to be examples of animated mapping and simulation.

Click on the ad to read more

Download free eBooks at bookboon.com

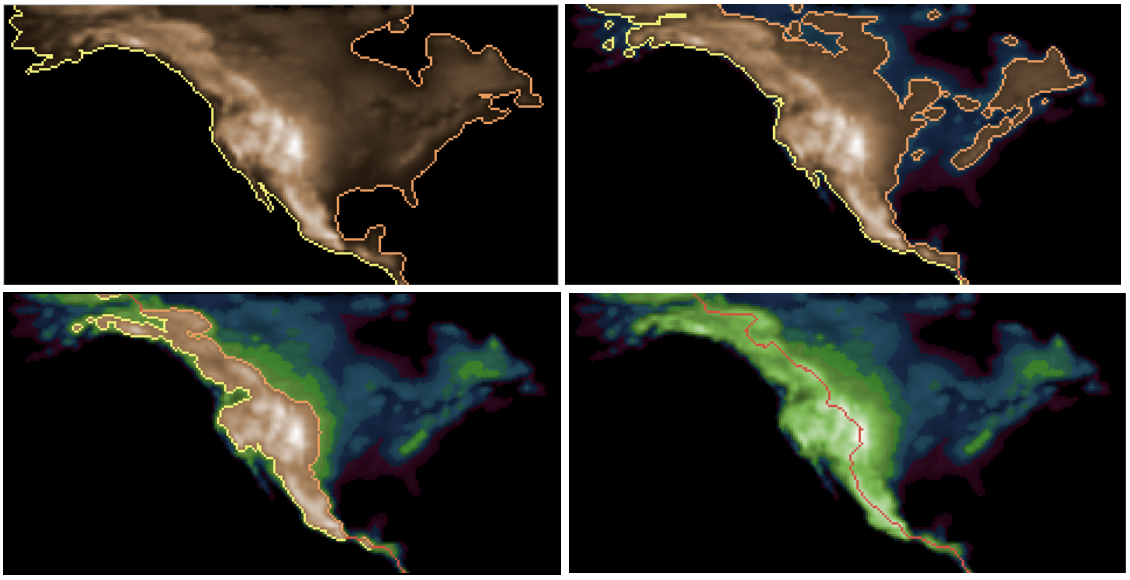


Figure 4.10: Animation showing how the North American continentaldivide is located (using the Continental Divide model).

4.7 Summary

The importance of movement for situated agents has been emphasized. From a design perspective, we can characterize an agent's behaviour and decision-making from an observer's frame of reference in terms of the movements it exhibits. If we are to map environments and behaviour in order to design agent-oriented systems for artificial intelligence, then movement must also be included. Animation of maps presents one way of achieving this.

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

- Movement for an agent concerns how the agent moves its body, and how it chooses to move around the environment.
- Motion refers to a constant change in the location of a body as a result of forces applied to it. Motion is observed from a particular frame of reference. Everything is in constant motion relative to an infinite number of frames of reference.
- The behaviour of a situated agent can be characterised in terms of movement and motion in an environment.
- Computer animation relies on an eye phenomenon called 'persistence of vision' to provide an illusion of movement.
- Animated maps and simulation are useful tools for representing and visualising dynamic environments.

Download free eBooks at bookboon.com

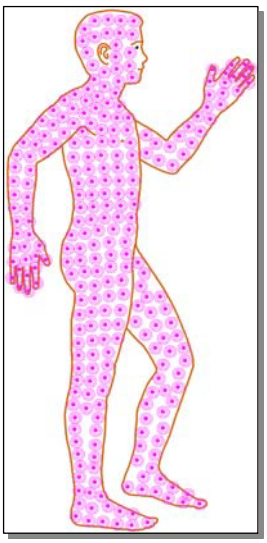
The code for the NetLogo models described in this chapter can be found as follows:

Model	URL
Two States	http://files.bookboon.com/ai/Two-States.nlogo
Life Cycle Stages	http://files.bookboon.com/ai/Life-Cycle-Stages.nlogo
NZ Birds	http://files.bookboon.com/ai/NZ-Birds.nlogo
Stick Figure Animation	http://files.bookboon.com/ai/Stick-Figure-Animation.nlogo
Stick Figure Walking	http://files.bookboon.com/ai/Stick-Figure-Walking.nlogo

Model	NetLogo Models Library (Wilensky, 1999) and URL
Shape Animation Example	Code Examples > Shape Animation Example http://ccl.northwestern.edu/netlogo/models/ShapeAnimationExample
Continental Divide	Earth Science > Continental Divide http://ccl.northwestern.edu/netlogo/models/ContinentalDivide
Grand Canyon	Earth Science > Grand Canyon http://ccl.northwestern.edu/netlogo/models/GrandCanyon

Download free eBooks at bookboon.com

5 Embodiment



You never really understand a person until you consider things from his point of view – until you climb into his skin and walk around in it.

Atticus Finch in '*To Kill a Mockingbird*' by Harper Lee. 1960.

The principle of sensory-motor co-ordination states that all intelligent behavior (e.g. perception, categorization, memory) is to be conceived as a sensory-motor co-ordination… The principle has two main aspects. The first relates to embodiment. Perception, categorization, and memory, processes that up to this point have been viewed from an information processing perspective only, must now be interpreted from a perspective that includes sensory and motor processes… Embodiment places an important role in this co-ordination…

The second, and more specific point of this principle is that through sensory-motor coordination embodied agents can structure their input and thereby induce regularities that significantly simplify learning. "Structuring the input" means that through the interaction with the environment, sensory data are generated, they are not simply given.

Rolf Pfeifer and Christian Scheier. 1999.
Understanding Intelligence. Pages 307–308. The MIT Press.

Download free eBooks at bookboon.com

The purpose of this chapter is to highlight the important role that the body has in determining intelligent behaviour through the body's senses and its interaction with the environment. The chapter is organised as follows. Section 5.1 shows how our body, mind and senses form the basis of many common metaphors used in human language. It also points out that the human body has much more than the five traditional senses of sight, hearing, smell, taste and touch. Section 5.2 describes several important features of autonomous agents – embodiment, situatedness, and reactive versus cognitive behaviour – and shows how they can be classified by these properties. Section 5.3 describes several ways that sensing can be added to turtle agents in NetLogo and defines a common modus operandi for embodied, situated agents. Section 5.4 provides a definition of cognition and perception. It also considers whether it is possible for purely reactive agents to perform non-trivial tasks without cognition, and provides several examples where this is possible. Section 5.5 provides a definition of embodied, situated cognition, an alternative to the traditional information processing view of cognition.

5.1 Our body and our senses

Just as for movement in the previous chapter, our body, mind and senses form the basis of many common metaphors used in human language. Our understanding as humans is derived from fundamental concepts related to our physical embodiment in the real world, and the senses we use to perceive it. We often express and understand abstract concepts that we cannot sense directly based on these fundamental concepts. Table 5.1 provides a list of common phrases that illustrates this.

Click on the ad to read more

Download free eBooks at bookboon.com

Source	Conceptual Metaphor	Sample Phrases
Body or mind.	The MIND is a BODY. MENTAL FITNESS is PHYSICAL FITNESS. MENTAL CONTROL is PHYSICAL CONTROL. COURAGE is a BODY PART. MENTAL PRESSURE is PHYSICAL PRESSURE. EXPERIENCES are related to PHYSICAL BODY FORM. MENTAL EXPERIENCES are related to PHYSICAL TEMPERATURE. MENTAL DISCOMFORT is PAIN.	*His mind is strong and supple.* *His mind is decaying. In the summer, the mind tends to go flabby.* *It's getting out of hand. Get a grip.* *The idea just slipped through my fingers.* *I did not have the heart or stomach for it.* *The pressure was unbearable.* *I'm pressed for time.* *He's light on his feet.* *It is getting quite hairy out there.* *They were in hot pursuit, but the trail had gone cold. She gave him an icy stare.* *She's a real pain. It was a dull ache of a day.*
Sight.	HOPE is LIGHT. BELIEVING is SEEING. SEEING is TOUCHING. INTELLIGENCE is a LIGHT SOURCE. WITHIN SIGHT is WITHIN CONTAINER. EXISTENCE is VISIBILITY. EXISTENCE is VISIBILITY.	*He has bright hopes.* *I can't see how this can be true.* *I felt his glance.* *He is very bright.* *It was well within my field of vision.* *New problems keep appearing.* *The controversy eventually faded away.*
Hearing.	COLOUR is SOUND. BELIEVING is HEARING. THINKING is a CLOCK. UNDERSTANDING is HEARING.	*He was wearing very loud clothes.* *That sounds like the truth to me.* *I could hear his mind ticking.* *Are you deaf? Didn't you hear me?*
Smell.	INVESTIGATING is SMELLING. FEELING is SMELLING.	*The situation didn't smell right.* *That smelt very fishy.*
Taste and eating.	DESIRE is HUNGER. EXPERIENCES is TASTE. GETTING is EATING.	*Sexual appetite. He hungers for her touch.* *That left a sour taste in my mouth.* *Cough up the money you owe me.*
Touch and feeling.	AFFECTION is WARMTH. EMOTION is PHYSICAL CONTACT. DARKNESS is a SOLID.	*She's a warm person.* *I was touched by her speech. That feels good.* *We felt our way through the darkness.*

Table 5.1: A selection of phrases to illustrate the importance of the body, mind and senses in human language; some examples from Metaphors and Touch (2009) and Lakoff (2009).

Traditionally, using a classification first attributed to Aristotle, the five main senses are considered to be sight, hearing, smell, taste and touch (as included in Table 5.1). Sometimes people also refer to a 'sixth sense' (also sometimes called extra sensory perception or ESP) – we can consider the term as a metaphor that people use in natural language for describing unconscious thought processes in terms of another sense. It is a mistake to consider a 'sixth sense' as being real, as there is very little scientific evidence to support it. However, our bodies do in fact have much more than just the five basic senses as shown by the selection in Table 5.2.

Download free eBooks at bookboon.com

Sense	Description
Sight (vision)	The ability of the eye to detect electromagnetic waves. This may in fact be two or three senses as different receptors are used to detect colour and brightness, and depth perception (stereopsis) may be a cognitive function of the brain (i.e. post-sensory).
Hearing (audition)	The ability of the ear to detect vibrations caused by sound.
Smell (olfaction)	The ability of the nose to detect different odour molecules.
Taste	The ability of the tongue to detect chemical properties of substances such as food. This may be five or more different senses as different receptors are used to detect sweet, sour, salty and bitter tastes.
Touch (mechanoreception)	The ability of nerve endings, usually in the skin, to respond to variations in pressure (such as how firm it is, whether it is sustained, or brushing).
Pain (nociception)	The ability to sense damage or near-damage to tissue. There are three types of pain receptors – in the skin (cutaneous), in the joints and bones (somatic) and in the body organs (visceral).
Balance (equilibrioception)	The ability of the vestibular labyrinthine system in the inner ears to sense body movement, direction and acceleration and maintain balance. Two separate senses are involved, to detect angular momentum and linear acceleration (and also gravity).
Proprioception (kinesthetic sense)	This ability to be aware of the relative positions of parts of the body (such as where the hand is in relation to the nose even with your eyes closed).
Sense of time	The ability of part of the brain to keep a track of time, such as circadian (daily) rhythms, or shorter-range (ultradian) timekeeping.
Sense of temperature (thermoception)	The ability of the skin and internal skin passages to detect heat and the absence of heat (cold).
Internal senses (interoception)	Internal senses that are stimulated from within the body from numerous receptors in internal organs (such as pulmonary stretch receptors found in the lungs that control respiratory rate, and other internal receptors that relate to blushing, swallowing, vomiting, blood vessel dilation, gas distension in the gastrointestinal tract and the gagging reflex while eating, for example).

