David W. Russell

The BOXES Methodology Second Edition

Black Box Control of Ill-defined Systems

Second Edition

Springer

The BOXES Methodology Second Edition

David W. Russell

The BOXES Methodology Second Edition

Black Box Control of Ill-defined Systems

Second Edition

Springer

David W. Russell
Audubon, PA, USA

ISBN 978-3-030-86068-4 ISBN 978-3-030-86069-1 (eBook)
https://doi.org/10.1007/978-3-030-86069-1

First edition of this title: https://www.springer.com/gp/book/9781849965279.

This Springer imprint is published by the registered company Springer Nature Switzerland AG
The registered company address is: Gewerbestrasse 11, 6330 Cham, Switzerland

To Donna,
without whose encouragement this work
would never have been completed,
and to my brother George.

Preface

It is interesting how casual browsing through a technical article can capture one's imagination and alter a career. In 1973, while looking for reference materials to supplement my doctoral studies in real-time adaptive, intelligent control, my attention came upon a series of papers describing a new genre of research namely *learning machines.*

The author's own doctoral studies [1] had focused on the design of a real-time pattern generating automaton for the control of an experimental fast reactor, in which control was maintained by filling and emptying tubes of moderator fluid based on what, today, would be called rule-based control laws. Yet, it was also a sly game! The selection of not just *how many* control tubes needed to be activated to respond to a power demand, but it was also necessary that an even *distribution* of tubes be maintained for stability reasons, and that the choice of tube be made with minimum *disturbance.* Of course, in the fast reactor setting, the proximity of any tube to any other enhances the control strength of both. But enough about fast reactors—what was fascinating was that when it was necessary to make changes to the control setting, the pattern needed to change but with minimum alteration of tube activations. It was a sort of Connect Four® situation in which one player's move not only affects his/her board plan but reverses and obliterates those of the opponent.

As in a board game, dynamic systems create control situations that can be calm, erratic or even chaotic, and change without much, if any, warning. To keep up with these demands it became obvious that computational assistance would be necessary. This would need to have the capacity to perform not only the usual control functions but also have the *intellect* to solve logical problems. Around that time, machine learning was emerging, pioneered by Prof. Donald Michie at Edinburgh and others. In this field of study, it was clear that the digital computer, with its speed of calculation could compare many options and return a good control decision in almost real-time.

But was it doing more than running sophisticated trial and error sequences, keeping a tabulation of what was the *best so far*, until it ran out of time when it returned an *intelligent guess*? Artificial intelligence was on the horizon!

With faster hardware came a new slew of algorithms that seemed to be able to present fast solutions to puzzles, games and conundrums that elude all but the

smallest percentage of humankind. For example, as far back as 1959, Samuel [2] had developed a program that purported to have learned to play checkers and improved with each game. What was especially intriguing was the fact that the automaton (which was the computer program and an avatar to assert board entries and read opponents moves) improved in the performance of certain tasks (i.e. not losing or winning) over time. Around that era many software developers (too numerous to mention here) began to work on the resolution of various game situations in the much more complex game of chess which culminated in the now historical match [3] in 1997 between Deep Blue® and the then reigning World Chess Champion, Garry Kasparov. In 2010, the state of the art in chess playing software systems had reached such a level of proficiency that Veselin Toparov trained for the World Chess Championship using the Blue Gene [4] Supercomputer, although eventually losing to Viswanathan Anand in games that were all close.

What has fascinated scientists, especially engineers, is not only the answer to *is the machine truly intelligent* but also *can an automaton perform real-work tasks better than a human or analog system*?

Concurrent with this activity by what was to become the *Artificial Intelligence* community, was a surge in interest in the replacement of analog process systems by what became known as adaptive digital controllers. It was only a matter of time before the two fields overlapped and the era of learning systems began. For example, among many others, a 1968 paper by Lambert and Levine [5] appeared titled "Learning Control Heuristics." By replacing analog control systems by digital computers it became obvious that the digital system could do much more than produce pretty graphs and digital meter readings. The era of analog controllers was coming to an end and the new world of control algorithms such as statistical trending, optimization, state space, and fuzzy logic had arrived.

All the system needed was the ability of a human to program its rule sets, be it the exclusion rule for noughts and crosses (a.k.a. tic-tac-toe in the US) in which it is not legal (or nice!) to overwrite a square already held by the opponent, or the control limits on the temperature of a chemical process. A fatal flaw had emerged: Was all the digital processor doing replacing functions that an analog computer or human could do with some rather expensive electronic apprentice. Most systems required a mathematical model of the process under control that was derived from physical laws or known rules. The programmer's task was to code the algorithms and attach them to a waiting program. Did this introduce intelligence or just mimic the workings of analog systems? Was direct digital control (DDC) the answer to solving complex control problems that were rapidly exhausting the capacity of analog controllers?

The assertion that machines (vis-à-vis) robots possess intelligence was and is fiercely debated by engineers and philosophical purists alike. The now infamous *Lighthill Controversy* debate that was aired live by BBC TV in 1973 was in essence an 81 minute argument between Donald Michie and leading researchers that is still blamed for the AI Winter, so called because of the bleak times AI researchers went through in terms of getting grants and support. Over thirty years later, a 2007 posting on the Steeb-Greebling Diary [5] sadly reported he ongoing disagreement between Michie and his colleagues. As an outcome of that same debate, it recounts that Sir

James Lighthill subsequently published a report that called halt to artificial intelligence research in all but two area. The resulting dissolution of Donald's research group in Edinburg left him isolated in the research unit.

Michie left Edinburgh in 1984 to become Professor of Computer Science at Strathclyde University, where he established the Turing Institute. He worked there for 10 years contributing much to the field of machine learning. In March 1990, the author visited Michie at the Turing Institute and gave a presentation based on the Liverpool work aptly titled *The Trolley and Pole Revisited*.

