

Institute for Research in Cognitive Science

IRCS Technical Reports Series

University of Pennsylvania

Year 1997

An Architecture For Behavioral
Locomotion

Barry D. Reich
University of Pennsylvania

University of Pennsylvania Institute for Research in Cognitive Science Technical Report No. IRCS-98-04.

This paper is posted at ScholarlyCommons.
http://repository.upenn.edu/ircs_reports/51

AN ARCHITECTURE FOR BEHAVIORAL LOCOMOTION

BARRY D. REICH

A DISSERTATION

in

COMPUTER AND INFORMATION SCIENCE

Submitted to the Department of Computer and Information Science in Partial Fulfillment of the
Requirements for the Degree of Doctor of Philosophy.

1997

Dr. Norman I. Badler
Dissertation Supervisor

Dr. Mark Steedman
Graduate Group Chairperson

© Copyright 1997
by
Barry D. Reich

Acknowledgments

I would like to begin by thanking my parents, my brothers and sister, and all my friends for their encouragement over the years. This work would not have been possible without their support.

I would like to acknowledge Ken Noble for his help in producing videos and slides for various presentations of this work, and Hyeongseok Ko for the walking code that increased its realism.

Thanks to the “Zaroff” group, particularly Chris Geib, Mike Moore, and Drs. Norman Badler and Bonnie Webber, for all the guidance, criticism, and contributions to the project which motivated a large portion of this work.

I wish to thank my thesis committee, Drs. Richard Paul, Benjamin Kuipers, Dimitris Metaxas, Mark Steedman, and Bonnie Webber, for constructive criticism that greatly improved the quality of this dissertation.

Thanks to Harold Sun for help in debugging the Locomotion Engine, testing the Locomotion Control System, and for discussions and insights concerning the theoretical formalism.

I thank Brett Douville for discussions and suggestions and for all his help with the editing of both the proposal and this document.

Many thanks to Tripp Becket for providing an excellent environment in which to do this work and for countless hours of help of all kinds over the years.

I would especially like to thank my advisor, Dr. Norman Badler, for years of advice, discussions, encouragement, opportunities, and the motivation to do this work.

This research is partially supported by ARO DURIP DAAH04-95-1-0023, DARPA AASERT DAAH04-94-G-0362, DARPA DAMD17-94-J-4486, DARPA through the Franklin Institute, DMSO DAAH04-94-G-0402, JustSystem Japan, NSF IRI95-04372, ONR through the University of Houston K-5-55043/3916-1552793, the U.S. Air Force DEPTH through Hughes Missile Systems F33615-91-C-0001, and the U.S. Air Force through BBN F33615-91-D-0009/0008.

ABSTRACT
AN ARCHITECTURE FOR BEHAVIORAL LOCOMOTION
BARRY D. REICH
NORMAN I. BADLER

We present a complete architecture for behavioral control of locomotion for both real and simulated agents and provide a design methodology for building the locomotion control systems that embody the architecture. A low-level locomotion engine controls an agent's actions directly based on intermediate-level reactive behaviors such as attraction and avoidance. High-level state machines schedule and control the reactive behaviors allowing for more "intelligent" decision processes, and an agent model provides a mechanism for varying locomotion according to agent state and personality attributes.

In addition to providing specifications for a locomotion engine, we address the problem of selecting and organizing an appropriate set of behaviors. We present selection criteria and a method for partitioning the behaviors to aid in implementation. We discuss the challenges specific to human locomotion and explain how to overcome them in the system design process. Finally, we introduce the notion of anticipation to the field of behavioral control and use it extensively throughout the system to produce agents whose actions are more realistic.

Contents

Acknowledgments	iii
Abstract	iv
List of Figures	ix
I Background	1
1 Introduction	3
1.1 What is Behavioral Control?	6
1.2 Thesis Overview	8
2 Motivation	9
2.1 Biology and Ethology	9
2.2 Computer Graphics	10
2.3 Entertainment	11
2.4 Robotics	11
3 History	13
3.1 Prior to Behavioral Control	14
3.2 The Advent of Behavioral Control	14
3.2.1 Emergent Behavior	15
3.2.2 Plans as Advice	15
3.3 Survey of Control Strategies	16

3.3.1	Purely Reactive Strategies	16
3.3.2	Behavior-Based Strategies	17
3.3.3	Planner-Based Strategies	18
3.3.4	Hybrid Strategies	19
3.4	The Move Toward Layered Architectures	19
4	Statement of Work	23
4.1	Minimizing the Active Behavior Set	24
4.2	Anticipation	25
4.3	Scope	25
II	The Architecture	27
5	System Requirements	29
6	Design Methodology	35
6.1	Creating the Locomotion Engine	36
6.2	Choosing the Behaviors	38
6.2.1	Selection Criteria	38
6.2.2	Additional Properties	39
6.3	Partitioning the Behaviors	39
6.3.1	Instantaneous Behaviors	40
6.3.2	Memory-Reliant Behaviors	41
6.4	Creating the Agent Model	43
6.4.1	System Parameters	43
6.4.2	Attributes	44
6.4.3	Establishing the Correspondence	44
6.4.4	Degrees	46
6.4.5	Calculating the Results	46
6.5	Building the State Machine	48
6.6	Extensibility	50

6.7	Design Summary	51
III	A Concrete Example	53
7	A Human Locomotion Control System	55
7.1	Development Environment	55
7.2	Locomotion Engine	56
7.2.1	The Simulation Sense-Control-Act Loop	56
7.2.2	Perception	57
7.2.3	Control and Action	59
7.3	Basic Sensors	59
7.3.1	Object	59
7.3.2	Location	61
7.3.3	Line	62
7.3.4	Proximity	63
7.4	Control Behaviors	64
7.4.1	Attract	64
7.4.2	Avoid	69
7.5	Behaviors	71
7.5.1	Level 0: Reflexive and Physics-Based Behaviors	73
7.5.2	level 1: Goal-Based Behaviors	76
7.6	State Machines	77
7.6.1	Turning	78
7.6.2	Path Following	80
7.6.3	Chasing	87
7.6.4	Combining the State Machines	89
7.7	Agent Model	90
8	Issues	95
8.1	Limited Perception	95
8.2	Anticipation	97

8.2.1	Locomotion Engine	98
8.2.2	Terrain Awareness	98
8.2.3	Attraction	101
8.2.4	Avoidance	106
8.2.5	Planner-Based	111
9	Results	113
9.1	Description	113
9.2	Adherence to System Requirements	115
9.3	Examples	119
9.4	Performance Analysis	124
IV	Conclusion	127
10	Discussion	129
11	Research Summary	131
12	Future	135
V	Appendices	139
A	Additional Behaviors	141
A.1	Attraction to Deep Spaces	141
A.2	Field-of-View Avoidance	144
B	Groups	147
B.1	Why Do Animals Group?	147
B.2	Group Research	148
	Bibliography	151

List of Figures

6.1	The Architecture	35
6.2	The Behavior Function Template	36
6.3	Partitioning the Set of Behaviors	40
6.4	Partitioning the Set of Instantaneous Behaviors	40
6.5	The Memory-Reliant Behavior Template	42
6.6	The Agent Model	44
6.7	Sample Agent Model Entries	45
6.8	Sample Parameter Value Calculations: The Initial Conditions	48
6.9	Sample Parameter Value Calculations: The Results	48
6.10	The Locomotion Control System Flow Diagram	49
7.1	The Behavioral Simulation System's Central Simulation Loop	57
7.2	Calculating the Set of Possible Next Foot Positions	58
7.3	The Sensor Function Template	59
7.4	The Mathematics of an Object Sensor	60
7.5	Asymmetry Caused by the Use of Bounding Cylinders	61
7.6	The Mathematics of a Location Sensor	61
7.7	The Mathematics of a Line Sensor	62
7.8	An Undetected Wall	63
7.9	The Mathematics of a Proximity Sensor	64
7.10	The Proximity Sensor's Simulated Object	65
7.11	The Control Behavior Function Template	65
7.12	An object sensor and an attract control behavior combine to form an attraction behavior which draws the agent to the goal	66