Table 5.2: Human senses.

5.2 Several Features of Autonomous Agents

From a design perspective, we can consider every agent to be embodied in some manner, using senses to gain information about its environment. We can define embodiment in the following manner.

An autonomous agent is embodied through some manifestation that allows it to sense its environment. Its embodiment concerns its physical body that it uses to move around its environment, its sensors that it uses to gain information about itself in relation to the environment, and its brain that it uses to process sensory information. The agent exists as part of the environment, and its perception and actions are determined by the way it interacts with the environment and other agents through its embodiment in a dynamic process.

Download free eBooks at bookboon.com

The agent's embodiment may consist of a physical manifestation in the real world, such as a human, industrial robot or an autonomous vehicle, or it can have a simulated or artificial manifestation in a virtual environment. If it deals exclusively with abstract information, for example a web crawling agent such as Googlebot, then its environment can be considered from a design perspective to be represented by an n-dimensional space, and its sensing capabilities relate to its movement around that space. Likewise, a real environment for a human or robotic agent can be considered to be an n-dimensional space, and the agent's embodiment relates to its ability to gain information about the real environment as it moves around.

We can consider an agent's body as a sensory input-capturing device. For example, the senses in a human body is dictated by its embodiment – that is, the entire body itself has numerous sensors that occur throughout the body, enabling our mind to get a 'picture' of the whole body as it interacts with the environment (see the image at the beginning of this chapter). Pfeiffer and Scheier (1999) stress the important role that embodiment and sensory-motor coordination has to play in determining intelligent behaviour (see quote also at the beginning of this chapter).

Click on the ad to read more

Download free eBooks at bookboon.com

Craig Reynolds (1999) describes further features of autonomous agents that can be used to distinguish various classes of autonomous agents such as situatedness and reactive behaviour. We can define situatedness in the following manner. An autonomous agent is situated within its environment, sharing it with other agents and objects. Therefore its behaviour is determined from a first-person perspective by its agent-to-agent interactions and agent-to-environment interactions. Autonomous agents exhibit a range of behaviours, from purely reactive to more deliberative or cognitive.

An autonomous agent is said to be *situated* in an environment shared by other agents and/or other objects. An agent can be *isolated* existing by itself, for example a data mining agent searching for patterns in a database, situated in an abstract *n*-dimensional space that represents the database environment. A situated agent can have a range of behaviours, from purely *reactive* where the agent is governed by the stimulus it receives from its senses, to *cognitive*, where the agent deliberates on the stimulus it receives and decides on appropriate courses of actions. With a purely reactive approach, there is less need for the agent to maintain a representation of the environment – the environment is its own database that it can simply 'look up' by interacting directly with it. A cognitive agent, on the other hand, uses representations of what is happening in the environment in order to make decisions. (We can think of these as maps as discussed in the previous chapter).

Reynolds notes that combinations of the attributes *embodied, situated* and *reactive* define distinct classes of autonomous agents, some of which are shown in Table 5.3. However, unlike Reynolds, we will consider that *all* agents are both situated in some environment (be it a real, virtual or an abstract environment, represented by some *n*-dimensional space) and embodied with the ability to sense and interact with its environment in some manner, whether explicitly or implicitly.

Type of Agent	Description	Some example(s)
Human and other real life agents	Organic entities that exist in the real world.	Ourselves; biological creatures.
Real autonomous robots	Mechanical devices that are situated in the real world.	Robots in a car manufacturing plant; autonomous vehicles (AVs).
Real semi-autonomous robots	Mechanical devices that are partly autonomous, and partly operated by humans, situated in the real world.	Mars rover robots, Spirit and Opportunity; bomb disposal robots.
Virtual agents	Real agents that are situated in a virtual world.	Non-playing characters (NPCs) in computer games.
Virtual semi-autonomous agents	Agents that are partly autonomous, and partly operated by humans, situated in a virtual world.	Avatars; first-person shooters in computer games; agents in sports simulation games.
Simulated agents	Agents that are studied by computational simulation situated in a virtual world.	Artificial life simulations; simulations of robots.

Table 5.3: Classes of embodied, situated agents (partly based on Reynolds, 1999).

Download free eBooks at bookboon.com

5.3 Adding Sensing Capabilities to Turtle Agents in NetLogo

We can design NetLogo turtle agents introduced in the previous chapter to include sensing capabilities using the embodied, situated design perspective. One reason for doing this is that we would like to use turtle agents to search mazes (and not just to draw them) in order to demonstrate searching behaviour using animated mapping techniques (see next section 5.4 and also Chapter 8). Searching is an important process that an agent must perform in order to adequately carry out many tasks. However, before the agent is able to search, it must first have the ability to sense objects and other agents that exist in the environment in order to find the object or agent it is searching for.

We could easily adopt a disembodied solution to maze searching. In this approach, the turtle agent doing the searching simply follows paths whose movement has been pre-defined within the program (like with the maze drawing models described in the previous chapter). An alternative, embodied approach is that the turtle agent is programmed with the ability to 'look' at the maze environment, interpret what it 'sees', and then choose which path it wants to take, not being restricted to the pre-defined paths that were required for the disembodied solution.

INNOVATIVE
LIKE YOU.

If you're hoping for a truly modern education, one where you're encouraged to speak your mind and to think long-term, both when it comes to your own future and the future of the planet. Then the University of Gothenburg is the place for you.

Study a Master's programme in Gothenburg, Sweden | www.gu.se/education

UNIVERSITY OF GOTHENBURG

Click on the ad to read more

Download free eBooks at bookboon.com

One easily programmed sensing capability often found in computer animation and computer game engines relies on proximity detection. Here the agent 'senses' whether there is an object nearby and then decides on an appropriate behaviour in response. The Look Ahead Example model that comes with the NetLogo Models Library implements turtle agents with a basic form of proximity detection, effectively simulating a rudimentary form of sight by getting the agents to 'look' ahead a short distance in the direction of travel before they move. The process of looking ahead allows each turtle agent to determine what is in front of it, and then change its direction if needed.

A screenshot of the model is shown in Figure 5.1. The image shows five turtle agents following different paths as represented by the red zigzag lines, with an arrowhead showing the turtle's current position and pointing in the current direction of travel. When the turtle detects an obstacle in front of it, it will then change its direction to a new random heading.

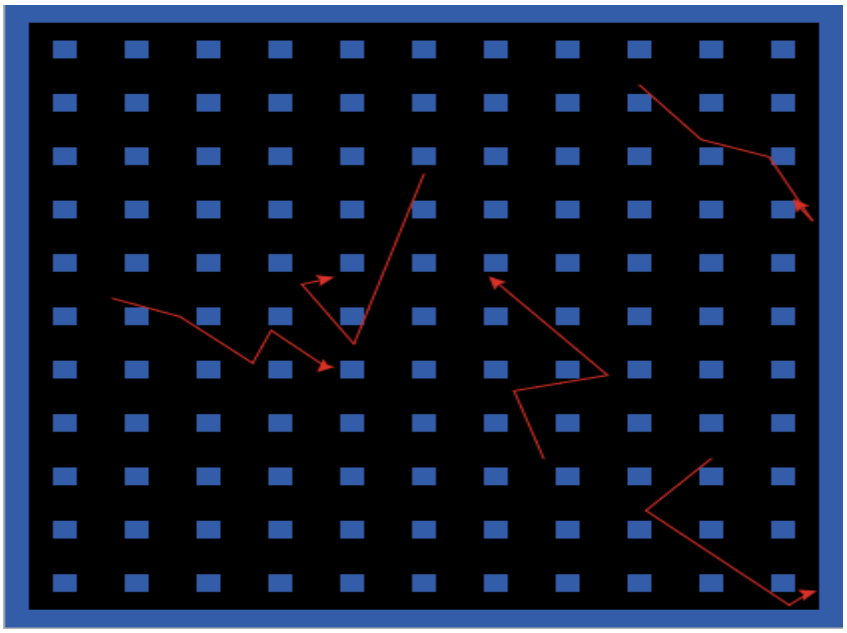


Figure 5.1 A screenshot of the Look Ahead Example 2 model produced by the program listed in NetLogo Code 5.1. The number-of-turtles variable has been set to 5.

The code for the model is shown in NetLogo Code 5.1. The code is slightly modified from that provided in the Models Library, and a `go once` button and a slider to specify the value of the `number-of-turtles` variable has been added to the interface. The `setup` procedure first sets up a checkerboard of blue patches in the environment (these are the objects the turtles are trying to avoid) and the turtles are then sprouted at random non-blue locations. The `go` procedure asks each turtle to check if there is a blue patch ahead, then gets the turtle to change direction to a random heading if there is, otherwise the turtle will move forward 1 step.

Download free eBooks at bookboon.com

```
;; This procedure sets up the patches and turtles
to setup
   ;; Clear everything.
   clear-all
   ;; This will create a "checkerboard" of blue patches. Every third patch
   ;; will be blue (remember modulo gives you the remainder of a division).
   ask patches with [pxcor mod 3 = 0 and pycor mod 3 = 0]
      [ set pcolor blue ]

   ;; This will make the outermost patches blue. This is to prevent the
   ;; turtles from wrapping around the world. Notice it uses the number of
   ;; neighbor patches rather than a location. This is better because it
   ;; will allow you to change the behavior of the turtles by changing the
   ;; shape of the world (and it is less mistake-prone)
   ask patches with [count neighbors != 8]
      [ set pcolor blue ]

   ;; This will create turtles on number-of-turtles randomly chosen
   ;; black patches.
   ask n-of number-of-turtles (patches with [pcolor = black])
   [
      sprout 1
         [ set color red ]
   ]
end
;; This procedure makes the turtles move
to go
   ask turtles
   [
      ;; This important conditional determines if they are about to walk into
      ;; a blue patch. It lets us make a decision about what to do BEFORE the
      ;; turtle walks into a blue patch. This is a good way to simulate a
      ;; wall or barrier that turtles cannot move onto. Notice that we don't
      ;; use any information on the turtle's heading or position. Remember,
      ;; patch-ahead 1 is the patch the turtle would be on if it moved
       ;; forward 1 in its current heading.

      ifelse [pcolor] of patch-ahead 1 = blue
         [ lt random-float 360 ] ;; We see a blue patch in front of us. Turn a
                                 ;; random amount.
         [ fd 1 ]                ;; Otherwise, it is safe to move forward.
   ]
   tick
end
```

NetLogo Code 5.1 Code for the Look Ahead Example 2 model.

If we run the model a while longer (400 ticks or more) with the variable `number-of-turtles` set to 50 instead of 5, then the red paths rapidly cover most of the empty space in the environment as shown in Figure 5.2.