Machine intelligence is still a hot topic of conversation. In 2008 the *Guardian* [7] ran a headline: "Artificial intelligence: God help us if machines ever think like people" in which Charles Arthur cites a paper by Gary Marcus [8] in which he states that there are two systems operating in our minds, one *ancestra*l and the other *deliberative*. The deliberative system is younger genetically speaking but the older one, being better wired into our subconscious, often gets the upper hand. What this suggests is that human minds are not really able to create much that is new regardless of the challenge, so why should an artificial system that may be relentlessly pursuing some algorithmic agenda produce innovative, yet viable and practical solutions to a problem? In the 21st century, there is hardly a day goes by that artificial intelligence is not proudly the topic of admiration in the public eye.

Returning to history around same this same time, a seminal work by Michie and Chambers [9] written in 1968 attracted the author's special attention. The latent thesis of the work was that *Machines can learn* and researchers have been pursuing this and the obvious extension *Do machines think* to this day. The outcome of the revelation was the foundation of the research team at the then Liverpool Polytechnic in the UK and where this journey began and the genesis of this book [10]. Almost 50 years later, the BOXES algorithm still fascinates researchers, especially this author and this second edition highlights some features and issues that scholars might have pondered.

Audubon, PA, USA David W. Russell

References

1. Russell, D.W. Advanced Analysis of Fluid Control Systems for Nuclear Reactors (1970) PhD thesis. Council for National Academic Awards, London.
2. Samuel A.L. Some studies in Machine Learning using the game of checkers 1959. *IBM J. of Res. Dev* 3, 210–229.
3. Deep Blue: Game 6: May 11 (1997) www.research.ibm.com/deepblue/watch/html/c.shtml (Accessed December 21, 2010)
4. Blue Gene. *IBM J. of Res. Dev.* 2005: 49, No: 2/3 191-489.
5. Lambert, J.D. and Levine, M.D. Learning Control Heuristics. 1968. *IEEE Trans on Automatic Control*. Vol:AC-13, 741–742.
6. The Lighthill controversy The Streeb-Greebling Diaries (2007) http://streebgreebling.blogspot.com/2007/07/lighthill-controversy.html

7. Arthur, C. Artificial intelligence: God help us if machines ever think like people 2008 http://www.guardian.co.uk/technology/2008/jun/20/artificial.intelligence
8. Marcus, G. *Kluge: The Haphazard Evolution of the Human Mind*. 2008. Houghton, Mifflin Harcourt Publishing Company, New York: NY ISBN 978-0-618-87964-9.
9. Michie, D. *On Machine Intelligence*. 1986. Ellis Horwood Ltd. Chichester, England. ISBN: 0-7458-0084-X.
10. Russell, D.W., The BOXES Methodology: Black Box Dynamic Control. 2012. Springer Verlag, London, England. ISBN 978-1-84996-528-6.

Acknowledgements

Special acknowledgement is due to Steve Rees and John Boyes, my long-suffering Liverpool team members, through whose endeavour the BOXES automaton was designed, constructed, and successfully operated. I recall the countless hours over four years that Steve tried to outplay the automaton manually and his glee at obtaining a controlled run of 10 seconds one day, when of course the automaton attained 10 seconds after an hour or so of training.

Acknowledgement is also made to James Alpigini whose contributions to the visualization of dynamic systems were also very valuable. Finally, I am grateful to Dennis Wadsworth of the Lockheed Martin Corporation for his enthusiasm and skill in adapting the BOXES algorithm into a form that positively enhances database disc access and his contribution to Chap. 12.

The writing of this second edition was made possible by the generous use of facilities and resources of the Engineering Division of Penn State Great Valley in which I am a proud emeritus professor.

I acknowledge the work of many fellow researchers who have used, altered, and otherwise adapted the BOXES method over the years and whose work may have been inadvertently omitted.

Lastly, I am indebted to my Springer editors and friends Anthony Doyle and Mrs. Padma Subbaiyan without whose help this second edition would never have come to print. All and any mistakes are mine alone and readers should please feel free to contact me with any suggestions you may have for improvements or corrections.

Donald Michie: A Personal Appreciation

I met Donald Michie in Edinburgh in the early 70's after reading some of his papers and became fascinated by a learning methodology that he called "The BOXES "Method." I continued to enjoy a fairly loose but valued relationship with him over the many years since; meeting at conferences, lunch in London on occasion, giving a lecture at the Turing Institute in Glasgow and so on. I have published over 30 technical papers—see Appendix C—on the method. Donald was a controversial and outspoken figure in UK academia over the years and always ready to branch out into new adventures. Among many others, I was much saddened by his much too soon departure from this life in July 2007 following a car crash.

It was truly gratifying to see the appearance of a book [1] titled *Donald Michie on Machine Intelligence, Biology and more* that contains a wonderful collection of Professor Michie's papers, and I quote from page 6:

> *With the 1990's also came, finally, recognition of his contributions to machine intelligence in the form of several awards. In 1995, he co-founded the Human Computer Learning Foundation, with the aim of exploring ways in which computers could be used to assist and improve human skills.*

Mechatronic machines do very well what we humans are extremely limited in doing; any task that involves flawless and reliable memory, undivided attention, and strict devotion to the task at hand. Donald opined (op cit 239) that "*The black death of our times is the world's escalating complexity*" and perhaps it is the profound simplicity of the BOXES method and Donald's enthusiasm for everything, that was my motivation for writing this monograph and now the second edition.

1. Ashwin Srinivasan (ed) (2009) Donald Michie on Machine Intelligence, Biology and more. Oxford University Press, London. Used with permission.