7.13	Walking Past the Goal	68
7.14	An Orbital Approach	68
7.15	A proximity sensor and an avoid control behavior combine to form an avoidance behavior. While an attraction behavior draws the agent to the goal, the avoidance behavior steers it around obstacles.	69
7.16	Two Agents Colliding	71
7.17	The Behavior Function Template	71
7.18	The Ducking Behavioral State Machine	74
7.19	The Ducking Function	75
7.20	An Agent Ducking Under Objects	75
7.21	The Inertia Behavior Function: A Slow Agent (a) and a Fast Agent (b)	76
7.22	The Attraction Behavioral State Machine	77
7.23	A Sample PaT-Net Shown Graphically	78
7.24	The Turning Behavioral State Machine	79
7.25	Turning to Face the Goal vs. Not Turning	80
7.26	The Path-Following Behavioral State Machine	80
7.27	The Initial Conditions of the Path-Following Example	81
7.28	The environment obstacle map before (a) and after (b) expansion	82
7.29	The expanded-obstacle map after wave-propagation	83
7.30	The Weight Map	84
7.31	The Path to the Goal	84
7.32	The waypoints (a) - The waypoints drawn on the original obstacle map for clarity (b)	85
7.33	The necessary waypoints (a) - The necessary waypoints drawn on the original obstacle map for clarity (b)	86
7.34	The Agent Following the Path	88
7.35	The Chasing Behavioral State Machine	89
7.36	The BSM Preconditions	90
7.37	The Locomotion Control Machine Flow Diagram	91
7.38	How the Agent Model Affects the Agent	92
7.39	An agent exhibiting “drunk-like” behavior through the use of the agent model	93

7.40	The Inertia Behavior Function: An Intoxicated Agent	93
8.1	Enforcing the Limited Perception Policy: (a) The environment (b) The agent's view (c) Attraction only (d) Attraction and wall avoidance (e) The path (f) The agent arriving at the goal	96
8.2	(a) A sawtooth path due to potential field discontinuities, and (b) Better results through anticipating the state of the world after the agent has taken the step	99
8.3	Attraction, avoidance, and terrain awareness behaviors combine to draw the three agents to the goal in the upper left corner	100
8.4	A close-up view of an attract behavior combining with a terrain awareness behavior to guide the agent to the goal	100
8.5	The lower agent is attracted to the upper agent using a traditional attraction behavior	101
8.6	The Mathematics of the Predictive Attraction Behavior	102
8.7	The First Attempt at Predictive Attraction	102
8.8	Initial Conditions for Predicting an Intersection	103
8.9	The Second Attempt at Predictive Attraction	105
8.10	A Visual Comparison of All Three Attraction Behaviors	106
8.11	The Results of a Short, Wide Avoidance Sensor	107
8.12	The Results of a Long, Narrow Avoidance Sensor	107
8.13	The Mathematics of a Predictive Avoidance Behavior	108
8.14	A Comparison of Traditional and Predictive Avoidance Behaviors: The Start of the Simulation	109
8.15	The Middle of the Simulation	110
8.16	The End of the Simulation	110
9.1	The Locomotion Control System	114
9.2	The Locomotion Control Machine	114
9.3	The Architecture and the Interface	116
9.4	The User Interface	117
9.5	An Example	123
9.6	Sensor Time Complexity Analysis	124
9.7	Control Behavior Time Complexity Analysis	125

9.8	Behavior Time Complexity Analysis	125
9.9	Simulation Time Complexity Analysis	126
A.1	A Depth Sensor	142
A.2	A 2-Bit Range Sensor	142
A.3	Range Sensor Weighting Functions	143
A.4	An agent maneuvering through a maze using only a depth sensor for navigation	144
A.5	Attraction, avoidance, and field-of-view avoidance combine to guide the agent to the goal without it being seen by the hostile agent hiding at the bottom	145

Part I

Background

Chapter 1

Introduction

A beginning is the time for taking the most delicate care that the balances are correct.

– *Frank Herbert, Dune* –

The recent increase in the use of robots and simulated agents has motivated the search for better control strategies. Autonomous robots are used for such tasks as factory automation or the exploration of dangerous or inaccessible environments. Autonomous simulated agents are used in such applications as training simulations, movie special effects, or video games. Controlling the motion of robots or animating the motion of simulated agents can be challenging. Scripting the motion of animated agents can be difficult and time-consuming, especially in multiple-agent simulations with complex interactions among agents. Since the mid-1980's, the trend has been away from traditional AI control systems for these tasks, and toward more automatic behavior-based systems that produce agents who appear to make intelligent decisions with minimal user interaction.

Behavioral control is a well-established, convenient way to model and generate interesting behavior with very little user input. The aim of this work is to provide a complete agent architecture that employs behavioral control techniques for locomotion and to present a design methodology for system construction. The resulting locomotion control systems provide reactive, robust control of real or simulated agents locomoting from place to place while interacting with each other and with the environment.

This thesis addresses the following problems:

1. Providing a design methodology for a complete locomotion control system
2. Selecting and organizing an appropriate set of behaviors
3. Approaching the challenges specific to human locomotion
4. Using anticipation in behavioral control

As Sloman and Poli observe [140], much of the early work in AI was concerned with architectural sub-components such as vision, language, planning, or learning, rather than with complete agent architectures. Interest in complete architectures for both physical robots and simulated agents has grown recently, as exemplified by Bates [17], Georgeff [65], and Hayes-Roth [72]. Following their lead, we present a complete agent architecture for behavioral locomotion.

Behavioral control (defined and discussed in Section 1.1) is based on mathematical models of behaviors such as attraction or avoidance that are observed in animals. In the design methodology chapter (Chapter 6), we present principles for choosing an appropriate set of behaviors and for partitioning these behaviors into subsets. First, behaviors are classified according to the architectural level at which they are best implemented, either at the behavioral level, or at a higher level. Then, the behavioral-level behaviors are further partitioned into two subsets of reflexive, instinctive behaviors (typically avoidances) and goal-based behaviors (typically attractions). Section 4.1 explains why we limit the number of goal-based behaviors active at any given time to one, and justifies this restriction.

Choosing an effective set of behaviors for a system depends primarily on the type of agent it controls. Different types of agents require different types of locomotion. Reynolds' birds require general behaviors, such as collision avoidance, as well as behaviors specific to flocking, such as matching the velocity of neighboring birds and keeping to the center of the flock [130]. Wilhelms and Skinner use jet motors to propel abstract geometric objects [157] and are therefore less constrained in the behaviors they may choose. Since we have no general preconceptions about the behavior of flying tetrahedra, it is difficult to make one appear either realistic or unrealistic.

Human agents require behaviors specific to walking or running unlike rolling robots, swimming fish, or flying birds. The locomotion engine we advocate is fully capable of handling the specific

challenges posed by human locomotion. Much of our discussion will relate to the domain of simulated human locomotion, but the ideas are independent of that particular domain. The architecture supports general locomotion (with some restrictions as discussed in Section 4.3).

The notion of **anticipation** is noticeably lacking in the field of behavioral control. Regardless of the domain, most systems incorporate attraction and avoidance models that sense and react to the immediate world with little or no concern for the future. This is particularly unrealistic for simulating human agents who “think ahead” while navigating. We introduce anticipation at all levels of the architecture.

The locomotion control system embodying our architecture controls locomotion at three levels. At the lowest level, a locomotion engine controls agent locomotion directly. Based on a set of reactive behaviors associated with the agent, it makes final choices about where the agent will place its feet at each step, how the agent will fly, where the agent will roll, *etc.* The set of reactive behaviors constitutes the intermediate level. Each of the reactive behaviors can be active or inactive; the locomotion engine ignores those that are inactive. The highest architectural level is a state-machine implementation of behavioral schemata which controls agent locomotion indirectly by activating, deactivating, and parameterizing the reactive behaviors according to a state-based decision process.

In addition to the three main architectural levels, we use an agent model to configure the state-machine and to contribute to the parameterization of the behaviors and the locomotion engine. The resulting locomotion style reflects the agent’s personality as well as its physical and mental state. Just as Cremer *et al.* use state and personality attributes such as velocity, patience, and aggression to parameterize their virtual drivers, giving different vehicles different observable behavioral characteristics [45, 46], we use attributes such as velocity, inertia, awareness, and caution to parameterize our agents, resulting in varied observed behavior.