Download free eBooks at bookboon.com

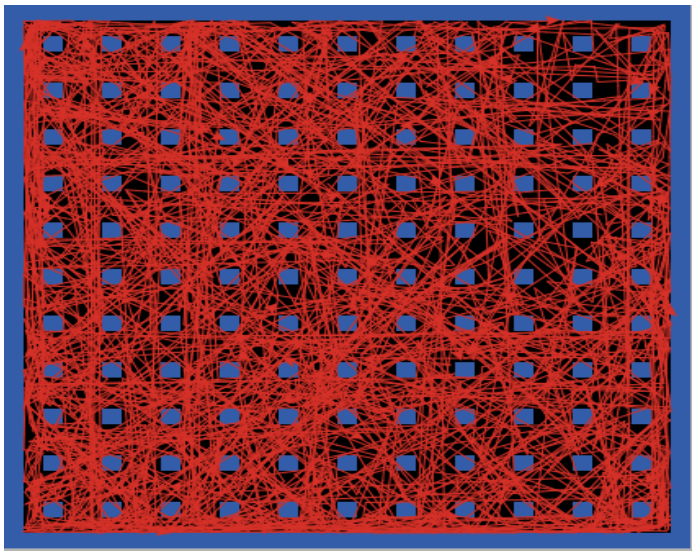


Figure 5.2 A screenshot of the Look Ahead Example 2 model when `number-of-turtles` has been set to 50, after approximately 400 ticks of the simulation.

MASTER OF SCIENCE PROGRAMMES

"The faculty at BI are always ready to guide and motivate students to think outside the box. The opportunity to interact with students from all over the globe contributes to global learning and thinking. BI's infrastructure caters to every student's perceivable need – the state-of-art canteen facilities, gymnasium, study rooms, IT facilities and more".

Althea Pereira, India, MSc in Leadership and Organisational Psychology, 2009

BI Norwegian School of Management (BI) offers a wide range of Master of Science (MSc) programmes in:

- Business & Economics
- Strategic Marketing Management
- International Marketing & Management
- Leadership & Organisational Psychology
- Political Economy
- Financial Economics
- Innovation & Entrepreneurship

For more information, visit www.bi.edu/msc
BI also offers programmes at Bachelor, Masters, Executive MBA and PhD levels.
Visit www.bi.edu for more information.

APPLY NOW

BI NORWEGIAN BUSINESS SCHOOL

EFMD EQUIS ACCREDITED

Click on the ad to read more

Download free eBooks at bookboon.com

This example shows how a relatively modest number of agents with very limited sensing capabilities still have the ability to completely cover an entire environment very rapidly. Such ability for a group of real-life agents is often a very useful trait, for example insects such as ants that need to find food sources. From an observer's frame of reference when running the model, one can easily come to the wrong conclusion that the agents are deliberately searching their environment, but an analysis of the NetLogo code reveals that the agents in this example are simply reactive rather than cognitive – they do not deliberately search the environment, but as a result of their simple combined behaviour, achieve the same effect that agents with more deliberate searching behaviour achieve through their ability to co-ordinate and combine results.

With proximity detection, we can also simulate sensors for 'touching' as well as 'looking'. The Wall Following Example model in the NetLogo Models Library approximates the action of touching for an agent by having it closely follow an object in the environment. A screenshot of the model is shown in Figure 5.3. The paths of the turtles are shown in the figure since the `pen-down` button has been selected in the model's interface. The behaviour the turtle agents exhibit is called 'wall' following as it is analogous to a real-life agent keeping close contact with the outside of a building similar to the hand on the wall behaviour for solving mazes described in Chapter 3. The 2D NetLogo environment shown in the image can be thought of as a map that provides a bird's-eye view of an imaginary 3D environment containing buildings, with the edges of the objects in the image representing where the outside walls of the building are.

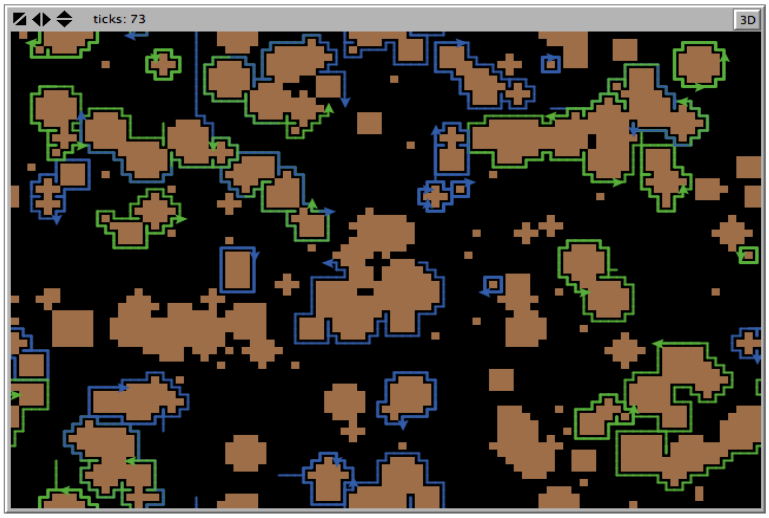
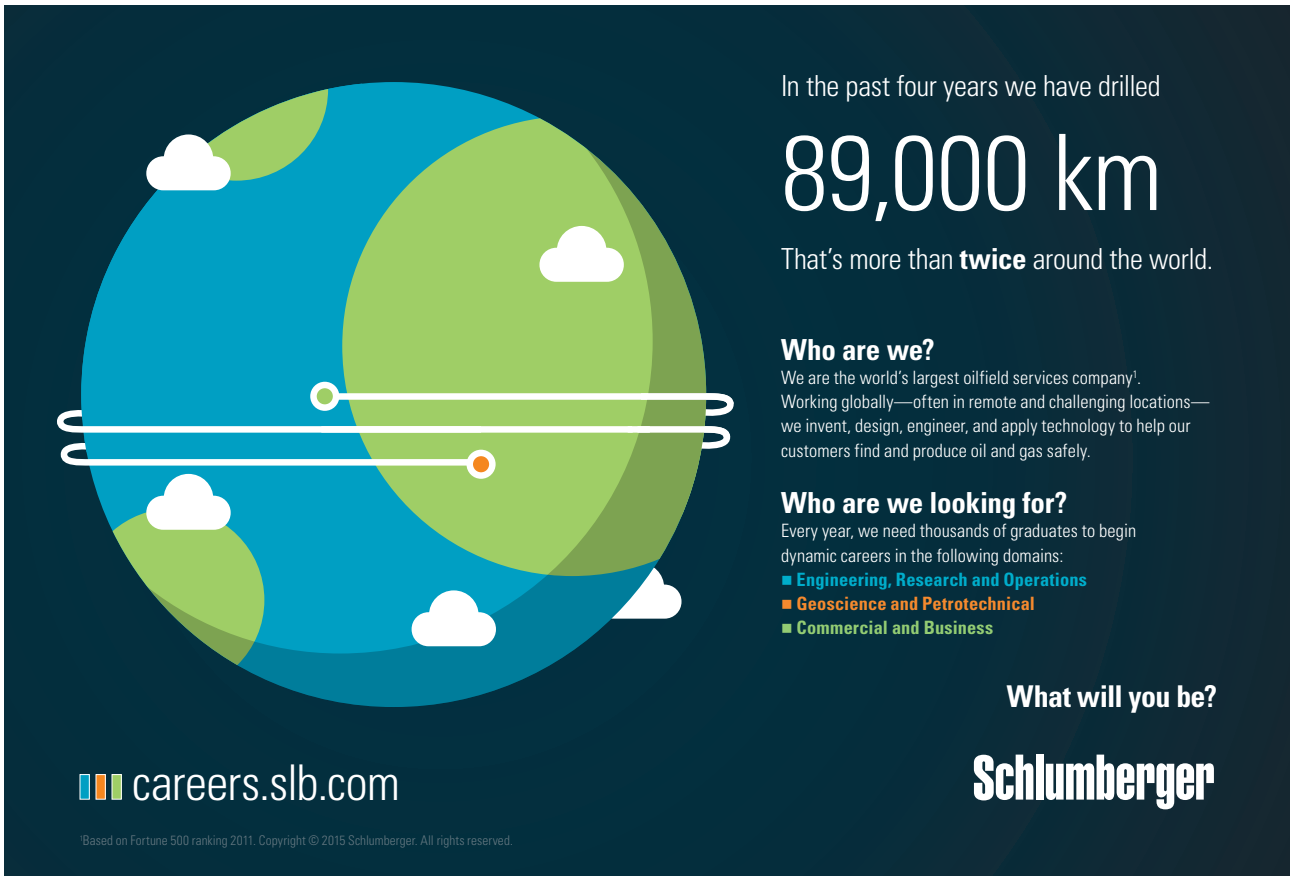


Figure 5.3: The Wall Following Example model demonstrates how turtle Agents can follow walls using a virtual 'sense' of touch.

Download free eBooks at bookboon.com

In the image, the blue turtles follow the right hand wall whereas the green turtles follow the left hand wall. The turtles prefer to keep the wall they are following on a particular side of their body – for blue turtles, they prefer the right side; green turtles prefer the left side. The code for the model is shown in NetLogo Code 5.2.

The procedure walk defines the walking behaviour of the turtle agents. Each agent has a `direction` variable that defines the side it prefers to keep the wall on. If there is no wall immediately on the preferred side, but there is a wall behind it on the preferred side, then the turtle agent must turn to the preferred side so that it does not lose the wall. If there is a wall directly in front, then the turtle agent keeps turning to the side opposite to the preferred side until it can find some open space in front of it where it can then move forward.



Click on the ad to read more

Download free eBooks at bookboon.com

```
turtles-own [direction]    ;; 1 follows right-hand wall,
                           ;; -1 follows left-hand wall

to setup
   clear-all
   ;; make some random walls for the turtles to follow.
   ;; the details aren't important.
   ask patches [ if random-float 1.0 < 0.04 [ set pcolor brown ] ]
   ask patches with [pcolor = brown]
   [ ask patches in-radius random-float 3 [ set pcolor brown ] ]
   ask patches with [count neighbors4 with [pcolor = brown] = 4]
   [ set pcolor brown ]
   ;; now make some turtles. SPROUT puts the turtles on patch centers
   ask n-of 40 patches with [pcolor = black] [
      sprout 1 [
         if count neighbors4 with [pcolor = brown] = 4
         [ die ] ;; trapped!
         set size 2 ;; bigger turtles are easier to see
         set pen-size 2 ;; thicker lines are easier to see
         face one-of neighbors4 ;; face north, south, east, or west
         ifelse random 2 = 0
            [ set direction 1 ;; follow right hand wall
              set color blue ]
            [ set direction -1 ;; follow left hand wall
              set color green ]
      ]
   ]
end
to go
   ask turtles [ walk ]
   tick
end
to walk ;; turtle procedure
      ;; 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-report wall? [angle] ;; turtle procedure
 ;; note that angle may be positive or negative. if angle is
 ;; positive, the turtle looks right. if angle is negative,
 ;; the turtle looks left.
 report brown = [pcolor] of patch-right-and-ahead angle 1
end
```

NetLogo Code 5.2: Code for the Wall Following Example model shown in Figure 5.3.