Contents

1 Introduction 1
1.1 Machine Intelligence 1
1.1.1 Are Computers Intelligent? 2
1.1.2 Can Computers Learn? 4
1.1.3 The Robot and the Box 5
1.1.4 Does the BOXES Algorithm Learn? 8
1.1.5 Can Computers Think? 8
1.1.6 Does the BOXES Algorithm Think? 9
1.2 The Purpose of This Book 10
1.3 A Road Map to Reading This Book 10
1.4 Concluding Thoughts 11
References 11

Part I Learning and Artificial Intelligence (AI)

2 The Game Metaphor 15
2.1 Computers Can Be Programmed to Play Games 15
2.1.1 Playing by Rules 16
2.2 Reactionary Strategy Games 18
2.2.1 Noughts and Crosses 18
2.2.2 OXO Not Noughts and Crosses 21
2.3 Incentives and Learning Velocity 22
2.4 Design of a Noughts and Crosses Engine 23
2.4.1 Overview 23
2.4.2 Software Substructures 24
2.4.3 Typical Results 25
2.5 Chance and Trial and Error 25
2.5.1 Chance and the Community Chest 26
2.5.2 Learning with Guesswork 26
2.5.3 Random Initialization 26
2.5.4 Positional Move Strengths 27

2.6 The Payoff Matrix 28
2.7 The Signature Table 29
2.8 Rewards and Penalties 30
2.9 Failure-Driven Learning 31
2.10 Concluding Thoughts 31
2.10.1 Reversi (Othello®) 32
2.10.2 The BOXES Method as a Game 32
References 33

3 Introduction to BOXES 35
3.1 Matchboxes 35
3.2 Components of the BOXES Method 36
3.2.1 Defining the Game Board 36
3.2.2 Identifying Game Situations 37
3.2.3 Selecting Game Piece Actions 39
3.2.4 Real-Time Data Handling 40
3.2.5 Detecting an End Game Situation 41
3.3 Updating the Signature Table 42
3.3.1 Overall Performance Data 42
3.3.2 Desired Level of Achievement 43
3.3.3 Individual Box Decision Data 43
3.4 Overall Software Design 44
3.5 Concluding Comments 46
References 46

4 Dynamic Control as a Game 47
4.1 Control of Dynamic Systems 47
4.1.1 The Dynamic System Game Board 48
4.1.2 State Variables and State Integers 49
4.1.3 Creating a Unique System Integer 50
4.1.4 Signature Table Control 52
4.1.5 End of Game Action 52
4.2 Actual Real-Time Data Collection 53
4.2.1 Short Duration Mechanically Unstable Systems 53
4.2.2 Continuous Systems with Sample Data 55
4.3 Update Procedures 56
4.4 Concluding Comments 57
References 58

Part II The Trolley and Pole

5 Control of a Simulated Inverted Pendulum Using the BOXES Method 61
5.1 Introduction 61
5.2 The Trolley and Pole Model 62
5.2.1 The Trolley and Pole Signature Table 63

5.2.2 Systems Engineering . . . 64
5.2.3 An Overall Performance Metric . . . 65
5.2.4 The Importance of State Boundaries . . . 66
5.2.5 Computation of a Unique System Integer . . . 66
5.3 Simulation Software . . . 67
5.3.1 The Campaign Setup Phase . . . 67
5.3.2 The Individual Run Phase . . . 68
5.4 Simulation Results . . . 69
5.4.1 Typical Results . . . 70
5.5 Update of Statistical Databases . . . 70
5.5.1 Determination of Decision Strength . . . 71
5.5.2 Near-Neighbor Advisor Cells . . . 73
5.6 Conclusions . . . 73
References . . . 74

6 The Liverpool Experiment . . . 75
6.1 Introduction to Reality . . . 75
6.2 The Liverpool Trolley and Pole Rig . . . 76
6.2.1 Practical Aspects of the Liverpool System . . . 76
6.2.2 Instrumentation and the State Variables . . . 79
6.2.3 Manual Auto-start . . . 80
6.2.4 Driving the Trolley . . . 80
6.2.5 The Microprocessor . . . 81
6.2.6 The BOXES Algorithm and the Real-Time Monitor . . . 81
6.3 Systems Engineering . . . 81
6.3.1 How Boundaries on Each State Variable Were Imposed . . . 82
6.4 Results from the Liverpool Rig . . . 83
6.5 Conclusions . . . 84
References . . . 84

7 Solving the Auto-Start Dilemma . . . 85
7.1 Introduction to the Auto-Start Dilemma . . . 85
7.2 Random Restart Simulation Software . . . 86
7.2.1 Restart Software . . . 87
7.2.2 Random Initial State Integers . . . 87
7.3 Automated Catch-Up Restart Method . . . 90
7.3.1 Catch-Up Restart in Simulated Systems . . . 92
7.3.2 Catch-Up Restart in a Real Trolley and Pole Rig . . . 92
7.3.3 Catch-Up Method Conclusion . . . 94
7.4 Systems Engineering . . . 95
7.5 Manual Experiments . . . 95
7.6 Conclusions . . . 96
References . . . 97

Part III Other BOXES Applications

8 Continuous System Control 101
8.1 Continuous Control 101
8.2 A Different Perspective on Failure 103
8.2.1 Constructive Failure 103
8.2.2 Limited Failure 104
8.2.3 Creating a Learning Interlude 105
8.3 Outcome-Based Performance Evaluation 107
8.3.1 An Example Outcome-Based Approach 107
8.3.2 Outcome-Based Assessment 108
8.4 Training Continuous Automata 108
8.5 BOXES Control of a Continuous System 109
8.5.1 PID Control 109
8.5.2 State Variable Control 110
8.5.3 BOXES for Continuous Systems 110
8.6 A BOXES Augmented Controller Results 112
8.6.1 Outcome-Based Reward and Penalty 112
8.6.2 Results Obtained for the Example 114
8.7 Conclusions 114
References 115