The locomotion engine (based on Becket’s behavioral simulation system [18]) communicates directly with a system that calculates joint angles to simulate walking [82, 83, 84], choosing each footstep for the agent based on a set of behaviors associated with that agent. The behaviors are variations of attraction and avoidance and are parameterized by such attributes as awareness, curiosity, fatigue, and intoxication. The agent’s observed behavior is the result of the choice of behaviors and parameters which are themselves the results of choices made at higher levels.

There are limits to what low-level reactive behaviors can achieve. When the desired complexity of an agent's behavior increases beyond this limit, higher-level locomotion reasoning must be introduced. Locomotion reasoning determines the characteristics of an agent's locomotion: *i.e.*, what types of attractions and avoidances with what choice of parameters will achieve the goal. An effective implementation of relatively complex behaviors requires a multi-level approach: one or more high-level organizational structures atop the low-level locomotion engine.

Our state-machine level provides this high-level organizational structure and supports locomotion reasoning. It makes decisions necessary for complex behaviors such as *path-following* (Section 7.6.2) and *chasing* (Section 7.6.3) where behaviors and parameters change over time. In addition to changing behaviors and parameters, the state-machine structure allows us to evaluate arbitrary conditions, passed as arguments, increasing the system's flexibility. If any one of these conditions is ever achieved, the state-machine takes an appropriate action. These conditions allow the agent to interrupt the current plan when an unexpected situation occurs, *e.g.*, when something of interest enters the agent's field-of-view. An appropriate message is passed up to the user or calling system allowing it to replan, potentially taking advantage of an opportunity.

One of our design goals was to create a cognitive research tool, a system that could be used as a testbed for cognitive theories, experimental algorithms, or heuristics. To that end we emphasized modularity and extensibility. Behaviors, for example, are not hard-wired, but can be added, removed, or modified with ease. Therefore, it is possible to test new behaviors or new implementations of existing behaviors by adding or swapping in the appropriate modules. The same is true of other parts of the system such as the state machines and the agent model. The path-generation mechanism (part of the path-following behavior, Section 7.6.2), the extrapolation or exploration algorithm (part of the chasing behavior, Section 7.6.3), and the set of agent attributes and system parameters (part of the agent model, Sections 6.4 and 7.7) may all be modified, removed, or replaced for the sake of experimentation.

1.1 What is Behavioral Control?

Behavioral Control has established itself in the fields of robotics and computer animation as a viable method of controlling real and simulated autonomous agents. Individual behaviors connect

sensors to effectors and combine and interact, producing complex observed behavior with desired properties (Brooks' notion of **emergent behavior** [29]). Behavior-based agents are more reliable than classical, artificial intelligence-based agents when adapting to a changing environment.¹ They require less computation, little or no memory, and minimal resources.

Behavioral control is a more general term for behavioral animation which includes real agents such as robots as well as simulated, animated agents. It is a control strategy for real or simulated agents that is usually applied to locomotion. Neither path-planning nor explicit instructions drive agent locomotion; agent control and apparent behavioral complexity result from the interaction of a few simple "behaviors" with a complex and changing environment. A **behavior** (Section 7.5) is a function mapping an agent's state in the environment to the stress of being in that state. An agent learns about its state in the environment through the use of a network of simulated sensors. Based on the information gathered by these sensors, the path through the terrain is computed incrementally, allowing the agent to react to unexpected events such as moving obstacles, changing terrain, a moving goal, or the effects of limited perception [85, 128].

Simulated sensors detect environmental features such as terrain type and obstacle and agent locations. A locomotion engine (Section 7.2) analyzes the combined sensory input and decides where the agent should move. This decision is made after each step the agent takes to support a dynamic environment.

Brooks explains two of behavioral control's central ideas as they relate to robotics [33]:

Situatedness: The robots are situated in the world - they do not deal with abstract descriptions, but with the "here" and "now" of the environment that directly influences the behavior of the system.

Embodiment: The robots have bodies and experience the world directly - their actions are part of a dynamic with the world, and the actions have immediate feedback on the robots' own sensations.

These ideas apply equally well to simulated agents situated in a simulated world. These agents perceive the world with sensors that mimic the actions of real sensors, reporting the same information under similar circumstances. Furthermore, these agents affect the world in the same way. Their actions are part of a dynamic with the simulated world.

¹See discussions in Chapters 2 and 3 for justification.

1.2 Thesis Overview

This thesis consists of four parts: background information, a presentation of the architecture and design methodology, complete examples of the architecture and a thorough description of all its parts, and a conclusion.

The next chapter discusses motivations for behavioral control techniques in a variety of different fields from Artificial Intelligence to Robotics. Chapter 3 gives a detailed history of the study of locomotion from Eadweard Muybridge's work in the nineteenth century [116] to the simulations of the present and includes a survey of control strategies. Part 1, Background, concludes with the statement of work.

Part 2, The Architecture, presents the architecture, its motivating system requirements (Chapter 5), and a detailed design methodology for building a system (Chapter 6).

In Part 3, A Concrete Example, we build a locomotion control system for simulated *human* agents giving a full description of every part of the system (Chapter 7). Chapter 8 discusses **limited perception** and **anticipation**, important issues in achieving realism. Part 3 concludes with our results including examples of the system in use (Chapter 9.3) and a performance analysis (Chapter 9.4).

We conclude in chapters 10, 11, and 12 with a discussion, a research summary including the scientific contributions of this work, and suggestions for future work.

Chapter 2

Motivation

If endowed with interesting, realistic behaviors, autonomous agents should appear to make intelligent decisions indistinguishable¹ from those of a real agent in the same situation. Additionally, they should do so with minimal user interaction; as much of the process as possible should be controlled by the system. As Wilhelms and Skinner observe, behavioral control conveniently generates interesting locomotion with relatively little user input [156, 157].

Behavioral control techniques are motivated by Biology and Ethology, Computer Graphics, Entertainment, and Robotics, discussed in subsequent sections.

2.1 Biology and Ethology

A growing dissatisfaction with “classical” AI control systems has led autonomous agent research to the field of biology, and more specifically the field of ethology (or ecological psychology [51]). The idea of emulating nature enjoys a long tradition in many scientific fields and has been found to be vital to behavioral control. Researchers attempt to understand the mechanisms animals use to demonstrate successful adaptive behavior and then simulate and utilize these behaviors in their control systems [21, 97]. In robotics, for example, the study of how animals grasp, manipulate objects, and locomote has led to improved robotic versions of the same tasks.

Anderson and Donath present a detailed and interesting review of relevant research in the field of animal behavior [7]. They motivate the use of such architectural components as low-level

¹Although we accept less, this is behavioral control’s ultimate goal.

reflexive behaviors, a variety of basic sensors and behaviors, a state-based architectural level, and preconditions for complex behaviors by examining how each of these is exhibited by and advantageous to different animals. Our system also exhibits and benefits from these components.

2.2 Computer Graphics

The field of computer graphics, or more specifically, computer simulation, motivates behavioral control both in combination with other fields and in its own right. Often when robotics researchers wish to use behavioral control techniques in their robots, they will first simulate the robots. Simulations are useful for initial testing because of the following desirable properties [20]:

- It is cheaper and faster to construct and modify the agent.
- Execution can be much faster. If the goal is to develop a control strategy for a robot, more simulations may lead to a better control strategy.
- The designer can abstract over limitations in hardware speed by running in simulation time as opposed to real time.

Boulic *et al.* and Renault *et al.* discuss synthetic vision, an animation approach where a simulated visual sense is used for navigation by a synthetic actor. Vision is the only channel of information between the actor and its environment [25, 129]. Ullman and Steels explain that despite enormous efforts to extract general purpose symbolic descriptions from visual data [99], no such “visual module” is expected to arrive soon [143, 152]. Another advantage of simulation is the ability to abstract over the unavailability of general machine perception. Agents embedded in a simulation can be supplied with high-level results of perception directly [14].

Behavioral control can be seen in several computer simulation domains. It can be used to animate and analyze human motion [75], and to create interesting autonomous agents for simulators for training purposes [97]. Simulation of autonomous agents is an end in itself. Simulations are used in military training, and in driving and flight simulators. Cremer *et al.*, for example, have been working on the creation of autonomous driving behavior for interactive driving simulations [45, 46]. Maes discusses software agents that have been proposed as one mechanism to help computer users deal with work and information overload [96, 97].