Download free eBooks at bookboon.com

Both the Look Ahead Example model and Wall Following model demonstrate a common modus operandi for embodied, situated agents that consists of first sensing followed by behaviour selection and execution. This modus operandi is one possible method of operation for a reactive agent that consists of the agent first sensing what is happening in the environment, and then choosing to execute certain behaviours based on what it senses.

For the Look Ahead Example model, the turtle agent first senses if there is a blue patch ahead, and then chooses between turning and moving forward behaviours based on what it finds out. For the Wall Following Example model, the agent first senses whether it has lost the wall, and responds by turning if necessary. Then the agent repeatedly senses if there is a wall in front of it and then it turns, until it finally senses there is no longer a wall in front of it, after which it moves forward.

The sensory apparatus of the agent is an important consideration in determining the resultant possible behaviours of the agent. If the agent has no sensing capabilities, it has no ability to react to its environment. Similarly, if the sensing capabilities are restricted in some respect, then the agent will not be able to respond to events that occur in the environment that is outside the range or ranges it can sense.

For example, for vision, cats can see in low light because of special eye muscles, some snakes have special organs and bats have node sensors that allow them to detect infrared light, and some birds can see in the ultraviolet down to 300 nanometres. Humans, in contrast, have a narrower vision range than these examples that restricts and defines our ability to respond to what we see.

Agent-environment interaction also has an important role to play in determining what the agent senses. This is illustrated by the Line of Sight Example model in the NetLogo Models Library. A screenshot of the model is shown in Figure 5.4.

Download free eBooks at bookboon.com

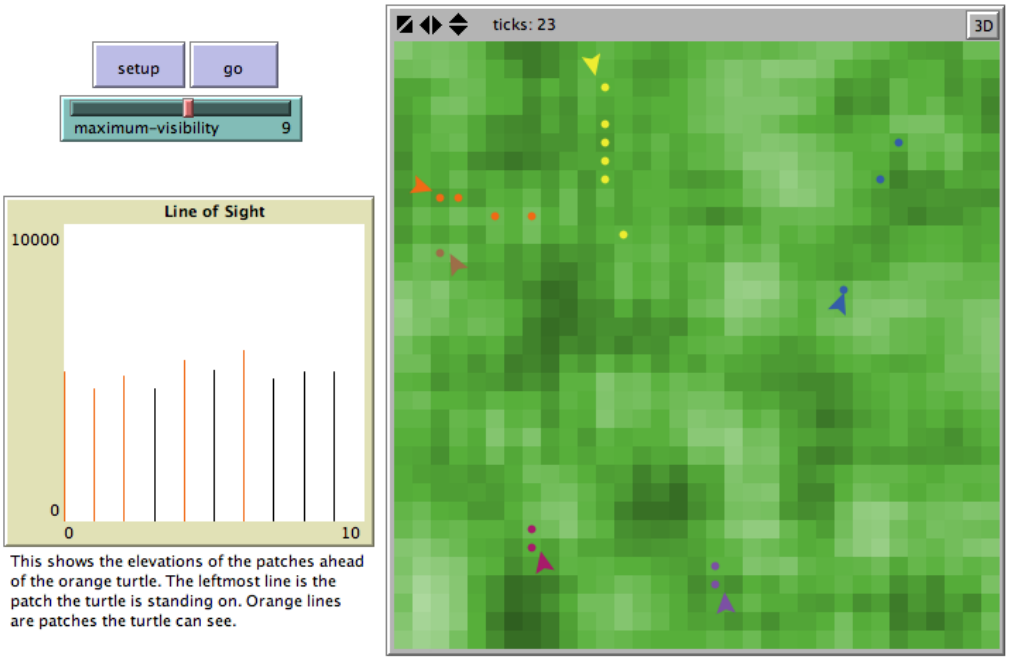
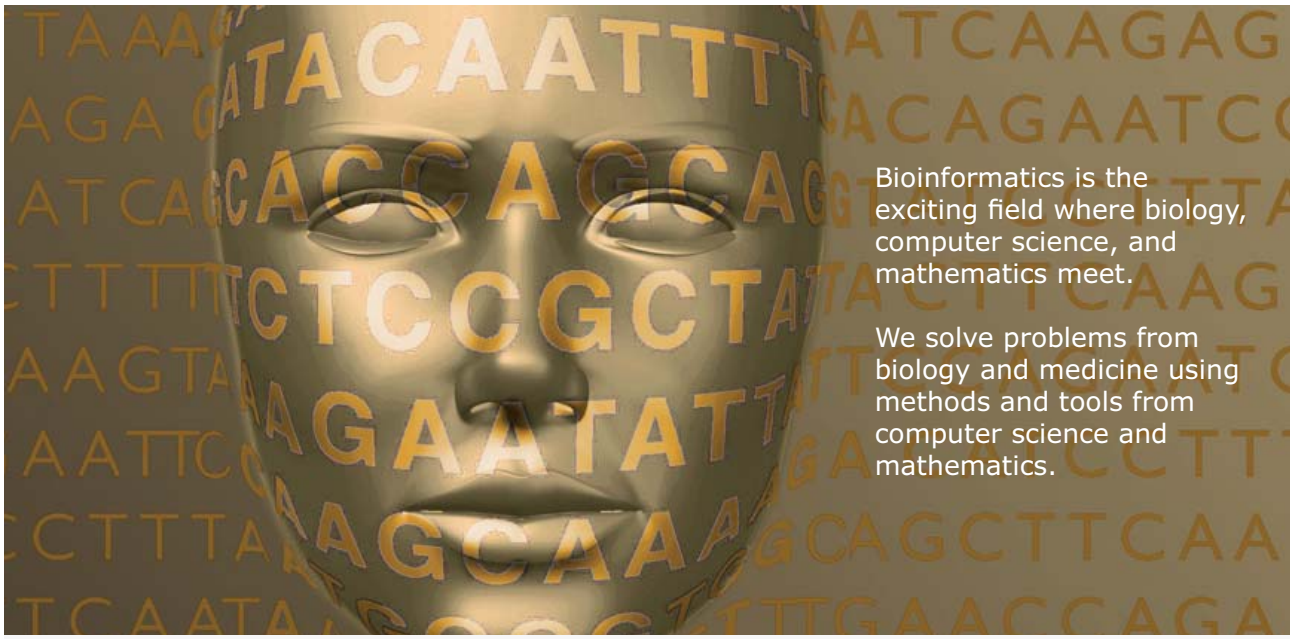


Figure 5.4: A screenshot of the Line of Sight Example 2 model.

UPPSALA UNIVERSITET

Develop the tools we need for Life Science
Masters Degree in Bioinformatics



Read more about this and our other international masters degree programmes at www.uu.se/master

Click on the ad to read more

Download free eBooks at bookboon.com

The image shows some turtle agents moving on random headings, with their current line of sight drawn in dots with the same colour as the colour of the agent. The environment is a virtual representation of a landscape, using a representation often used in computer graphics called a heightmap.

The green shading of each patch reflects its elevation – the lighter the shading, the higher the elevation of the patch. For example, a prominent peak occurs in the lower left corner of the 2D environment; the dark green region immediately to the peak's right indicates a deep depression.

The line of sight is computed from the first person perspective of each agent. Given the agent's current heading and location, a straight line is projected forward by determining if the patches ahead of the agent are not obscured by anything at higher elevations along the way. In the left hand of the figure, a plot is shown of what the orange agent can 'see' in front of it. The leftmost vertical column in the plot shows the elevation of the patch the orange agent is standing on. The vertical columns to the right show the elevations of the patches in front of the agent according to its current heading. As the next two patches are lower than the agent's current elevation, then these have been coloured orange to indicate that the agent will be able to 'see' that part of the terrain. The view of the third patch, however, is obscured by the higher elevation of the second patch, so this has been coloured black to indicate that the agent would not be able to 'look' directly at that part of the landscape. Similarly, viewing of what is at the fourth and sixth patches is possible as they are not obscured by higher elevations of any intervening patches, but the other patches are all obscured from the line of sight.

As an analogy to a real-life situation, consider a human trying to locate a control point (e.g. a red flag) in an orienteering event in sand-dune terrain. Often the red flag will be obscured from the person's line of sight until the person is quite close to the flag, especially if it is placed in a depression. However, as the person moves along in the terrain getting closer to the flag, it will become visible when the person reaches a vantage point that has no intervening hills or higher sloping terrain blocking the view.

This example provides an illustration of the importance of using the situated first-person perspective for the design of embodied agents. The resultant behaviour of the agents is more believable from an observer's frame of reference, and the agent-environment interaction adds to the realism.

The code for the model is shown in NetLogo Code 5.3.

Download free eBooks at bookboon.com

```
breed [walkers walker]
breed [markers marker]
patches-own [elevation]

to setup
   ca
   set-default-shape markers "dot"
   ;; setup the terrain
   ask patches [ set elevation random 10000 ]
   repeat 2 [ diffuse elevation 1 ]
   ask patches [ set pcolor scale-color green elevation 1000 9000 ]
   create-walkers 6 [
      set size 1.5
      setxy random-xcor random-ycor
      set color item who [orange blue magenta violet brown yellow]
      mark-line-of-sight
]
end

to go
   ;; get rid of all the old markers
   ask markers [ die ]
   ;; move the walkers
   ask walkers [
      rt random 10
      lt random 10
      fd 1
      mark-line-of-sight
   ]
   ;; plot the orange walker only
   ask walker 0 [ plot-line-of-sight ]
   tick
end

to mark-line-of-sight ;; walker procedure
   let dist 1
   let a1 0
   let c color
   let last-patch patch-here
   ;; iterate through all the patches
   ;; starting at the patch directly ahead
   ;; going through MAXIMUM-VISIBILITY
   while [dist <= maximum-visibility] [
      let p patch-ahead dist
      ;; if we are looking diagonally across
      ;; a patch it is possible we'll get the
      ;; same patch for distance x and x + 1
      ;; but we don't need to check again.
      if p != last-patch [
```


Download free eBooks at bookboon.com

```
        ;; find the angle between the turtle's position
        ;; and the top of the patch.
        let a2 atan dist (elevation - [elevation] of p)
        ;; if that angle is less than the angle toward the
        ;; last visible patch there is no direct line from the turtle
        ;; to the patch in question that is not obstructed by another
        ;; patch.
        if a1 < a2
           [ ask p [ sprout-markers 1 [ set color c ] ]
             set a1 a2 ]
        set last-patch p
      ]
      set dist dist + 1
   ]
end
```

NetLogo Code 5.3: Code for the Line of Sight Example 2 model shown in Figure 5.4.

Click on the ad to read more

Download free eBooks at bookboon.com

In the main `go` method, the turtle agents perform a small random right then left turn (this will change their heading slightly), they move forward 1 step and then they mark out their line of sight by performing the `mark-line-of-sight` procedure. This procedure iterates through all the patches in front of the agent out to the maximum distance as defined by the interface variable `maximum-visibility`. For each distinct patch along the line of the agent's current heading, it first finds the angle between the turtle's position and the top of the patch. If the angle to the last visible patch is less than this angle, then its view of the patch is not obstructed and it draws a marker in the environment to show that it can see the patch in question.