9 Other On/Off Control Case Studies 117
9.1 On/Off Control 117
9.1.1 Learning Algorithms 118
9.1.2 Run Time Data 118
9.1.3 Signature Table Update 119
9.1.4 Application Summary 119
9.2 Fedbatch Fermentation 119
9.2.1 The Fedbatch Fermentation Process 120
9.2.2 A Fedbatch Fermentation Model 122
9.2.3 BOXES Control of a Fedbatch Fermentor 125
9.3 A Municipal Incinerator with BOXES Control 129
9.3.1 A Model of a Municipal Incinerator 129
9.3.2 BOXES Control of the Municipal Incinerator 130
9.4 Reversing a Tractor Trailer 131
9.4.1 Model of the Tractor Trailer 131
9.4.2 BOXES Control of Tractor Trailer Reversal 132
9.5 Conclusions 134
References 134

10 Two Nonlinear Applications 135
10.1 Introduction 135
10.2 Database Access Forecasting 136
10.2.1 Application of BOXES to Disk Accessing 136
10.2.2 Learning in the Adapted BOXES Algorithm 138

10.2.3 Forecasting ... 139
10.2.4 Simulation Results ... 139
10.2.5 Conclusions Disk Accessing ... 141
10.3 Stabilizing Lorentz Chaos ... 141
10.3.1 Introduction ... 141
10.3.2 BOXES and the Fractal Dimension ... 143
10.3.3 Results for Lorenz Chaos Under BOXES Control ... 146
10.3.4 Conclusions Lorentz Equations ... 147
10.4 Conclusions ... 147
References ... 147

Part IV Extending the Algorithm

11 Accelerated Learning ... 151
11.1 Introduction ... 151
11.2 Preset Fixed Signature Table Values ... 152
11.3 Reduction of the Importance of Short Runs ... 154
11.4 Collaboration Among Cells ... 156
11.4.1 Playing Bridge ... 157
11.4.2 Swarm and Ant Colony Systems ... 157
11.4.3 Advisors in the BOXES Algorithm ... 158
11.4.4 Locating Peer Advisor States ... 158
11.5 Relocation of Boundaries ... 160
11.6 Conclusions ... 161
References ... 162

12 Two Advising Paradigms ... 165
12.1 Introduction ... 165
12.1.1 Decision Strengths of Cells ... 167
12.1.2 Identification and Ranking of Advisor Cells ... 169
12.1.3 Advisor Strength Criteria ... 171
12.2 Advising Schemas ... 172
12.2.1 Advising by Voting ... 172
12.2.2 Advising Using Cell Strengths ... 173
12.2.3 Delayed Advising ... 174
12.3 Advisor Accountability ... 176
12.4 Conclusions ... 178
References ... 178

13 Evolutionary Studies Research ... 179
13.1 Introduction ... 179
13.2 State Boundary Experiments ... 180
13.2.1 Variation in the Number and Size of State Zones ... 180
13.2.2 Preliminary Conclusions ... 181
13.3 An Evolutionary Paradigm ... 181

13.3.1 Types of Zones . . . 182
13.3.2 An Evolutionary Method . . . 182
13.3.3 Interpreting Signature Table Statistics . . . 184
13.3.4 An Evolutionary Algorithm . . . 188
13.3.5 Example Results . . . 188
13.4 Conclusions . . . 189
References . . . 190

Part V Further Thoughts

14 A Priori Knowledge . . . 193
14.1 What are Black Box Systems? . . . 193
14.1.1 An Intelligent Pacemaker . . . 194
14.1.2 Controlling a Steel Rolling Mill . . . 195
14.1.3 Why Use the Cart-and-Pole Exemplar? . . . 196
14.2 What does the BOXES Paradigm Need to Know? . . . 196
14.2.1 The Physical System to be Controlled . . . 196
14.2.2 The Division of the State Variables into Zones . . . 197
14.3 Other BOXES Configuration Data Items . . . 197
14.3.1 The Signature Table . . . 198
14.3.2 Evaluating System Merit . . . 198
14.3.3 In-Run Data Collection . . . 198
14.3.4 The Cell Usage Database . . . 199
14.3.5 Learning Parameters . . . 199
14.4 Fixed Cell Strategies . . . 201
14.4.1 Create Non-changeable States . . . 201
14.4.2 Strong Cell Freezing . . . 202
14.4.3 Can Fixed Cells Participate in Evolutionary Studies? . . . 202
14.5 Conclusions . . . 202
References . . . 203

15 Detecting and Handling Jitter . . . 205
15.1 Introduction . . . 205
15.2 Why Does Jittering Occur? . . . 206
15.3 How Jitter Affects Merit . . . 207
15.3.1 A Numerical Illustration . . . 209
15.4 How Jittering Corrupts Individual Cell Data . . . 209
15.5 Detection of Jittering in a BOXES System . . . 211
15.6 Possible Strategies for Jitter Remediation . . . 213
15.6.1 Executive Internal Action . . . 214
15.6.2 Jitter Proof Software . . . 215
15.7 Conclusions . . . 216
References . . . 217

16 Handling Untrained Data 219
16.1 Glossary of Computational Terms 219
16.2 A Mathematical BOXES Test Engine 221
16.2.1 Scenario 1: Linear Increase or Decrease 222
16.2.2 Scenario 2: Linear Saw Tooth 222
16.2.3 Scenario 3: Complex or Random Pattern 223
16.3 Forgetfulness Observations Using the BOXES Test Engine 223
16.3.1 Aging Global Use (GU) 223
16.3.2 Aging Global Life (GL) 224
16.3.3 System Merit just Aging the Global Life (GL) 225
16.3.4 System Merit for a More Complex Scenario 225
16.3.5 Effect of Differing Values of Δk 226
16.3.6 Summary of δk as a Learning Agent 227
16.4 An Alternate Aging Strategy Using a FIFO Stack Architecture 227
16.4.1 The FIFO Data Structure 227
16.4.2 Application of the FIFO Stack to BOXES Merit 228
16.4.3 Connection of the Test Engine to the FIFO Stack 228
16.4.4 Merit Values Using the FIFO Stack 229
16.5 Conclusions 231
References 231