Possibly the strongest motivation is that of multiple-agent animations. Multiple agents along with complex, dynamic environments result in more interaction and require careful choreography. An animator may have to make a series of adjustments until everything looks right. Reynolds explains that scripting the path of a large number of individual objects using traditional computer animation techniques would be tedious. Given the complex paths that birds follow, for example, it is doubtful this specification could be made without error [130, 132]. Behavioral control obviates this tedious and time-consuming animation technique.

2.3 Entertainment

Behavioral control and autonomous agents are beginning to make their way into the realm of entertainment. Examples include video games, simulation rides, movies, animation, animatronics, theater, puppetry, toys, and even party lines [97]. Maes mentions one researcher who incorporated an autonomous agent into a text-based Multi-User Simulation Environment (MUSE) system [107]. Reynolds created an animation called “Stanley and Stela in Breaking the Ice” [131] in which he modeled flocks of birds and schools of fish using a behavioral control system. His algorithms were later applied to bats in *Batman II* and to the stampede in *The Lion King*. Tu and Terzopoulos, and Terzopoulos *et al.* modeled realistic fish behavior producing entertaining, short, animated movies such as “Go Fish!” [150, 146]. Chris Crawford, a noted game designer, complains that “Artificial Intelligence is almost always the weakest part of game design” [44]. The quest for better “monster AI” has resulted in an increasing use of behavioral control techniques throughout the game industry.

2.4 Robotics

In robotics, behavioral control systems control all kinds of robot locomotion from rolling and hopping to walking and running. The primary motivation is autonomy. The goal is to develop real-time, autonomous robots that can operate in general, unstructured, unconstrained environments. In many instances, behavioral control has successfully achieved this goal.

Motivation for real-time, autonomous robots includes factory automation, navigation and retrieval tasks in outdoor environments, and map building. In addition, there is much interest in

extraterrestrial autonomous rovers [63, 110], robots for surveillance, exploration, and other tasks in environments that are inaccessible or dangerous for humans [97]. Bonasso, Antonisse, and Slack use a reactive system with layered control modeled after Brooks' subsumption architecture (explained in Section 3.3.1) which they find to be useful in an unpredictable, partially unknown environment [24]. This work is comparable to Miller's efforts on the JPL Rover [109, 110] which combined Firby's RAPs [54] with Gat's ALFA circuit language [56, 58].

Chapter 3

History

Locomotion is an active process by which one moves from place to place [76]. Locomotion attributes include: starting, stopping, changes in speed, alterations in direction, and modifications for changes in slope or for maneuvering over, under, or around obstacles. The type of terrain and its slope effect the energy cost of walking [134]. Humans tend to minimize the metabolic energy required to get from one place to another [77], walking or running depending on size and speed. For example, adults start running at about 2.5 meters per second, children at lower speeds [6, 145].

Eadweard Muybridge¹ (1830-1904), known as “the father of motion pictures,” pioneered the application of photography to the study of human and animal locomotion. He took photo sequences and then projected them onto a screen with a device he called a “zoopraxiscope” resulting in the world’s first illusion of moving pictures [117, 118, 119, 120].



¹For more information on Eadweard Muybridge that is presented in a nice format, start at “<http://www.linder.com/muybridge.html>” on the world-wide web.

Animal locomotion has been a subject of serious study for over a century. Since the early days of computers and computer science, researchers have been trying to apply their observations in an attempt to get simulated animals and robots to navigate automatically. They applied control strategies such as behavioral control and planner-based techniques, embedded in simple or layered architectures, to individual agents and to groups. This chapter discusses their results.

3.1 Prior to Behavioral Control

Prior to behavioral control, researchers, particularly in the field of robotics, treated the problem of obstacle avoidance statically. They analyzed the environment and made a plan assuming the environment would not change. At best, if an environmental change was detected, the robot would stop, replan, and then start again, operating with the same assumption (Adams cites the work of Cox and Elfes as examples [43, 52]) [1].

Previous techniques often involved symbolic reasoning which attempts to construct a plan to achieve a goal using logical transformations on symbols that represent a model of the world. Symbolic reasoning techniques began with STRIPS [53] and ended with Chapman's proofs of the intractability of the general planning problem [36] (see the discussion in Section 3.3.3). They were abandoned for many reasons (*e.g.*, their combinatorial expense and brittleness) and replaced by behavioral control techniques which do not suffer from the same problems.

3.2 The Advent of Behavioral Control

Recent work, first in robotics and then in computer animation, has motivated the search for more automatic methods of controlling motion. Duchon *et al.* [51] describe a revolution from the slow, intensely computational "sense-model-plan-act" architecture of such robotic systems as SRI's Shakey [122] and Moravec's CART [114] to the current architectures that remove the "model-plan" part of the loop, replacing it with simple, fast "control."

Out of discontent with traditional methods and increasing willingness to sacrifice complete control for realism grew the increasing popularity of behavioral control [14]. Previously, for computer animation, the animator was responsible for scripting an agent's path. This awarded absolute control over all locomotion components, but was time-consuming and repetitive [69] and

did not promote thinking about locomotion in natural qualitative terms [98].

The behavioral control method of designing autonomous agents derives from ethology, the scientific study of animal behavior. Agents are endowed with simple behaviors such as attraction and avoidance. A behavioral engine, often implemented as a sense-control-act (S-C-A) loop, evaluates the state of the environment and generates motion according to certain rules of behavior. Rather than scripting an agent's exact movements, the user need only choose an appropriate set of behaviors (not always an easy task). The system takes care of the details of simulating these behaviors; the agent discovers its own path through the environment.

3.2.1 Emergent Behavior

A key idea in behavioral control systems is that of **emergent behavior** [29]. Brooks' modern notion of emergent behavior can be traced back at least as far as Herbert Simon who pointed out, in the late 1960's, that complex observed agent behavior is often due to a feedback mechanism interacting with a complex environment and not complexity in the agent itself or in the control system [137]. It may be an observer who ascribes complexity to an organism, not necessarily its designer. More recently, emergent behavior was popularized by Braitenberg who claimed that simple networks can produce complex motion from which observers infer intelligence. Braitenberg described vehicles whose seemingly intelligent, even emotional behavior results from sensory stimuli passed through a network of nodes, eventually driving motors [26].

Emergent behavior is a result of the interaction of an agent with a dynamic environment. It is characterized by the manifestation of global states or patterns which are not explicitly programmed in, but result from local interactions among the system's components and with the environment [102]. Because emergent phenomena are by definition observed at a global level, they depend on the existence of an observer [105]. It is the observer who attributes emotions, intelligence, or will to the resulting observed behavior. For further relevant discussions on this topic, see Langton [88, 89], Maturana and Varela [106], or Steels [144].

3.2.2 Plans as Advice

One of the important ways in which behavioral control differs from planning is that behavioral control uses plans to *guide*, not *control* action [4]. Gat and others strongly advocate Agre and

Chapman's **plans-as-advice** or **plans-as-communication** theory which suggests influencing action rather than controlling it. Slack's Navigation Templates for mobile robot control are an example of the plans-as-advice theory. He uses constraints to bound the robot's motions, rather than dictating a particular trajectory. In that way the plan adapts to the robot's changing perceptions of the world [139].

This is similar to Gat's ATLANTIS [59] architecture where a traditional planning system is combined with a reactive control mechanism. He uses the results of planning to guide the robot's actions but not to control them directly. A sequencer uses the planner's output only as advice and also takes into consideration other sensory information in making its decisions. This too can be viewed as a concrete implementation of Agre and Chapman's plans-as-advice theory. In addition to implicit use through the utilization of behavioral control, we use the plans-as-advice theory explicitly; see, for example, our path-following implementation discussed in detail in Section 7.6.2.

3.3 Survey of Control Strategies

Autonomous agents, whether simulated or real, are controlled through an architecture embodying a particular control strategy. We survey purely reactive, behavior-based, planner-based, and hybrid control strategies that combine two or more of these approaches. For an excellent, detailed discussion, see Matarić [101].

3.3.1 Purely Reactive Strategies

The simplest control strategy, conceptually, is the purely reactive one implemented as a set of condition-action pairs (*e.g.*, Brooks and Connell [34, 41] and Agre and Chapman [3]). The biggest advantage of these systems is their speed, due to their minimal computation. They consist of a set of purely reactive rules, containing little or no internal state. They simply use sensor readings to index into this set, selecting and executing the corresponding action. A direct coupling between the sensors and actions results in fast, efficient operation [101].