The range of vision can be expanded to include a cone rather than a single line by using the built-in NetLogo reporter `in-cone`. This reporter reports the set of agents that fall within the cone of vision defined by the distance and viewing angle parameters. The cone is centred around the turtle's current heading. If the angle is 360°, then the results is equivalent to the reporter `in-radius`. The Vision Cone Example model provided by the NetLogo Models Library demonstrates the use of this reporter, as shown by the screenshots in Figure 5.5. The left screenshot in the figure was obtained by the setting the Interface variables `vision-radius` to 50 and `vision-angle` to 40; the right screenshot was obtained by setting those variables to 20 and 360 respectively.

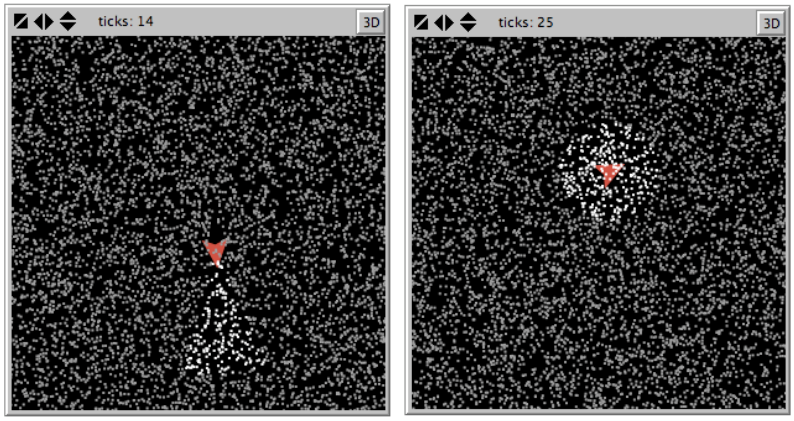


Figure 5.5: Two screenshots from the Vision Cone Example model showing the effect of the in-cone reporter when different parameters are used to define the distance and angle of the cone.

The code for this model is shown in NetLogo Code 5.4. The model uses two types of agents – a `wanderer` breed for the big red agent that explores the environment with its vision cone sense as shown in the screenshots; and a `stander` breed that is used to draw 6000 randomly spread gray agents in the environment in the `setup` procedure.

Download free eBooks at bookboon.com

```
breed [ wanderers wanderer ] ;; big red turtle that moves around
breed [ standers stander ] ;; little gray turtles that just stand there

to setup
   clear-all
   ;; make a background of lots of randomly scattered
   ;; stationary gray turtles
   create-standers 6000
   [
      setxy random-xcor random-ycor
      set color gray
   ]
      ;; make one big red turtle that is going to move around
      create-wanderers 1
      [
         set color red
         set size 15
      ]
      ;; make the vision cone initially visible
      go
end

to go
   ask standers [ set color gray ]
   ask wanderers
   [
      rt random 20
      lt random 20
      fd 1
      ;; could use IN-CONE-NOWRAP here instead of IN-CONE
      ask standers in-cone vision-radius vision-angle
      [
         set color white
      ]
   ]
   tick
end
```

NetLogo Code 5.4: Code for the Vision Cone Example 2 model shown in Figure 5.5.

In the main `go` method, as for the Line of Sight example, the turtle agents perform a small random right, then a left turn, followed by moving forward 1 step. Then setting the colour of all standers in the vision cone defined by the `vision-radius` and `vision-angle` variables shows their current field of vision.

Download free eBooks at bookboon.com

Sensing in agents need not be restricted to the traditional senses such as vision, or touch. For example, equilibrioception (see Table 5.2), or the sense of balance, combined with other senses such as the visual system and proprioception, allows humans and animals to gain information about their body position in relation to their immediate surroundings. Sensing can involve the detection of any aspect of the environment, such as the presence, absence or change in a particular attribute. This then provides the agent with an ability to follow a gradient in the environment relating to the attribute. For example, a useful sense for real-life agents is the ability to detect changes in elevation, in order to be able to head in an upwards or downwards direction when required.

The Hill Climbing Example model in the NetLogo Models Library shows how such a sense can be implemented. A screenshot of the model is shown in Figure 5.6 and the code is shown in NetLogo Code 5.5. The screenshot shows turtle agents that start out at random locations then move steadily upwards to the highest local point in their immediate surrounding landscape (i.e. towards the patches with the lightest shading). Since a single high point serves as the highest point for many of its surrounding patches, this results in the agents' paths coalescing into linear patterns of lines as shown in the figure. The vertical line to the left of the figure, for example, represents the top of a ridgeline in the 2D environment. As an analogy with real-life human agents, we can imagine a number of hill walkers climbing from different start positions up a ridge, and once they reach the ridge line, they all end up following a single path to the top.

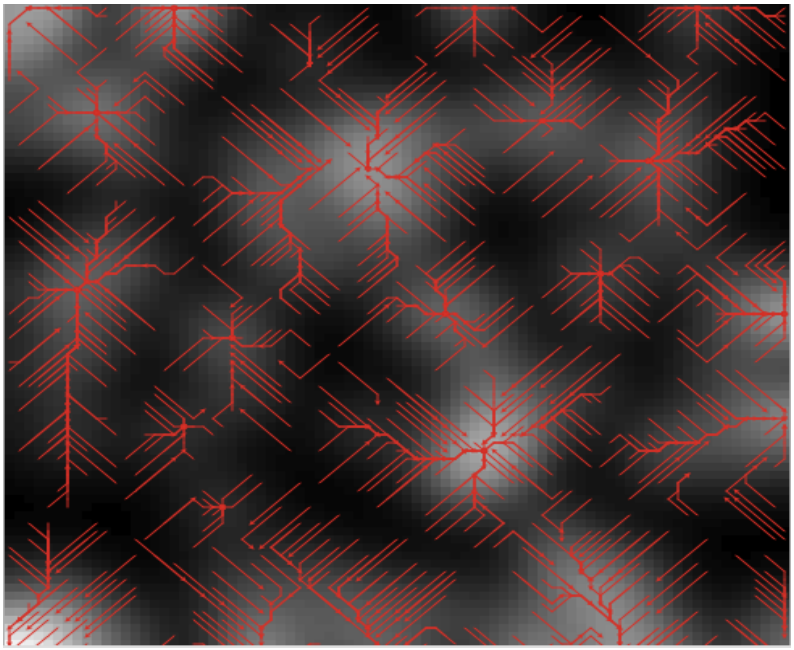


Figure 5.6: Screenshot of the Hill Climbing Example model.

The model uses the `uphill` command in NetLogo that directs an agent to move to a neighbouring patch that has the highest value for a particular patch variable. The variable in this case is the patch's `color`.

Download free eBooks at bookboon.com

The `go` method asks each turtle to head in an upward direction, until that is no longer possible, in which case the turtle variable `peak?` is set to true to indicate the turtle has reached the top. When all turtles have reached the top, then no more processing is done.

```
turtles-own
[
   peak? ;; indicates whether a turtle has reached a "peak",
         ;; that is, it can no longer go "uphill" from where it stands
]

to setup
   clear-all
   ;; make a landscape with hills and valleys
   ask n-of 100 patches [ set pcolor 120 ]
   ;; slightly smooth out the landscape
   repeat 20 [ diffuse pcolor 1 ]
   ;; put some turtles on patch centers in the landscape
   ask n-of 800 patches [
      sprout 1 [
         set peak? false
         set color red
         pen-down
      ]
   ]
end
to go
   ;; stop when all turtles are on peak
   if all? turtles [peak?]
      [ stop ]
   ask turtles [
      ;; remember where we started
      let old-patch patch-here
      ;; to use UPHILL, the turtles specify a patch variable
      uphill pcolor
      ;; are we still where we started? if so, we didn't
      ;; move, so we must be on a peak
      if old-patch = patch-here [ set peak? true ]
   ]
   tick
end
```

NetLogo Code 5.5: Code for the Hill Climbing Example 2 model shown in Figure 5.6.

Download free eBooks at bookboon.com

5.4 Performing tasks reactively without cognition

Is it possible for an agent to perform a non-trivial task without knowing they are performing it? In other words, can an agent 'solve' a problem without cognitively being aware they are solving a problem? To answer this question, first we must ask what we mean by 'cognitively being aware'. Cognition refers to the mental processes of an intelligent agent, either natural or artificial, such as comprehension, reasoning, decision-making, planning and learning. It can also be defined in a broader sense by linking it to the mental act of knowing or recognition of an agent in relation to a thought it is thinking or action it is performing. Thus, cognitive behaviour occurs when an agent knowingly processes its thoughts or actions.

Under this definition, an autonomous agent exhibits cognitive behaviour if it *knowingly* processes sensory information, makes decisions, changes its preferences and applies any existing knowledge it may have while performing a task or mental thought process. Cognition is also related to perception. Perception for an agent is the mental process of attaining understanding or awareness of sensory information.

Let us now return to the question posed at the beginning of this section. Consider an insect's abilities to forage for food. Ants, for example, have the ability to quickly find food sources, but do this by sensing chemical scent laid down by other ants, and then react accordingly using a small set of rules. Despite not being cognitively aware of what they are doing, they achieve the goal of fully exploring the environment, efficiently locating nearby food sources and returning back to the nest.

ericsson.
com

Shaping tomorrow's world – today
Our business is at the heart of a connected world – a world where communication is empowering people, business and society. Our networks, telecom services and multimedia solutions are shaping tomorrow. And this might just be your chance to shape your own future.

It's a people thing
We are looking for high-caliber people who can see the opportunities, people who can bring knowledge, energy and vision to our organization. In return we offer the chance to work with cutting-edge technology, personal and professional development, and the opportunity to make a difference in a truly global company.

We are currently recruiting both new graduates and experienced professionals in four areas: **Software, Hardware, Systems** and **Integration & Verification**.

Are you ready to shape **your** future? Begin by exploring a career with Ericsson. Visit www.ericsson.com/join-ericsson

ERICSSON

Click on the ad to read more

Download free eBooks at bookboon.com

An Ants model provided in the NetLogo Models Library simulates one way how this might happen. A screenshot of the model is shown in Figure 5.7. It shows a colony of ants spread out throughout the environment, with its nest shown by the purple region at the centre of the image. Originally, there were three food sources, but at this stage in the simulation, the previous two have already been devoured and the remaining food source has been found and is in the process of being collected for return to the nest. The white and green shaded region represents how much chemical scent has been load down in the environment, with white representing the greatest amount.