Part VI Conclusions

17 Summary and Conclusions 235
17.1 Some Philosophical Commentary 235
17.2 Bloom's Taxonomy 237
17.3 Introduction and Part I: Learning and Artificial Intelligence . . . 238
17.3.1 Chapter 1: Introduction 238
17.3.2 Chapter 2: The Game Metaphor 238
17.3.3 Chapter 3: Introduction to BOXES 239
17.3.4 Chapter 4: Dynamic Control as a Game 239
17.4 Part II: The Trolley and Pole 239
17.4.1 Chapter 5: Control of an Inverted Pendulum Using BOXES 240
17.4.2 Chapter 6: The Liverpool Experiment 240
17.4.3 Chapter 7: Solving the Auto-start Dilemma 240
17.5 Part III: Other BOXES Applications 241
17.5.1 Chapter 8: Continuous System Control 241
17.5.2 Chapter 9: Other On/Off Control Case Studies 242
17.5.3 Chapter 10: Two Nonlinear Applications 242
17.6 Part IV: Improving the Algorithm 242
17.6.1 Chapter 11: Accelerated Learning 243
17.6.2 Chapter 12: Two Advising Paradigms 243
17.6.3 Chapter 13: Evolutionary Studies Research 244

17.7 Part V: Further Thoughts 244
17.7.1 Chapter 14: A Priori Knowledge 245
17.7.2 Chapter 15: Detecting and Handling Jitter 245
17.7.3 Chapter 16: Handling Untrained Data 245
17.8 Modifications to Standard BOXES Software 246
17.9 Research Questions for Future Study 246
17.9.1 Is There an Optimal Number of States? 246
17.9.2 Is There a Generic Merit Formula? 248
17.9.3 Is There a Generic Cell Strength Formula? 248
17.10 Conclusions 248
17.10.1 Some More Final Thoughts 249
References 249

Appendix A: Glossary of Terms and Abbreviations 251

Appendix B: BOXES Software Notes 253

Appendix C: BOXES Publications, Lectures, and Presentations by the Author 269

Author Biography 273

Index 275

Chapter 1
Introduction

Abstract This chapter contains a brief introduction to machine intelligence and how scientists and engineers often approach similar problems from two very distinct standpoints. That a computational engine can play chess better than almost everyone on the planet goes without saying, but is it really *thinking*? While it is obvious that digital computers are unimaginably fast, data retentive and focused, are they merely doing what they have been told to do as Ada Lovelace suggested, or are they capable of learning and making exploration inside their solution space? Automata have taken the software application process beyond soft tasks such as game playing and moved them into the real world of process control, material handling, digital assistants, and autonomous motion. This is exactly what this book covers using the BOXES methodology as the software agent to be discussed at some length throughout. This introductory chapter is written in a deliberately non-technical style and serves to introduce the basis of the topic at hand. It ends with a road map of the how the book is organized to assist the casual reader to browse and skip to sections of special interest.

1.1 Machine Intelligence

There are simply scores of books written about machine intelligence, too numerous to even try to reference here. Interested readers should try a Web search when for instance on December 3, 2010, Google™ identified 8.7 million hits with *machine intelligence* in the subject line. In 2021, the number of hits was 562 million. Authors who write on this subject are engineers, (computer) scientists, mathematicians, philosophers, and pundits seemingly from every walk of life. Some devote their contributions to clever search methods that navigate through esoteric multi-tuple tree structures, others seek to process strings of natural language using context-dependent lexicology to reduce sentence complexity and reduce ambiguity, while others construct expert or fuzzy logic systems which are in use everywhere. All add valuable contribution to the field, and each year intelligent systems applications become more sophisticated and less visible to the user.

D. W. Russell, *The BOXES Methodology Second Edition*,
https://doi.org/10.1007/978-3-030-86069-1_1

Luger and Stubblefield [1] in the preface of their book offer two delightful discriminatory phyla concerning the major schools of AI methodology which they called the *neats* and the *scruffies*. The *neats* focus on theory and knowledge representation, whereas the *scruffies* are interested more in the application of intelligent systems. Their book correctly postulates that the two approaches have much common ground and they further propose that there is much synergy when both are combined. In any research or application that involves computational intelligence as it is called, there is controversy. The controversy that has swirled around machine intelligence has been largely fuelled by the confusion between machine function, cognition, being, and self-awareness. This book does not address those issues beyond this chapter, but it is very fascinating to consider how the interface between humans, automata, and humanoids is becoming more blurry every day.

Video games are now so immersive and addictive that their level of reality has to be carefully regulated to safeguard the sanity of the players. It is sad but true that some people would rather live in this second world than in reality. The more sophisticated virtual reality (VR) environments [2] provide a wonderful arena for training or retraining personnel in a new technology or operational technique. Using a haptic interface allows the trainee to not only gain new knowledge but also experience how the real system will feel. Of course, the well-known DaVinci® robot can observe a complex surgical maneuver once and reproduce it repeatedly on patients who may be thousands of miles away.

VR can be so persuasive that it can even provide an advanced form of artificial life, for example, for persons with crippling disabilities. Using stimulation of the person's senses, participants can become so engrossed and immersed in the system that they are able to believe that they are actually flying a plane, scoring a goal, or attending a class and learning a new language. Avatars can now reflect the facial expressions of the user to convey a sense of mood as it takes the user's part in a discussion or play.