A number of researchers, including Connell [41] and Gat [62], have demonstrated purely reactive robots with various degrees of navigational ability. One of the most prominent and

ground-breaking researchers in the field, Brooks began publishing papers on his **subsumption architecture** for mobile robots, the most famous of the purely reactive control strategies, in 1986. Brooks argues that to create artificial intelligence:

We must incrementally build up the capabilities of intelligent systems, having complete systems at each step of the way and thus automatically ensure that the pieces and their interfaces are valid [32].

Brooks' subsumption architecture is a layered control system where higher-level layers can subsume the roles of lower levels by suppressing their outputs. However, in general, lower levels continue to function as higher levels are added [29, 30]. The lowest level is the collision avoidance level. Robots steer around objects locally and halt in emergency situations, a behavior Brooks finds necessary in a dynamic environment. The next level is goal-oriented through the use of attraction; it allows robots to go places.

Opponents of purely reactive control systems feel that they are limited and less powerful than behavior and planner-based systems. The designer must be able to predict all relevant world states that can be sensed and all corresponding actions relative to the agent's goals. All the agent's goals must be defined at design-time, a significant limitation. Sufficiently general reactive systems would be prohibitively complex according to Mataric [101]. For example, in reactive systems where agents have no internal models, it is often the case that a goal can be achieved only through fixation. If at any time the goal is imperceivable (as when an obstacle obscures the agent's view), the goal will be lost [143]. A behavior such as *chasing* (Section 7.6.3), where at times an agent loses sight of its target, can not be implemented effectively with a purely reactive control system.

The next step in the evolution of these control systems is clear. Many researchers have examined the prospect of combining reactive and deliberative control including Arkin [11], Georgeff [66], Kaelbling [79], Payton [124], and Soldo [141], to name a few. We discuss this new direction in Section 3.4.

3.3.2 Behavior-Based Strategies

Behavior-based control systems are commonly viewed as a superset of reactive systems. As opposed to the reactive approach, which uses condition-action pairs, behavior-based systems are

based on combinations of “behaviors.” Combinations of behaviors yield a more general, and thus more powerful, framework on which to build a control strategy.

Behavioral control has proven to be an effective control strategy, succeeding in complex domains where symbolic reasoning fails. Particularly in the domain of collision-free navigation, behavior-based agents are typically much more reliable than classical agents while requiring only a small fraction of the computational power [61]. Behavioral control is fast, often supporting real-time implementations; it is robust, allowing for a wide range of domains; and it makes local, incremental decisions, allowing for a dynamic environment. There are, however, several problems with pure behavioral systems: they lack intelligence and decision-making capabilities; they can be very difficult to design; they often produce highly non-optimal solutions; and they can get stuck in behavioral local minima or cyclical behavioral patterns.

These problems have forced behavioral control systems to evolve toward the layered control systems, discussed in Section 3.4, in the same way that reactive systems have evolved. Behavioral control is combined with deliberative, often state-based control resulting in an overall improvement in operation through high-level mediation and intervention.

3.3.3 Planner-Based Strategies

Planner-based strategies employ a centralized world model for sensory information, verification, and action generation (*e.g.*, Chatila and Laumond [39], Giralt *et al.* [68], and Moravec and Cho [115]). The biggest difference between planner-based strategies and other strategies is that a planner uses information in the world model to generate a sequence of actions, a complete plan (*e.g.*, Nilsson [121]), as opposed to the generation of a single action such as a footstep [101, 121]. As a result, planners require a large amount of information about the world to be effective.

While planners used in concert with other control systems have had promising results, the feeling that pure planning systems are unsuited to autonomous agent control is pervasive in the literature (*e.g.*, Beer, Chiel, and Sterling [21], Brooks [28, 30], Chapman [37], Ginsberg [67], and Mataric [101]). Real situations generally cannot be completely represented because planning is inherently combinatorially explosive, making the systems relying on these limited models brittle, inflexible, and slow [4, 32, 36, 94, 143].

Agre and Chapman observe that real situations are characteristically *complex, uncertain*, and

immediate requiring the ability to decide what to do *now*, so speed and short-term decision-making are important:

Life is fired at you point blank: when the rock you step on pivots unexpectedly, you have only milliseconds to react. Proving theorems is out of the question [4].

Kaelbling stresses the need for intelligent systems operating in complex, unpredictable environments to be reactive:

A robot that blindly follows a program or plan without verifying that its operations are having their intended effects is not reactive. For simple tasks in carefully engineered domains, non-reactive behavior is acceptable; for more intelligent agents in unconstrained domains, it is not [80].

Gat agrees, pointing out that unexpected changes in the environment can invalidate the stored information and lead to erroneous actions [61].

3.3.4 Hybrid Strategies

Hybrid architectures are a compromise between purely reactive or behavior-based approaches and pure planner-based approaches. They include both, combining a low-level behavioral or reactive system with a high-level planner for decision making. The low level is responsible for the agent's instinctive behavior (*e.g.*, collision avoidance), as well as goal-oriented attraction-based behaviors. The high level introduces "intelligence" into the system allowing for action selection and increased control over the low level. Hybrid architectures have recently increased in popularity (*e.g.*, Arkin [10], Payton [125], and Connell [42]). They have been found to successfully overcome the limitations exhibited by the individual strategies, discussed in previous sections.

3.4 The Move Toward Layered Architectures

Two common complaints with pure behavioral control are that users cannot control behaviors at a high level of abstraction and that selecting and correcting behavioral parameters is difficult; hand coding them as needed is a laborious process. The limitations of reactive behavioral control

suggest that “intelligent” behavior requires both reactive behavior with response to stimuli and a certain amount of stored information [12, 13, 15, 20, 70]. As Gat notes:

Avoiding internal state is tantamount to assuming that nothing about the world can usefully be predicted [60].

Clearly, this assumption is invalid; several researchers have pointed out that everyday activity is mostly routine and predictable. Furthermore, stored information such as the layout of a building in which an agent is situated is necessary for non-local navigation such as walking from one room to another. This task is impossible using only local, instantaneous sensor information because in most places, no information is available to indicate the direction of the destination. Some state information (that is, a world model) is required: a map, prior knowledge about the layout of the halls, and so on.

Added intelligence or decision-making capabilities can be achieved through the combination of a reactive or behavioral control layer with a layer for storing information. Many systems of the last decade have used this kind of “layered control” including: Ahmad *et al.* [5], Cremer *et al.* [45, 46], Firby [54], Gat *et al.* [57, 59, 63], Georgeff and Lansky [66], Kaelbling [78], Kuipers and Levitt [87], Soldo [141], Tyrell [151], and Wilhelms and Skinner [157]. Reactive behaviors configuring a sense-control-act loop drive an agent at the lowest level or levels; various kinds of planning or decision-making structures make up the highest level. Becket argues persuasively for the two-level approach [20] as do Wilhelms and Skinner:

A combination of high-level and low-level techniques combined in an intelligent fashion will distinguish the animation systems of the future [157].

A wide range of high-level techniques have been used for encoding, scheduling, and controlling low-level reactive behaviors including programming languages, planners (hybrid architectures), and state-machines. Gat’s ALFA [56] is a programming language for reactive control mechanisms for autonomous mobile robots similar to Kaelbling’s REX [78], except that it can also support a subsumption architecture. Firby’s Reactive Action Packages (RAPs) are the basic building blocks for a situation-driven execution system. A RAP completely describes the execution of a task. It contains the task’s goal satisfaction test and methods to achieve the goal in different situations [54].

The **Hide and Seek** project [12, 13, 15, 112, 113, 149]² demonstrated the usefulness of an AI planner in introducing reasoning to a behavioral control system. It also motivated the addition of the state machine layer to the architecture. The Hide and Seek system required complex path-following and chasing behaviors that could not be simulated with pure behavioral control. A higher-level organizational structure was required. Following the lead of researchers such as Mataric [104], we chose the popular state-machine paradigm which successfully introduced the needed intelligence into our system.

²The Hide and Seek project combines Geib's hierarchical planner, ItPlanS [64], and Moore's search planner [111] with the locomotion control system (LCS), presented in Chapter 7, to produce a fully automatic simulation of the game of Hide and Seek. The system (Zaroff) employs a layered architecture to control player locomotion. LCS controls locomotion directly. The planner level achieves indirect control by sending locomotion requests to LCS. LCS controls, schedules, and coordinates reactive behaviors to simulate the requested locomotion.