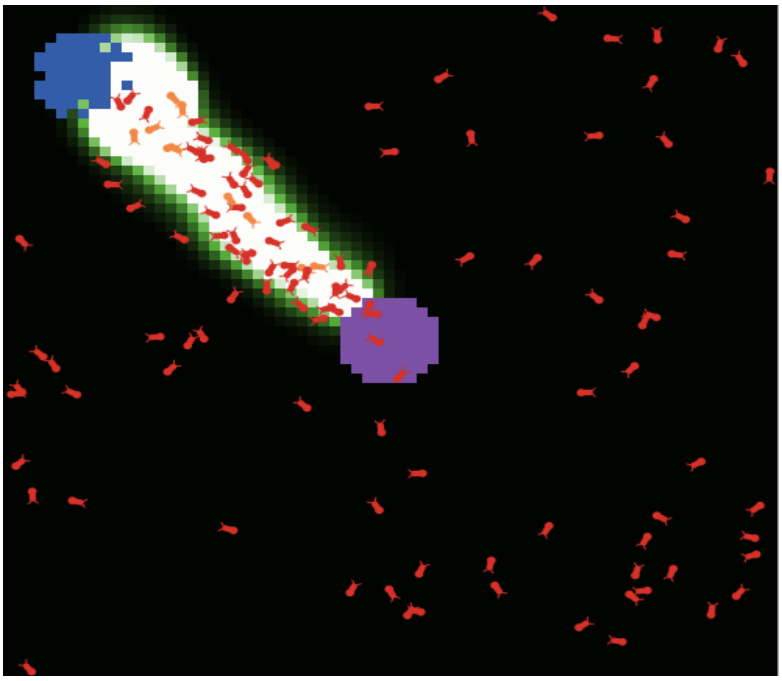


Figure 5.7: Screenshot of the Ants model.

The code for the part of the model that defines the behaviour of the ants is shown in NetLogo Code 5.6. A full listing of the code can be found by selecting the Ants model from the Models Library, and then by clicking on the `Procedures` button at the top of the NetLogo interface.

Download free eBooks at bookboon.com

```
patches-own [
   chemical                ;; amount of chemical on this patch
   food                    ;; amount of food on this patch (0, 1, or 2)
   nest?                   ;; true on nest patches, false elsewhere
   nest-scent              ;; number that is higher closer to the nest
   food-source-number      ;; number (1, 2, or 3) to identify the food sources
]

to recolor-patch ;; patch procedure
   ;; give color to nest and food sources
   ifelse nest?
   [ set pcolor violet ]
   [ ifelse food > 0
      [ if food-source-number = 1 [ set pcolor cyan ]
        if food-source-number = 2 [ set pcolor sky ]
        if food-source-number = 3 [ set pcolor blue ] ]
      ;; scale color to show chemical concentration
      [ set pcolor scale-color green chemical 0.1 5 ] ]
end

to go ;; forever button
   ask turtles
   [ if who >= ticks [ stop ]    ;; delay initial departure
     ifelse color = red
     [ look-for-food ]           ;; not carrying food? look for it
     [ return-to-nest ]          ;; carrying food? take it back to nest
     wiggle
      fd 1 ]
   diffuse chemical (diffusion-rate / 100)
   ask patches
   [ set chemical chemical * (100 - evaporation-rate) / 100
      ;; slowly evaporate chemical
      recolor-patch ]
   tick
   do-plotting
end

to return-to-nest ;; turtle procedure
   ifelse nest?
   [ ;; drop food and head out again
     set color red
     rt 180 ]
   [ set chemical chemical + 60;; drop some chemical
     uphill-nest-scent ]         ;; head toward the greatest value of nest-scent
end

to look-for-food ;; turtle procedure
```


Download free eBooks at bookboon.com

```
   if food > 0
   [ set color orange + 1 ;; pick up food
     set food food - 1 ;; and reduce the food source
     rt 180 ;; and turn around
     stop ]
   ;; go in the direction where the chemical smell is strongest
   if (chemical >= 0.05) and (chemical < 2)
   [ uphill-chemical ]
end

;; sniff left and right, and go where the strongest smell is
to uphill-chemical ;; turtle procedure
   let scent-ahead chemical-scent-at-angle 0
   let scent-right chemical-scent-at-angle 45
   let scent-left chemical-scent-at-angle -45
   if (scent-right > scent-ahead) or (scent-left > scent-ahead)
   [ ifelse scent-right > scent-left
      [ rt 45 ]
      [ lt 45 ] ]
end

;; sniff left and right, and go where the strongest smell is
to uphill-nest-scent ;; turtle procedure
   let scent-ahead nest-scent-at-angle 0
   let scent-right nest-scent-at-angle 45
   let scent-left nest-scent-at-angle -45
   if (scent-right > scent-ahead) or (scent-left > scent-ahead)
   [ ifelse scent-right > scent-left
      [ rt 45 ]
      [ lt 45 ] ]
end

to wiggle ;; turtle procedure
   rt random 40
   lt random 40
   if not can-move? 1 [ rt 180 ]
end

to-report nest-scent-at-angle [angle]
   let p patch-right-and-ahead angle 1
   if p = nobody [ report 0 ]
   report [nest-scent] of p
end

to-report chemical-scent-at-angle [angle]
   let p patch-right-and-ahead angle 1
   if p = nobody [ report 0 ]
   report [chemical] of p
end
```

NetLogo Code 5.6: Code defining the reactive behaviour of the ant agents in the Ants model shown in Figure 5.7.

Download free eBooks at bookboon.com

Each patch agent has a number of variables associated with it – for example, `chemical` stores the amount of chemical that ants have laid down on top of it, and `nest-scent` reflects how close the patch is to the nest. The `recolor-patch` procedure shows how the patch's colour is reshaded according to how much chemical has been laid down on top it. The `go` procedure defines the behaviour of the ants. If the ant is not carrying food, it will look for it (by performing the `look-for-food` procedure) otherwise it will take the food back to the nest (by performing the `return-to-nest` procedure). As the agent is returning to the nest, it drops a chemical as it moves. The ants use a sense of smell to virtually 'sniff' this chemical to guide them towards the food source. As more and more ants carry the food back to the nest, they reinforce the strength of the chemical, but this will also become diffused over time as the strength of the chemical is reduced each tick in the `go` procedure.

Download free eBooks at bookboon.com

The two procedures `uphill-chemical` and `uphill-nest-scent` define the ants' reactive behaviour in following the chemical scent to the food source and in returning to the nest. The ants are essentially applying similar behaviour to the hill climbing turtle agents for the Hill Climbing Example model above (which follow the gradient defined by the patch variable `pcolor` in relation to the 2D environment's terrain). The ant agents in the Ants model on the other hand follow the gradient defined by the patch variable `chemical` if they are looking for food, and return along the gradient defined by the patch variable `nest-scent` if they are returning to the nest. They do this by 'sniffing' first left, then right, then following the path with the strongest smell. The wiggle procedure provides a random element to the movement of the ants that ensures that they will be very effective at exploring the entire environment similar to the turtle agents in the Look Ahead Example 2 model depicted in Figure 5.2.

In this example, many agents employing simple reactive behaviour are able to get a non-trivial task done simply as a side effect of their interaction with each other and with the environment. If a single agent were to exhibit the same ability – that of fully exploring its environment for food, and finding its way back home once it has found some – then we as observers might attribute some degree of intelligence to the agent. The ants use a form of *collective* intelligence – the intelligence comes from the collective result of many agents interacting with each other.

Is it also possible for a single reactive agent (rather than a collection of agents) to successfully complete a non-trivial task without knowing it is doing so? For example, can a single agent get through a maze by employing simple reactive behaviour when it has no ability to recognize that there are alternative paths to explore? A model called Mazes has been developed to demonstrate how this is possible. Screenshots of the model are shown in Figures 5.8 to 5.10. They show turtle agents using simple reactive behaviours to move around the three mazes that were defined in Chapter 3 – the empty maze, the Hampton Court Palace Maze and the Chevening House maze.

Note that we have to be careful how we discuss the behaviour and actions of the turtle agent in the model. We cannot say that the agent is 'exploring' the maze – exploration assumes some degree of volition on the part of the agent doing the exploring. Similarly, we cannot say that the agent has 'solved' the maze when it has reached the centre or reached the exit. Solving requires cognition – that is, recognition of the task at hand. All we can say is that the agent moves around the maze, and in so doing, it manages to reach some state (such as the centre or exit) as a side effect of its behaviour.

The model provides an interface chooser called `turtle-behaviour` that allows us to define the reactive behaviour of the turtle agent as it moves around the maze. There are four behaviours defined: `Hand On The Wall`, `Random Forward 0`, `Random Forward 1`, and `Random Forward 2`. The NetLogo code defining these behaviours is shown in NetLogo Code 5.7.

Download free eBooks at bookboon.com

```
to walk ;; turtle procedure
   ;; turn right if necessary
   ifelse set-pen-down [ pen-down ] [ pen-up ]

   if count neighbors4 with [pcolor = blue] = 4
      [ user-message "Trapped!"
        stop ]
   let xpos [goal-x] of maze 0
   if xcor >= xpos and xcor <= xpos + [goal-width] of maze 0 and
      ycor = [goal-y] of maze 0
      [ ifelse [goal-type] of maze 0 = "Exit"
         [ user-message "Found the exit!" ]
         [ user-message "Made it to the centre of the maze!" ]
         stop ]

   if (turtle-behaviour = "Hand On The Wall") [behaviour-wall-following]
   if (turtle-behaviour = "Random Forward 0") [behaviour-random-forward-0]
   if (turtle-behaviour = "Random Forward 1") [behaviour-random-forward-1]
   if (turtle-behaviour = "Random Forward 2") [behaviour-random-forward-2]
end

t'to-report wall? [angle dist]
   ;; note that angle may be positive or negative. if angle is
   ;; positive, the turtle looks right. if angle is negative,
   ;; the turtle looks left.
   let patch-color [pcolor] of patch-right-and-ahead angle dist
   report patch-color = blue or patch-color = sky
   ; blue if it is a wall, sky if it is the closed entrance
end
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

to behaviour-random-forward-0
; moves forward unless there is a wall, then tries to turn left, then right
; then randomly turns as a last resort
   if wall? 0 1
      [ ifelse wall? 90 1
         [ lt 90 ]
         [ ifelse wall? 270 1
           [ rt 90 ]
           [ ifelse random 2 = 0 [lt 90] [rt 90] ]]]

      ;; move forward
      fd 1
end
```


Download free eBooks at bookboon.com

```
to behaviour-random-forward-1
; moves around mostly in straight lines
    ifelse wall? 0 1
       [ bk 1 ]
   ;else
       [ let r random 20
       ifelse r = 0
          [ if not wall? 90 1 [rt 90 fd 1] ]
       ;else
         [ ifelse r = 1
            [ if not wall? 270 1 [lt 90 fd 1] ]
            [ let f random 5
               while [not wall? 0 1 and f > 0]
                  [ fd 1
                  set f f - 1]]]]
end

to behaviour-random-forward-2
; moves around in random small steps
   let r random 3
   ifelse r = 0
      [ if not wall? 90 1 [rt 90 fd 1] ]
      [ ifelse r = 1
         [ if not wall? 270 1 [lt 90 fd 1] ]
         [ if not wall? 0 1 [fd 1] ]]
end
```

NetLogo Code 5.7: Code defining the reactive behaviour of the turtle agent that moves around the mazes shown in Figures 5.8 to 5.10.