The insertion of technology into main street life is here to stay. It is estimated [3] that in 2010 there were over 100 million Twitter ™ users, a number that is increasing at a reported 300,000 new subscribers every day. Humanoid personal digital assistants (PDA) exist (for the very rich), and the nouveau *Jeeves* [4] (who was Wodehouse's *gentleman's personal gentleman*) is an Internet-enabled system that adaptively manages calendars, suggests dress for the day's weather, merges schedules, and handles routine conversations with other PDAs. Technology is becoming more invisible, ubiquitous, and necessary part of society everywhere.

1.1.1 Are Computers Intelligent?

What is intelligence? Is a person who plays *chess* (or Shatranj as it was originally known in the first Millennium) at the master level *more* intelligent than somebody who does not play at all? Does intelligence depend on a person's ability to reason and strategize within some rule set? Computers play at high levels and regularly defeat

lesser players. Is intelligence simply defined by what a person can *do*, or are there deeper matters that must be considered? It can be argued that the game of chess can be learned by anybody after enough losses and repetitions. A grandmaster must not only play many games but also possess the drive to improve and a semi-photographic memory to learn from not only the game in progress, but also reference past games that s/he played and games that were played by others over the years.

Once all of these factors are in place, a player can imagine a next move. Before instantiating what seems to be the best move available in the current board situation, the player must now view the future—searching as far down an enormous solution tree as able—to see the consequence of the move in regard to winning the game while thwarting the present and future wiles of an opponent. And all this must be done quickly. It is reported that there are 6,134 combinations of position which the pieces on the chessboard can occupy and the number of legal moves is estimated to be somewhere between 10^{43} and 10^{50}. The overall game-tree complexity of chess was first calculated by Claude Shannon as 10^{120}, a number known as the Shannon number. Typically, an average position dictates one of thirty to forty possible moves, but there may be as few as zero (in the case of checkmate or stalemate) or as many as 218 [5]. With this in mind, it is not inconceivable that a program can search through millions of legal moves per second and consequently outplay all but the most expert chess aficionados. Computationally, a major problem is to decide which of maybe several moves is best within the game clock's timer. So, in addition to finding strategically advantageous moves, the program must have a way of ranking each solution.

Automata do not necessarily learn from past mistakes, rather the computer by virtue of its blistering processing speed and power can take advantage of the fact that chess is a deterministic game and can identify sequences of moves by tree searching and flawless memorization. If the game is encoded correctly, it is fairly simple to reference prior games played by humans as is commonplace in many expert systems. If the current game situation matches one of these games, a set of moves can be generated to defend or attack a position. So, is the computer as *intelligent* as its human opponent? In fact, the chess program performs much better than most humans do in playing this game. Is this because humans quickly lose interest and lack concentration when multiple options are available, or is it that they are susceptible to devious gambits, or simply that they cannot look far enough ahead down a yet unknown and inscrutable solution tree?

It would appear that the chess playing program is really a *scruffy* application, yet there have been countless *neat* approaches to pruning the search tree to gain temporal advantage when a finite time is strictly allocated to each move. The so-called rapid-fire chess game may prove more difficult for the automaton to play than the more formal competition game. The computer is unaware that it is playing chess, and its quest for the perfect next move is only limited by the time constraints imposed by the rules of the competition. The program arrives at and memorizes a list of possible moves and ranks them according to some measure of their strength.

This is not a simple process as the program might be fending off a latent opponent's attack or making an offensive thrust or crafting a draw because of the hopelessness of its situation. This is very much akin to how humans weigh options before making

a final decision. When all the known options have been exhausted, or as the time clock nears expiration, the best move (so far) is submitted to continue game play. This is a fundamental component of the BOXES methodology to be discussed in this monograph.

Natural language processing (NLP) is another AI topic that has been extensively researched. As electronic devices have evolved, not only can a system recognize what has been said, the truth of the statement, and who has said it. Training of such systems is intuitive and quite simple provided a lexicon of words is provided. Modern automobile global positioning systems (GPS) all boast speech recognition software. Yet, language is much more than *words*, so it takes more than computational correctness to understand all of the subtleties and nuances, for example, in a spoken *sentence*. But, again does a speech synthesizer know what it is *saying* or is it merely responding to prompts that produce a choice of sounds and tones from a syntactic list? A hearing viewer only has to read the close captioning text during a TV show to see the rather hilarious defects in these systems. However, when programmed and used correctly, spoken words or even eyelid blinks can result in hitherto unimaginable levels of communication and control, which has created life changing improvements, for example, in the quadriplegic community. It is often the interface and trust between the computer and the user that is of paramount importance.

1.1.2 Can Computers Learn?

Can a machine learn to do a task? The introduction of computer science to our university curricula, followed by the more systematic discipline of software engineering, has enabled researchers and scholars to greatly expand upon what had previously been a set of theoretical conjectures prior to the 1950s. The visual and plug-and-play programming paradigms have made applications much simpler to create than possible in the procedural programming process. The writing of small programs and constructing Web sites is taught in many high schools around the world. Computer programming as a commercial skill begat the information technology (IT) industry. From its number crunching origins, to the replacement of handwritten point-of-sale and banking processes to today's handheld automated systems, we all take for granted, the computer has become an integral part of everyday life. And yet society demands more and more from what are really quite simple processes. Cartographically, it is much easier to read and write typescript than handwritten text and is generally less error prone and consequently much more efficient in document editing. An email or text message contains far more traceable information content than a partially heard telephone call or a hastily scribbled note. Modern software can automatically and accurately transcribe language into text and forward the message on command as an email or text message.