Chapter 4

Statement of Work

The problem this work addresses is that of synthesis and control of locomotion for simulated agents or robots. Synthesis includes complete navigation from the starting point to the goal and all decisions contributing to agent locomotion choices. Control includes manipulation of locomotion parameters affecting the resulting observed agent behavior. The interface to the control system must be high-level and easy to use; commands such as **go to location X** or **go to object X** should be sufficient to generate realistic, reactive, robust locomotion taking the agent to the goal. The solution is to provide specifications for a **locomotion control system** combining behavioral control, state-based reasoning, and an agent model to parameterize the system.

We present a complete layered locomotion control system along with a discussion of each part of the system and a detailed design methodology to facilitate construction. Beginning with a set of requirements to which the system must adhere, we present design methodologies for each architectural layer, keeping the requirements in mind. We explain how to build a locomotion engine, how to choose an appropriate set of behaviors for the task at hand, how to build a state machine to introduce high-level decision-making capabilities, and how to incorporate an agent model to parameterize all system layers.

Although the majority of what is presented in this thesis can be generalized to any animal or robot, some of the discussion is motivated by and addresses issues specific to **human** locomotion. Simulating humans differs from simulating robots, fish, birds, or abstract objects due to our unique locomotion. No system addressed these differences adequately until Becket developed his Behavioral Simulation System in 1993 [18]. Building on Becket's work, we have extended his

system to reflect the human decision-making process more accurately while retaining the ability to model other animals or robots.

The following sections discuss minimizing the active behavior set, the notion of anticipation in behavioral control, and the scope of this work.

4.1 Minimizing the Active Behavior Set

A common objection to the traditional approach to behavioral control is that an implementation requires an unreasonable amount of weight “fiddling” or parameter tweaking [14]. Emergent behavior becomes overwhelmingly difficult to control when the behavioral set becomes large as a result of the combinatorial complexity of specifying interactions among behaviors or in finding reasonable behavior weights. Many researchers have pointed out this problem including Brooks [31], Chapman [38], Maes [93], and Mataric [101].

Some have tried to solve this problem through the application of learning algorithms. Unfortunately, current learning algorithms tend to be too general, requiring domain-specific knowledge to be effective in the particular domain. Our solution is parsimony. We claim that a small set of avoidance behaviors combined with at most **one** active attraction behavior is sufficient to achieve realistic locomotion.

Two competing attraction behaviors will cause an agent to wander back and forth between the two goals in a “figure eight.” When a human agent, for example, must go to the post office and to the bank, it is unrealistic for that agent to combine these goals at the behavioral level. Clearly, the agent should not be attracted to the weighted average of the two locations. A better solution, one that more accurately reflects the human decision-making process, is to have a higher level system resolve the problem into a sequence of single-goal attractions, configuring the lower level to appropriately generate locomotion. We use a prioritized queue of goals where only the first is the active attractant. If an opportunity to satisfy another goal presents itself, the priorities can be changed, possibly resulting in queue reordering.

Anderson and Donath agree that it is necessary to separate the mechanism which allows for control of agent behavior from the actual devices which affect each reflexive behavior [7]. We

therefore assign the task of scheduling and controlling low-level behaviors to a high-level state-machine. We minimize the set of active behaviors by implicitly encoding the behavioral hierarchy into the state machine, reducing the problem from an $O(n^2)$ behavior interaction to an $O(n)$ behavior choice.

4.2 Anticipation

Matarić discusses the implications of homogeneous agents embodied with similar abilities and goal structures [104]. “Identical and similar agents have innate knowledge of each other ... their behavior is implicitly or explicitly predictable to each other.” The following example is attributed to Brooks:

If when driving on a two-lane road we encounter an oncoming car, we are confident that the right behavior is to stay in our lane, since the other car will follow the same strategy. However, if instead of a car an elephant is approaching, there is no clearly right behavior since there is no way of predicting what the elephant will do.

Humans understand the way other humans behave and tend to use this understanding, consciously or unconsciously, in their locomotion choices. Humans anticipate each other. A significant contribution of this work is to introduce the notion of anticipation to the field of behavioral control.

4.3 Scope

We limit the scope of this work by allowing no learning, planning, or interagent communication. Our approach differs from that of Ridsdale, for example, whose agents used neural networks to learn skills [133]. As discussed in Section 4.1, learning techniques for behavioral weighting are not yet powerful enough for our domain. Instead we solve the problem of behavioral weighting by minimizing the number of active behaviors at any given time. Our architecture does not include a planner, but supports one through a high-level interface. As discussed in Section 3.4, the Hide and Seek project demonstrated this capability.

Although we discuss an implementation of a terrain awareness behavior in Section 8.2.2, we do not attempt to build up a map of the environment. Our agents are goal oriented and are only

interested in getting from here to there. The system activates, guides the agent to its goal, and then deactivates; information is not retained from one activation to the next. This was our choice in an attempt to support a dynamic environment including changing terrain. For references on mapping, see Elfes [52], Kuipers [87], Matarić [100], and Miller and Slack [109].

Brogan and Hodgins' work, among others', includes explicit communication through message passing [27]. We use no explicit interagent communication. All information is acquired through the use of simulated sensors that probe the environment and, in the case of computerized agents, is filtered to limit an agent's perception to that which a real agent in a similar situation would be able to sense (Section 8.1). This derives from the ethological concept of *stigmergic* communication that is exhibited by some animals and insects who exchange information only indirectly by sensing each other's external state [102].

Our architecture is designed, primarily, to support unaided, discrete, legged locomotion. By unaided, we mean that we have not attempted to support locomotion such as skiing or skating. By discrete, we mean that we have not tried to model flying, unless that flying is discretizable (for birds, one cycle of flapping might be a discrete unit). By legged, we mean that we have not attempted to model swimming, for example. Some of these locomotion types may be simulated using modified versions of our system, however, we leave that to future work.

No existing system claims to produce provably accurate, realistic animal behavior that is indistinguishable from real animal behavior, nor is it likely that any system of the near future will make this claim. Our system is no exception; accuracy is left to the designer. Our goal is to provide a layered architecture that facilitates realism and accuracy through versatility and modularity, providing the designer with the tools necessary to model a wide variety of behaviors and an architectural structure and design methodology well suited to experimentation and quick, intuitive adjustments.

Part II

The Architecture

Chapter 5

System Requirements

We present here a complete agent architecture for behavioral locomotion and a design methodology for constructing it. We begin, in this chapter, by presenting the system requirements that motivated its design. The system must be:

Aware	Fault Tolerant	Purposeful
Dynamic	Flexible	Realistic
Easy to Use	Modular	Robust
Efficient	Opportunistic	
Extensible	Parameterizable	

Aware: An agent must be made aware of its environment. Achieving this sort of awareness is an easy problem to solve, typically through the use of a network of simulated or real sensors. When polled, the sensors sense the environment and report their results. The greatest challenge is to choose the set of sensors that best reflects a real agent's sensory capabilities, or the set that provides the robot with the most useful information appropriate to its assigned tasks (the latter being a problem with no single clear solution).

Dynamic: The agent must be able to cope with a changing environment in a timely fashion. This is subtly different from achieving environmental awareness. To support a dynamic environment, the agent must *remain* aware of the environment at all times, even when its controlling architecture is involved in intense computation. It is dangerous to invoke time-consuming systems such as planners or vision systems without providing a mechanism of awareness during that time. For example, imagine a chicken crossing the road. Suddenly

and unexpectedly, a car turns the corner and heads toward the chicken. A chicken that is aware of its surroundings would be able to speed up to get to the other side before being hit, while one whose locomotion control system was involved in intense computation would not be so lucky. Clearly it is important for the chicken to remain aware of its surroundings, even while pondering something else.

The problem of supporting a dynamic environment through maintained awareness can be solved in two ways. The first is to establish a limit to the amount of computation allowed during each phase of the S-C-A loop (assuming a behavioral control implementation). A tighter loop decreases the time between successive environmental samples. The second solution is to parallelize the system; computationally intensive processes may be performed in parallel. Meanwhile the agent continues to interact with its environment; it remains aware of the world.