The `walk` procedure defines how the turtle agent moves around the maze. The agent has the ability to sense whether there is a wall nearby (using a rudimentary sense based on proximity detection), and this is defined by the `wall?` reporter in the code. The code then defines the four different types of behaviour. The `behaviour-wall-following` procedure defines classic 'hand on the wall' behaviour – the agent tries to keep its hand on the wall similar to the turtle agents in the Wall Following Example depicted in Figure 5.3. For the `behaviour-random-forward-0` procedure, the turtle agent first senses if there is a wall, then it tries to turn left, then right, then randomly turns as a last resort before moving forward 1 step. For the `behaviour-random-forward-1` procedure, the turtle agent moves around mostly in straight lines unless blocked by a wall. For the `behaviour-random-forward-2` procedure, the turtle agent moves around in random small steps.

The definitions of these behaviours are relatively simple, but several of them produce very complex movements as a result. Nowhere in the definitions, however, is there any cognition on the part of the agent – the agent executing these behaviours is not knowingly doing so, aware in the knowledge that a certain action or actions will lead to a desired result.

Download free eBooks at bookboon.com

Figure 5.8 shows how the different turtle behaviours 'explore' the empty maze. The `Hand On The Wall` behaviour results in the turtle agent following either the left hand wall or the right hand wall (as in the figure) depending on the random choice the agent makes immediately after entering the maze. The turtle agent will very quickly reach the exit of the maze at the top. In contrast, the Random `Forward 0` behaviour will result in the turtle agent never managing to find the exit to the maze. It will repeatedly follow the walls in either an anti-clockwise direction (as in the figure) if the agent turns immediately right after first entering the maze, or in a clockwise direction if it first turns left. However, it will never manage to get out of the endless loop it is trapped in. Such a fruitless result is typical for a reactive agent. It cannot figure out that it is going wrong, and then adjust its behaviour accordingly in order to reach a desired state – it just simply reacts.

Click on the ad to read more
Download free eBooks at bookboon.com

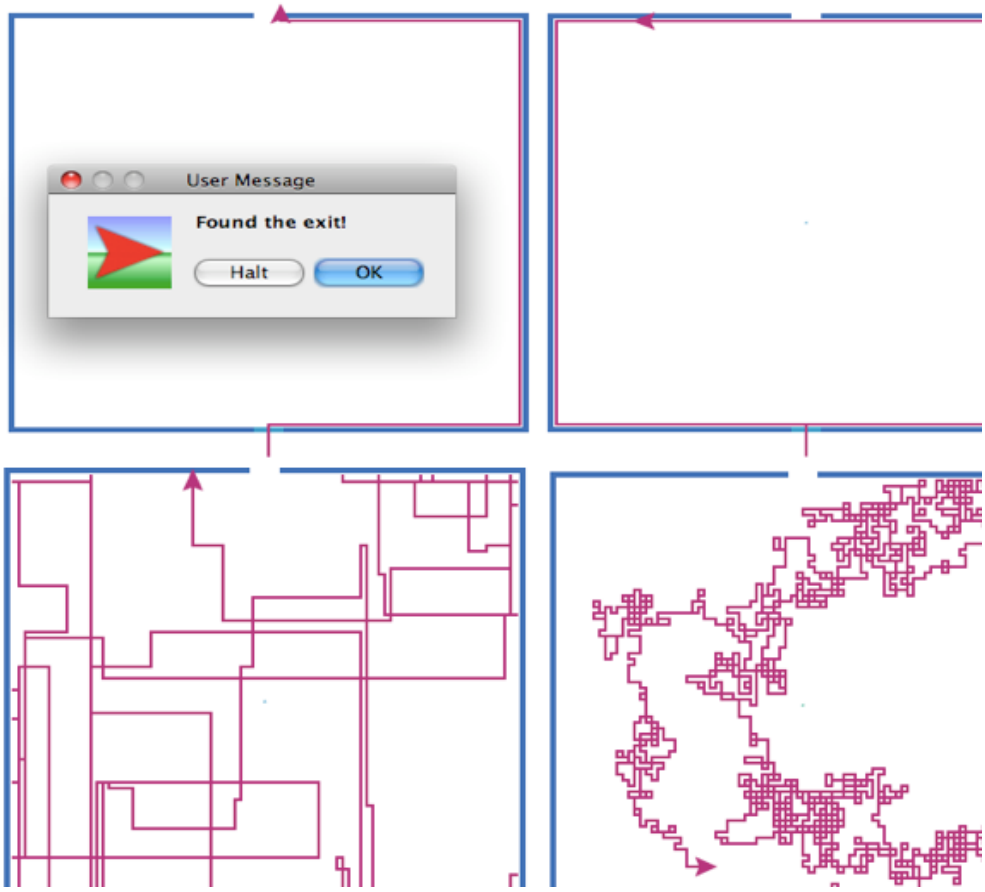


Figure 5.8: How the different turtle behaviours explore the empty maze – top left: Wall Following; top right: Random Forward 0; bottom left: Random Forward 1; bottom right: Random Forward 2.

From an observer's point of view, the fruitlessness of the behaviour is less apparent than with the remaining two behaviours implemented in the model, as it seems as if the agent has some sort of plan, repeating the same steps over and over. (This of course, would be a wrong assumption to make; the agent has no plan at all). Agents executing the two remaining behaviours, however, exhibit a similar pattern to each other, that of 'mindlessly' flittering around the environment in a seemingly aimless random manner, not unlike a small bird or moth (as for the `Random Forward 2` behaviour) would do in real life or a fly repeatedly hitting itself against a closed window (as for `Random Forward 1` behaviour). Both can be frustrating to watch, as both can reach the exit given enough time, but often when they get close to the exit, they can head off in completely the wrong direction as shown in Figure 5.8.

Download free eBooks at bookboon.com

Figure 5.9 shows how the different turtle behaviours explore the Hampton Court Palace Maze. The Hand `On The Wall` Behaviour is well suited to this environment, as the turtle agent executing such behaviour will always reaches the goal which is the centre of the maze, regardless of whether the agent first chose to turn left or right (as shown in the figure). The `Random Forward 0` behaviour, in contrast, will result in the agent only ever managing to explore the first third of the maze, as it cannot do a right or left turn halfway down a corridor. The other two behaviours will eventually result in success for the agents executing them, but this takes substantial time and effort on the part of the agents, and a great deal of luck.

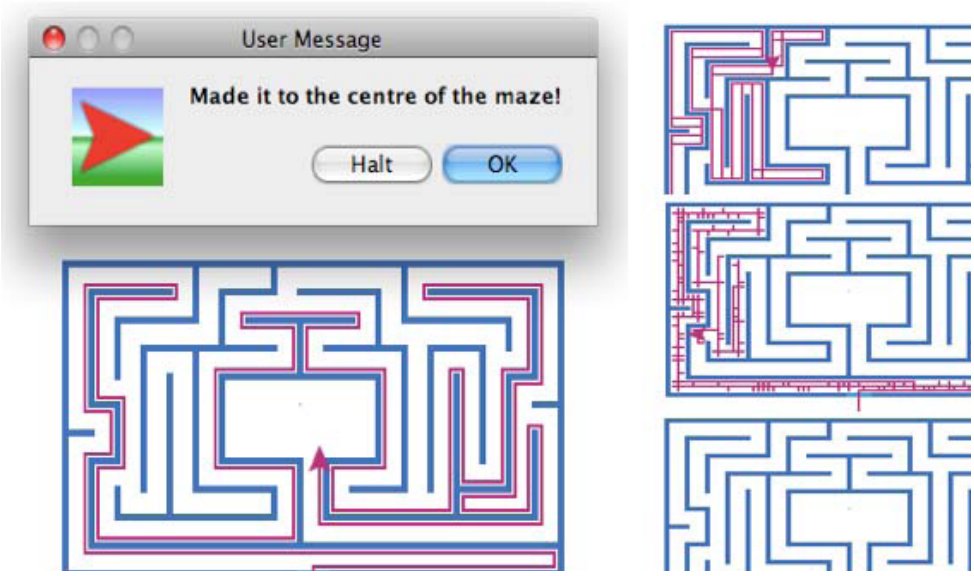


Figure 5.9: How the different turtle behaviours in the Mazes model explore the Hampton Court maze – left: Wall Following; top right: Random Forward 0; middle right: Random Forward 1; bottom right: Random Forward 2.

Figure 5.10 shows how the different turtle behaviours explore the Chevening House maze. `The Hand On The Wall` Behaviour this time is not well suited to this environment, as the maze has been designed deliberately to thwart such behaviour. In contrast, the `Random Forward 0` behaviour can successfully complete the maze in some cases, whereas for the previous maze it was always unsuccessful. This illustrates how the environment plays a crucial role in determining the success (or lack of it) of agents with different behaviours. The behaviour, however, overall is not very effective, as most of the time the agent ends up trapped, endlessly bouncing back and forth immediately above the centre of the maze. The other two behaviours again results in eventual success for the agents executing them, but again, this takes substantial time and effort (and luck) on the part of the agents.

Download free eBooks at bookboon.com

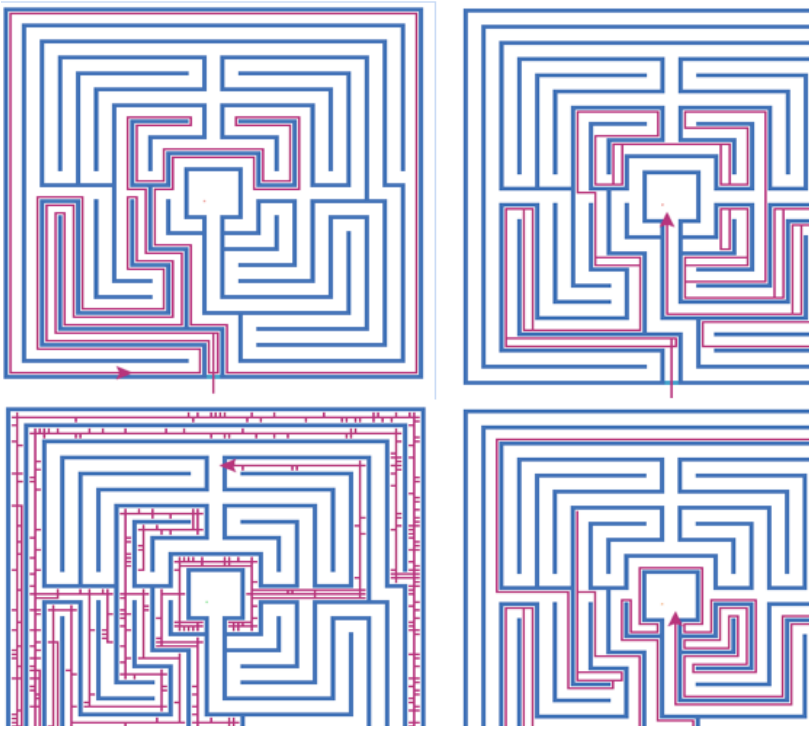


Figure 5.10: How the different turtle behaviours explore the Chevening House maze – top left: Wall Following; top right: Random Forward 0; bottom left: Random Forward 1; bottom right: Random Forward 0 followed by Wall Following.