There are many technical aspects of computing. Attaching machinery to process controllers and inserting intelligent electronics that provide engine control and navigation was a natural next step. Going further, with the assistance of Hollywood's

fascination with robots and humanoids, the notion of autonomous *thinking* machines appeared. But can an automaton learn for (or about) itself? Expert programmers can write software that provides a facsimile of real-world interaction, intelligent decision-making, and demonstrable reasonable action. Some say that this is how the human brain operates and it is only scalability that separates it from an automaton. Merckle [6] opined that the thinking that the brain is a type of computer has gained general acceptance. It is commonly thought that the brain handles 10^{16} synaptic operations per second which was far in excess of any man-made machine at the time. In 2020, the Fugaku supercomputer [7] boasted over 400 times the speed of the brain; however, because the brain does more than process mathematical data, it might seem feasible to assume the position that bigger and faster computers could possibly reproduce other brain tasks such as learning and potentially thinking and feeling. That a computer could solve problems better and faster than the average human became very clear. One of the pioneer systems that illustrated this was developed at Stanford University.

1.1.3 The Robot and the Box

Early work at the Stanford Research Institute (SRI) reported mobile automata such as Shakey [8] in the 1960s, and its successor Flakey [9] in the 1990s had demonstrated that a mobile computer system could be constructed that could process real-world data, such as location, the recognition of objects using camera vision, and their orientation using gyroscopes. With this information, it could create logically correct plans of action in response to a series of task commands. A rudimentary natural language processor enabled tasks to be input verbally, and the system could solve a variety of problems and produce appropriate actions such as steering and pushing. As the task was being solved, the robot could navigate around obstacles using self-formulated plans of action. What was more intriguing was the ability of the software to handle options and interruptions and to build modifiable plans of action in real time.

Figures 1.1 and 1.2 depict scenes that include objects, rooms and doors that can be opened and closed by outside *gremlins* as SRI called them. A typical task directive might be

>> **move** block A **from** room 1 **to** room 2>>

Figure 1.1 shows the obvious direct route plan, and Fig. 1.2 is the modified plan if doors (D2) and (D3) were found to be closed. If the robot had started to push the block toward door D2 when it was slammed shut, the system would abort the current plan and propose another route on-the-fly. In order for the algorithm to be able to formulate plans, it must be cognizant of the building layout (i.e., what rooms are available and where the doors are), where it is, and the status of each door (open/closed) on a moment by moment basis. It is a given that the robot had been given sufficient power to overcome inertia and push the box. The three feasible but conditional plans in this simple system are as follows:

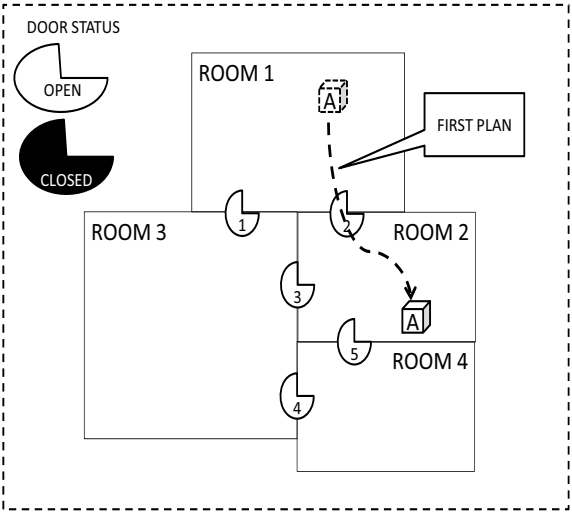

Fig. 1.1 Initial plan for the set task

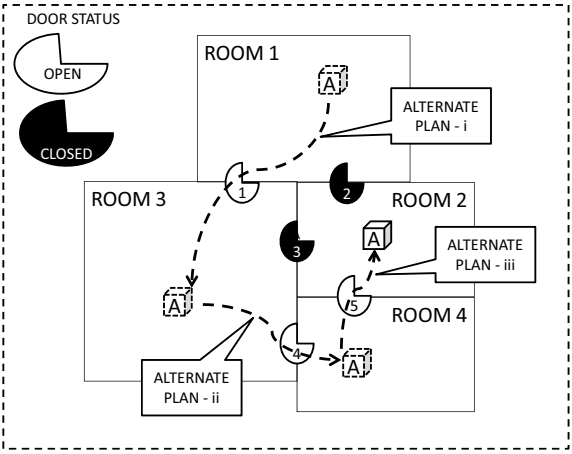

Fig. 1.2 Modified task plan if doors D2 and D3 are closed

- Plan A: D2: The most direct route
- Plan B: D1 → D3 {D2 is closed}, and
- Plan C: D1 → D4 → D5 {D2 and D3 are closed}.

Other plans can be envisaged for various combinations of door status values. Of course, some combinations, for example, D1 and D2 closed, or D2, D3, and D4, closed would pose an impossible task.

To produce this kind of planning system, a system architecture such that as shown as Fig. 1.3 proved to be very effective and is still in use today.

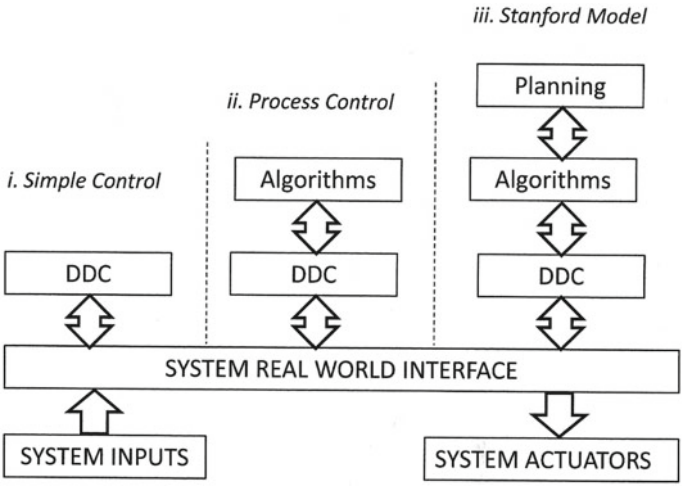

Fig. 1.3 Evolution of the components of a three-tier architecture

The lower direct digital control (DDC) level handles motion and processes physical data such as driving the wheels, stopping for obstructions, and steering the robot in response to real-world input data as it seeks to complete its task.