Easy to Use: One of the major motivations behind behavioral control is to provide a way to generate reasonable animation (or robot control) with very little user input. The motivation behind architectures for behavioral control systems is to further reduce the user's burden by insulating him or her from system details. These architectures must be easy to use; once the user inputs a request the agent must operate autonomously. An animator or operator would not choose to use a system involving constant intervention if doing the work by hand were easier.

Good systems are versatile, providing the user with many options for a variety of observed behaviors, but these options should not be a hindrance. Default values should be provided. Furthermore, upon request from the user, the system should provide detailed information on what is occurring internally to facilitate generating the desired results. Few systems work perfectly every time.

Efficient: We want the system to run quickly and efficiently. To that end we advocate simplicity at every stage of design, a view shared by Agre and Chapman [3], Brooks [32], Soldo [141], and others. Algorithms should be constant time whenever possible and should require minimal computation to support a real-time implementation.

Extensible: The system must be extensible at every level. Methods should be established for adding to each system module that take into account the effect of the new part on the existing system. It is not always sufficient to allow for the addition of a new behavior, for example, without putting careful thought into how it will interact with the existing set of behaviors.

Flexible: The architecture should support as wide a variety of behaviors as possible. The recent trend in maximizing flexibility has been to combine two or more control strategies into one hybrid architecture, or to layer the architecture employing a form of decision-making atop a low-level reactive or behavioral layer.

Fault Tolerant: The real world is rife with uncertainty. Actions sometimes fail to produce desired effects, unexpected events occur, and robotic sensor data can be noisy, incomplete, and imprecise. A robot acting in the real world or a simulated agent acting in a simulated world must anticipate problems that may arise and plan for them. First demonstrated by Firby [54], this is known as both **cognizant failure** and **flexible plans** and is discussed by many researchers including Agre and Chapman [3], Gat *et al.* [60, 61], and Slack [139].

According to Gat, obtaining reliable sensor data is one of the thorniest problems in mobile robot navigation. Often, this problem is resolved by applying the notion of cognizant failure, typically through a high-level reasoning process. Another problem solved by this technique is that of “getting unstuck.” When an agent is stopped by a combination of obstacles, it must turn away from the obstacles so that it can proceed. If this problem were anticipated, the system would detect the situation and resolve it. Otherwise, the agent might halt or wander in a circle, getting caught in a local minimum. Gat and Slack note that the choice of whether to turn right or left to avoid the obstacles is non-trivial [57, 138].

Modular: At times one must abstract over the unavailability of certain parts of the system. For example, in computer graphics, vision is often simulated through ray casting or z-buffering algorithms combined with database queries, for lack of sufficiently powerful or efficient image processing techniques. We claim that although we make these simplifications, we do not lose generality if we take a modular approach to system design; as replacements become available, they can be plugged in. In robotics applications, sensors can be replaced when better ones become available. Kaelbling strongly advocates modularity, pointing out that it

makes a system easy to implement and understand [80].

Opportunistic: As an agent locomotes through its environment, it may have a single goal or it may have multiple goals. As discussed in Section 4.1, although we limit the size of the active goal set to one at any given time, we acknowledge the possibility of multiple goals. An opportunistic system maintains a single goal while considering others as well and reacts when advantageous circumstances arise, possibly switching to another goal.

Agre and Chapman [3], Brooks [32], and Webber and Badler [154] advocate opportunistic agents. As a motivating example, imagine a simulated elephant in search of a bag of peanuts. As part of its search pattern it is to walk to the end of a hallway. Its two goals are to walk to the end of the hallway and to find a bag of peanuts. At the system level the first goal is active, so the system takes the elephant down the hallway. Imagine that there is a bag of peanuts along the way. An opportunistic system would have a mechanism for interrupting itself before the elephant walked past the peanuts and could take advantage of the situation immediately while a non-opportunistic system would have to wait until the elephant achieved its current goal.

Parameterizable: Although one of the design goals is simplicity, control over the agent's locomotion style must not be sacrificed. To this end, the system designer should make as much of the system parameterizable as is reasonably possible. Varying the parameters should result in variable perceived agent behavior, making it possible to simulate state or personality attributes.

For example, imagine an elephant walking through a field. A system providing only a single locomotion style would be limited and uninteresting. The elephant's walk would always look the same. A parameterized system providing access to such parameters as locomotion speed and agent inertia would allow for much more interesting, varied behavior. A "rushed" elephant might walk faster and an elephant that had too much to drink at the local bar might stagger as a result of its lowered inertia.

Purposeful: According to Brooks [32], a creature should do something in the world; it should have some purpose in being. An agent should have a goal. The goal may be survival-oriented with no specific desired location as in foraging, or it may be more concrete as in walking to

a specific location, agent, or object. In either case, the agent's locomotion is motivated; it does not wander aimlessly.

Realistic: An agent must exhibit correct observed behavior, the notion of *perceived realism* or *correctness*. It should appear to do what a real human (or other animal) would do in a similar situation. This requirement applies to the gamut of behaviors ranging from the blatant, such as interobject penetration, to the subtle, such as foot placement. It is obvious to any observer that something is wrong when an agent walks through a wall. However, when the physical simulation of a human's walk is incorrect, even by a small amount, the resulting animation will look "wrong" to an observer who has spent his or her entire life passively noticing how humans walk. "Because people are attuned to the subtleties of human appearance and motion, they are very demanding of graphical representations of human figures" [75]. Both of these situations can be equally damaging to the final animation.

With the goal of achieving realism, we enforce the policy of limiting an agent's perception (Section 8.1) and we introduce the notion of anticipation (Section 8.2) throughout the system. **Limited Perception** prevents an agent, whose sensors are potentially omniscient in a simulated environment, from reacting to stimuli of which it should not be aware. Attraction to an object the agent can "see" is reasonable whereas attraction to an object that is hidden behind a building is not. **Anticipation** extends the system from the traditional approach of reacting to the world at the moment it is sensed to the novel approach of reacting to the state in which the world is predicted to be at some point in the future.

Robust: Nearly all modern systems include robustness as a system requirement. A robust system is one in which an agent's behavior is maintained despite minor changes to the environment. Furthermore, an agent's behavior must not collapse even when it is moved to a completely new environment. A properly designed, properly built system should allow for a wide variety of interesting environments.

We were careful to consider all of these requirements in designing our locomotion control architecture. We are now ready to present the architecture itself and its design methodology.

Chapter 6

Design Methodology

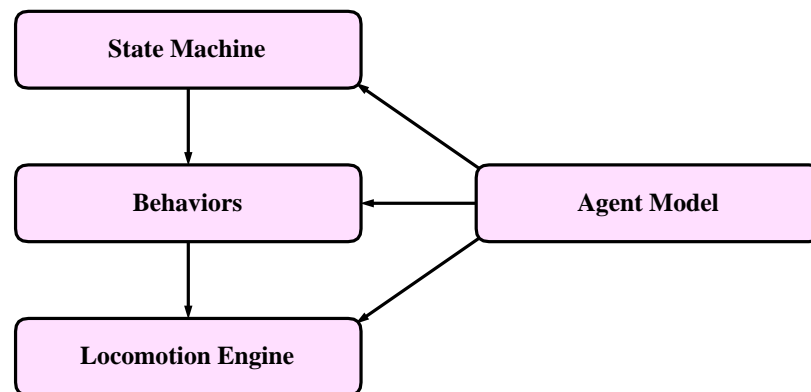


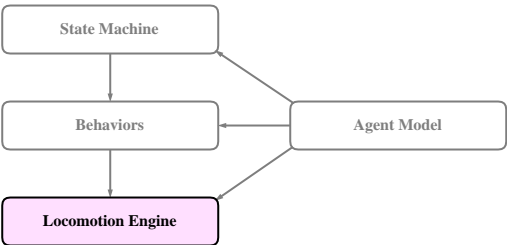
Figure 6.1: The Architecture

This chapter presents a formal description of the architectural design methodology. The goal is to build a behavior-based locomotion control system that adheres to the system requirements laid out in Chapter 5. The choice of behavioral control has the advantage of speed, addressing the system's efficiency requirement. Figure 6.1 illustrates the architecture. The following sections present design methodologies for the locomotion engine, the behaviors, the state machine, and the agent model.