Click on the ad to read more

Download free eBooks at bookboon.com

An alternative possibility to the execution of a single behaviour is to combine two or more of the behaviours. This is very easy to test out in the Mazes model – simply select a different behaviour using the `turtle-behaviour` chooser in the Interface while the program is executing. For example, for the image on the bottom right of Figure 5.10, the initial behaviour was set to `Random Forward 0`, and then at an opportune time (in order to help the turtle agent reach the centre of the maze), the behaviour was switched to `Hand On The Wall`. This combination of behaviours is usually more effective than any single behaviour by itself, even when the switching is done randomly, as the Hand On The Wall is the most effective at making progress, but it needs temporarily shutting off in order to enable the agent to jump across to the island at the centre of the maze at some point. The lessons to be learned from this are that combined behaviours can be more effective than a single behaviour, and this may become necessary when an environment has changed if the agent wishes to remain effective at a particular task.

5.5 Embodied, Situated Cognition

The way the simple reactive turtle agents in the Mazes model above move around the mazes can be compared to results for agents behaving cognitively – that is, agents that know they need to get to the centre or exit of the maze, and who are able to recognize when they have reached a place in the maze that has multiple paths to be searched. In other words, they have the 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). This act of recognition is a fundamental part of cognition, and also a fundamental part of searching behaviour. 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.

The traditional approach to cognition takes an information processing point of view that perception and motor systems are input and output devices. For example, one particular method of characterising cognition is as a 'sense – think – act' cycle: first the agent perceives something (sense), then it processes what it perceives (think), then it executes an action (act). Recent studies in embodied cognitive science, however, have challenged such a simplistic approach to characterising cognitive behaviour, and places more emphasis on the dynamic interaction of the body with the environment through sensory-motor coordination. The alternative approach, called *embodied, situated cognition*, posits that all aspects of an agent's cognition are shaped by the interaction of an agent's body with the environment it is situated within and with other agents that co-exist in that environment.

In this approach, the body and mind of the agent interact simultaneously with each other and with the environment. There are no internal representations that characterise movements of the agent and its body as separate and distinct events in a 'sense – think – act' method of operation. Instead, sensing, thinking and acting all occur and influence each other simultaneously. As a result, all aspects of cognition, such as categories, concepts, ideas, and thoughts, and all aspects of cognitive behaviour such as decision-making, searching, planning, and learning, are shaped by the interaction between mind, body and environment.

Download free eBooks at bookboon.com

The embodied, situated cognition perspective is method de rigueur for the field of embodied cognitive science. It is adopted, for example, by researchers in Linguistics based on the early work by Lakoff and Johnson, which has highlighted the importance of conceptual metaphor in human language, and which has shown how it is related to our perceptions via our embodiment and physical grounding. It is also widely adopted in robotics, stemming from Rodney Brooks (1986) groundbreaking work in the area, who argues that machines must be physically grounded to the real world via sensory and motor skills through a body. This work breaks down intelligence into layers of behaviours that are based on the agent being physically situated within its environment and reacting with it dynamically. This approach has also become popular in other areas such as behavioural animation and intelligent virtual agents. We will explore this behavioural approach in more detail in the next chapter.

5.6 Summary and Discussion

An agent's body has a crucial role to play in cognition. The way the agent's body interacts with and is situated within its environment in conjunction with its sensing and motor systems determines much of its behaviour. It may seem to be stating the obvious that autonomous agents have a body and are situated in an environment, but from a design perspective, the importance of designing intelligent behaviour based on these attributes cannot be understated; for example, using a first-person design perspective forces the designer to consider aspects from the point of view of the agent, rather than imposing the design from an external point of view.

The traditional approach to characterising cognition – that of a 'sense – think – act' cycle posits that cognition is accomplished in separate sensing, thinking and acting behaviours. Although adequate for designing simple reactive agents, the approach has limitations when considering how to design agents that exhibit intelligent behaviour. An alternative approach, called embodied, situated cognition, emphasizes that sensing, thinking and acting occur simultaneously.

This chapter has presented a number of models in NetLogo to demonstrate various aspects concerning the implementation of embodied, situated agents. Agents require some way of sensing the world and the models have shown how various senses can be simulated, such as: vision (the Look Ahead Example, Line of Sight Example and Vision Cone Example models); and touch (the Wall Following Example model). Agents need not be restricted to the traditional human sense. They can also use senses tailored to recognize changes in the environment such as elevation of the terrain (the Hill Climbing Example model) or chemical laid down in the environment (the Ants model). The environment clearly has an important role to play in determining the agent's behaviour in these examples, but it can also affect what the agent is capable of sensing based on the situation; for example, for the Line of Sight model, the terrain determines what is visible to the agent in a dynamic process that is related to the agent's current location and movement; for the Mazes model, some of the agents become trapped, as their sensing behaviour overlooks a viable path.

Download free eBooks at bookboon.com

However, two important aspects of embodied, situated cognition have not been shown by any of these models. Firstly, for realistic, and believable, behaviour to develop in embodied, situated virtual agents, full physical simulation of the agent's body and the way it interacts with the virtual environment is required. And secondly, for cognition to occur in the common sense of the word, an agent should *knowingly* process sensory information, make decisions, change its preferences and apply any existing knowledge it may have while performing a task or mental thought process; in other words, they *know* what they are doing.

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

- Embodiment for an autonomous agent concerns the body an agent has that it uses to move around with, sense and interact with the environment.
- Situatedness concerns an autonomous agent being located in an environment with other agents and objects, and its behaviour being influenced by what is happening in the environment as a result.
- Reactive behaviour concerns an agent reacting without conscious thought to an event that is occurring. In contrast, cognitive behaviour takes a more deliberative approach – the agent *knowingly* processes sensory information, makes decisions, changes its preferences and applies any existing knowledge it may have while performing a task or mental thought process.
- The 'sense – think – act' cycle refers to a particular modus operandi for an autonomous agent exhibiting intelligent behaviour. This cycle consists of the agent applying three behaviours in turn: the agent first senses what is happening in the environment, then it thinks about what the senses is telling it including deciding on what to do next, then it does what it has decided to do based on some appropriate action.
- The embodied, situated perspective on cognition posits that all aspects of an agent's cognition are shaped by the interaction of an agent's body with the environment it is situated within. Sensing, thinking and acting are done simultaneously and in conjunction with each other, not separately.

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

Model	**URL**
Mazes	http://files.bookboon.com/ai/Mazes.nlogo

Model	**NetLogo Models Library (Wilensky, 1999) and URL**
Ants	Biology > Ants http://ccl.northwestern.edu/netlogo/models/Ants
Hill Climbing Example	Code Examples > Hill Climbing Example; see modified code at: http://files.bookboon.com/ai/Hill-Climbing-Example-2.nlogo
Line of Sight Example	Code Examples > Line of Sight Example; see modified code at: http://files.bookboon.com/ai/Line-of-Sight-Example-2.nlogo
Look Ahead Example	Code Examples > Look Ahead Example; see modified code at: http://files.bookboon.com/ai/Look-Ahead-Example-2.nlogo
Vision Cone Example	Code Examples > Vision Cone Example; see modified code at: http://files.bookboon.com/ai/Vision-Cone-Example-2.nlogo
Wall Following Example	Code Examples > Wall Following Example http://ccl.northwestern.edu/netlogo/models/WallFollowingExample

Download free eBooks at bookboon.com

6 References

Aglets. 2008. URL http://aglets.sourceforge.net/. Date accessed December 26, 2008.

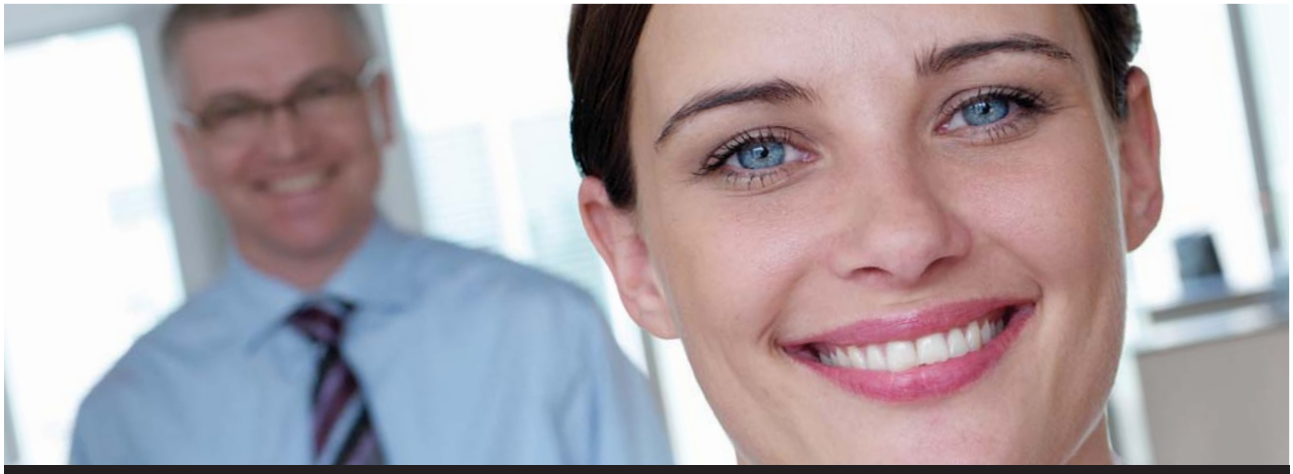
Al-Dmour, Nidal and **Teahan**, William. 2005. "The Blackboard Resource Discovery Mechanism for Distributed Computing over P2P Networks". *The International Conference on Parallel and Distributed Computing and Networks (PDCN)*, Innsbruck, Austria, February 15–17, 2005.

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.



We have ambitions. Also for you.

SimCorp is a global leader in financial software. At SimCorp, you will be part of a large network of competent and skilled colleagues who all aspire to reach common goals with dedication and team spirit. We invest in our employees to ensure that you can meet your ambitions on a personal as well as on a professional level. SimCorp employs the best qualified people within economics, finance and IT, and the majority of our colleagues have a university or business degree within these fields.

Ambitious? Look for opportunities at www.simcorp.com/careers

SimCorp

www.simcorp.com

Click on the ad to read more

Download free eBooks at bookboon.com

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.

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.

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.

Download free eBooks at bookboon.com

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.

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.

Download free eBooks at bookboon.com

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.

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.

Click on the ad to read more

Download free eBooks at bookboon.com

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.

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.

Download free eBooks at bookboon.com

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.

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.

Download free eBooks at bookboon.com

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.

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.

Click on the ad to read more

Download free eBooks at bookboon.com

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.

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.

Download free eBooks at bookboon.com

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.

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.

Download free eBooks at bookboon.com

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.

Masters in Management

Stand out from the crowd

Designed for graduates with less than one year of full-time postgraduate work experience, London Business School's Masters in Management will expand your thinking and provide you with the foundations for a successful career in business.

The programme is developed in consultation with recruiters to provide you with the key skills that top employers demand. Through 11 months of full-time study, you will gain the business knowledge and capabilities to increase your career choices and stand out from the crowd.

Applications are now open for entry in September 2011.

London Business School

London Business School
Regent's Park
London NW1 4SA
United Kingdom
Tel +44 (0)20 7000 7573
Email mim@london.edu
www.london.edu/mim/

For more information visit www.london.edu/mim/
email mim@london.edu or call +44 (0)20 7000 7573

Click on the ad to read more

Download free eBooks at bookboon.com