The algorithms section provides signal filtering, scaling, and conformance with established control laws and limits so that no unreasonable or unsafe signals are ever presented to the robot. This middle agent compares inputs from its control level to positional maps, references safety mandates, and formulates prudent courses of action which is passed according to some preset timing sequence to the lower level as motion directives for actuation. The biggest issue in this type of system is having an adaptive obstacle avoidance plan and hence the need for the third software level.

The planning level compares strategies and selects options using software that reasons. During sample periods, the top layer of the system reasons how to best enable the task to be performed and then selects what is the best options possible any given sample time. When complete or when some event clock is about to expire, the current best plan is sent to the middle layer for passage to the low level controllers. The reader may recall a similar constraint that the timer imposes in the chess playing system described in Sect. 1.1.1.

These three levels correspond in human terms to reactive (low level), skilled (mid-level), and reflective (high level) responses to internal and external stimuli. Our sensory capacities provide the external system triggers to which we respond at one of the three levels. Of course, the same set of external stimuli can often provide quite different responses in different people in differing circumstances. The notions of mood and personality present a real challenge to the design of electronic learning systems that interface with reality.

Provided mechanisms can be devised to access what engineers call *state variables* that accurately encode the real-world problem and handle appropriate actuation signals, and the SRI work demonstrated that it was possible to write programs that could not only process control data but also execute planning, search, and decision algorithms. The framework of this software uses meta-rules and game theoretic methodologies and essentially blends a *scruffy* task with *neat* planning and dynamic supervisory agents.

1.1.4 Does the BOXES Algorithm Learn?

It will be shown that the BOXES algorithm is a set of software agents that function in a similar manner to the Stanford system. Because learning is not just a rote process, the refined BOXES method includes essential aspects of forgetfulness, ambition, peer advising, and evolution all of which are present in human development.

The original BOXES experiments were used to control an inverted pendulum which was implemented using a free-swinging pole atop a cart being driven back and forth by a reversible motor while simultaneously keeping the pole from falling and the cart away from the end of its track. In the Liverpool experiment to be described later, the rig could be operated manually or in auto mode, so the progression of learning runs could be measured for the system or for human operators of varying skill levels. That the human operators were more intelligent is without a doubt, but it was obvious that balancing the system needed many training runs before controlled runs of between 5 and 10 s could be attained. The BOXES automaton quickly outperformed the human after quite short training periods. It became apparent that even expert human users, acting upon their reflexes, tend to panic or lose concentration, whereas the automaton had infinite dedication to the task, inside knowledge of the state space, and confidence in how its decisions were being implemented. Without a doubt, the performance of the system under BOXES control improved as it gained experience in its application domain. It would appear that it most certainly learned the task.

1.1.5 Can Computers Think?

Scholars still ponder the more ethereal and existential aspects of learning, intelligence, and consequence. That machines can learn how to do certain tasks using meta-rules, or rules about rules, was discussed above, but are they thinking? This begs the question: what is *thought*? And is it linked to *being*? Descartes is perhaps the most famous philosopher who sought to define reality as the outcome of cognition and introspection.

But if reality includes the faculty to mislead or lie, the well-known Turing Test [10] may well become suspect or even flawed. If leading questions such as "*are you a computer?*" or "*I am human, are you?*" are forbidden or able to be not answered

truthfully, then the test is incomplete. It would seem that Turing's contemporary Ada Lovelace objected to the whole notion of machine perception by stating "… *that computers can only do as they are told and consequently cannot perform original actions*."

Others disagree. In 2010, Versace and Chandler [11] published a thought provoking article, in which they asserted that memristors [12] may one day provide a means of creating *wetware* which is loosely defined as the elements of a non-Von Neumann type of computer architecture that will blend hardware and software. Furthermore, such systems will possess the innate ability to instantaneously reconfigure their internal structures and communication paths. This is what is thought to be similar to how the human brain behaves while recovering from accidental damage. Their research is heading toward the use of memristors to "… *simultaneously perform a full set of logic operations at the same time as they function as nonvolatile memory.*"

When such devices become commercially available, the authors describe how memristors will be integrated to form a brain-inspired chip that is purported to emulate behaviors similar to those found in human neuron axon synapse exchanges which are currently thought to constitute brain activity. But there are many questions, both technical and ethical, to be answered before such systems become commonplace or even created in the laboratory.

Machine intelligence is controversial, and its proponents such as Donald Michie often had to defend their research fiercely from the attacks of naysayers. A basic cause for such rancor is the misconception that a smart machine, performing complex tasks at incredible speed … is in some way alive and capable of sympathetic or empathetic decisions. If such is the case, then the whole issues of morality and legal culpability arise.

1.1.6 Does the BOXES Algorithm Think?

The BOXES methodology does not claim to think, but it most certainly learns how to solve difficult tasks and may provide a solution to the control of poorly defined systems. Control decisions are found by accessing values in real time from a software array called a *signature table.* The index to the signature table is computed in real time from the system's state variables to form a unique *system integer*. Using this index, the signature table is accessed, and a control decision returned. Values in the signature table are only updateable between samples or at the end of a run by blending real-time data with historical statistics. Using a statistical rewards and punishment algorithm, the system learns from its experiences. The values in the decision table effect control and may drive the system into previously un-entered and uncharted regions of the state space where conventional controllers may never have had cause to explore. While adopting some of the metaphors ascribed to human cognition, the BOXES method is certainly not thinking.