At the lowest level, the **locomotion engine** acts as a liaison between the system and the environment, controlling agent locomotion directly based on active **behaviors** and insulating the user from system details. At the highest level, the **state machine** is capable of a discrete decision process and is responsible for scheduling and controlling the set of behaviors, thereby controlling

agent locomotion indirectly. Adding an **agent model** to the architecture to contribute to the state-machine’s configuration and the behaviors’ and locomotion engine’s parameterization allows for a locomotion style reflecting the agent’s physical state and personality.

6.1 Creating the Locomotion Engine



Effectively an embodiment of a sense-control-act loop, the locomotion engine is responsible for controlling agent locomotion based on the agent’s set of active behaviors. In this context, we model the abstract notion of a **behavior** as a function that maps a state of the world to the stress of being in that state (Figure 6.2). More specifically, we refer to this as an **instantaneous behavior** as described in Section 6.3.1.



Figure 6.2: The Behavior Function Template

We define an **operation** on the world to be a mapping from the current world state to a new world state where a particular action has occurred. In the case of legged locomotion, the agent takes a discrete step. In the case where action is continuous such as a flying bird or a rolling robot, action is discretized, usually by choosing a small, fixed time increment. The agent rolls or flies for this amount of time according to a chosen velocity vector. In either case we must choose a small number of operations (corresponding to actions) which we will consider for the agent during the next time increment. The locomotion engine evaluates each of these operations during its **sense** phase. During its **control** phase it selects the one corresponding to the least stressful new world state and applies the selected operation during its **action** phase.

The following is a formal description of the locomotion engine's central S-C-A loop as it applies to a single agent. Given the current state of the world, W_t , a set, \mathcal{O} , of m operations on the world, and a set, \mathcal{B} , of n behavior functions associated with an agent in the world, we wish to calculate the set, \mathcal{F} (for future), of possible next states of the world and the associated stresses, \mathcal{S} and \mathcal{T} .

World State at Time t	W_t
Operations	$\mathcal{O} : \{o_1, \dots, o_m\}$
Behavior Functions	$\mathcal{B} : \{b_1, \dots, b_n\}$
Next World States at Time $t + 1$	$\mathcal{F} : \{f_1, \dots, f_m\}$
Stresses	$\mathcal{S} : \{s_{11}, \dots, s_{mn}\}$
Total Stresses	$\mathcal{T} : \{t_1, \dots, t_m\}$

In each of the following, $i \in [1, m]$ and $j \in [1, n]$. First, we calculate the set of next world states by applying the set of operations to the current world state ($\mathcal{F} = \mathcal{O} \times W_t$):

$$f_i = o_i(W_t) \quad (6.1)$$

Next, we calculate the set of stresses by applying the set of behavior functions to the set of next world states ($\mathcal{S} = \mathcal{B} \times \mathcal{F}$):

$$s_{ij} = b_j(f_i) \quad (6.2)$$

Finally, we calculate the set of total stresses for each next world state by summing the subset of stresses associated with that state:

$$t_i = \sum_{j=1}^n s_{ij} \quad (6.3)$$

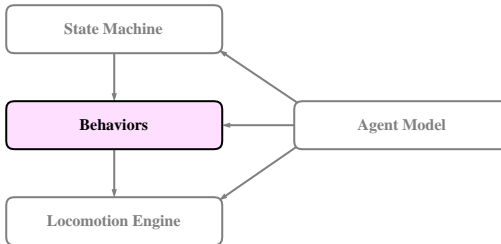
Now we choose a next world state of minimal total stress and the corresponding operation:

$$W_{t+1} = o_i(W_t) \quad \text{where} \quad o_i \in \left\{ o \mid \sum_{j=1}^n b_j(o(W_t)) \equiv \min(t_i) \right\} \quad (6.4)$$

To achieve the chosen next world state, apply the operation to the current world state.

The locomotion engine's **sense phase** is simply the application of the behavior functions. The remaining calculations including choosing which operation to apply make up the **control phase**, and the application of the chosen operation occurs as the **action phase**.

6.2 Choosing the Behaviors



The most important step in constructing this type of system is selecting a set of behavioral competences. This set determines the agent's locomotion repertoire and contributes to design decisions for the remainder of the system. We now address the issue of how to choose an appropriate set of behaviors, including conditions for selection and properties the behaviors should enjoy. Much of this discussion was motivated by Matarić [105].

6.2.1 Selection Criteria

We adopt Matarić's selection criteria:

We postulate that, for each domain, a set of behaviors can be found that are *basic* in that they are required for generating other behaviors, as well as being a minimal set the agent needs to reach its goal repertoire.

Essentially we choose a set of behaviors which form a basis, in the mathematical sense, for the universe of agent responses. Each behavior must be necessary and the complete set must be sufficient for the task at hand.

Necessity: Each behavior we choose for a particular domain must be necessary for that domain.

There must exist a goal that cannot be achieved by any of the other basic behaviors or their combinations. A basic behavior is one that cannot be reduced to a combination of other behaviors.

Sufficiency: The entire set of behaviors we choose for a particular domain must be sufficient for that domain. Every desired response must be expressible through some combination of basic behaviors, *i.e.*, no other behaviors are necessary.

6.2.2 Additional Properties

In addition to necessity and sufficiency, we propose the following additional properties each behavior should possess:

Locality: A behavior must obey the principle of **limited perception** discussed in Section 8.1. Simply put, a simulated agent must not have access to information that would be unavailable to a real agent in an identical situation.

Robustness: Under reasonable conditions, a behavior should induce intended results regardless of changes made to the environment. An agent moved from one environment to another should exhibit similar observed behavior.

Scalability: The behavior should be robust with respect to environment size.

Simplicity: The implementation should be as simple as possible while still accurately simulating the behavior. Complex behaviors are more difficult to write, debug, and apply.

6.3 Partitioning the Behaviors

As discussed in Section 3.3, three approaches to building control systems prevail: traditional planner-based approaches, purely reactive or behavioral approaches, and combinations of the two. In some cases, a state-based decision process replaces or is combined with a planner. We advocate a combined, layered approach allowing for both reactive and state-based control.

We partition the set of behaviors into two subsets, referring to behaviors that do not require stored information, *i.e.*, memoryless behaviors, as **instantaneous behaviors** (this is consistent with Gat's notion of reactive behaviors that continuously react to the environment [61]), and to those that do require memory as **memory-reliant behaviors** (Figure 6.3). These two types of behaviors require two different implementations, both provided by the architecture. We implement instantaneous behaviors through the combination of the locomotion engine and behavior layers,

and memory-reliant behaviors through the state-machine layer. To determine in which way the behavior is more appropriately implemented, the system designer must simply decide whether or not the behavior requires memory.

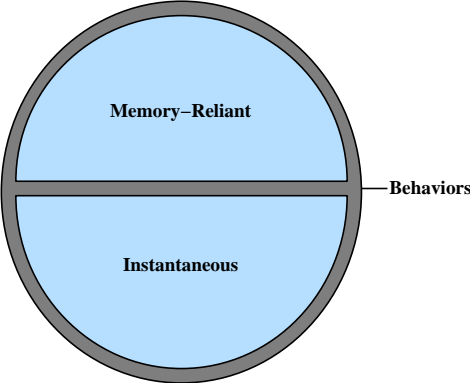


Figure 6.3: Partitioning the Set of Behaviors

6.3.1 Instantaneous Behaviors

For each instantaneous behavior the system designer must provide a behavior function which maps the instantaneous (current) world state to a measure of stress (Figure 6.2). The locomotion engine applies these functions during its sense phase.

We now partition the set of instantaneous behaviors into two subsets: level 0 and level 1 behaviors (Figure 6.4).

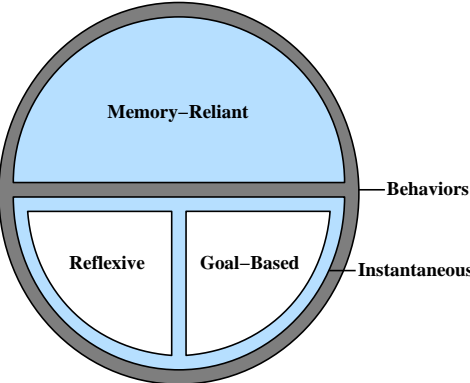


Figure 6.4: Partitioning the Set of Instantaneous Behaviors

level 0: Reflexive and Physics-Based Behaviors

Level 0 behaviors are named after Brooks' subsumption architecture's level 0 [32]. They are the set of instinctive reflexes which guarantee "survival," generally through collision avoidance [101], exhibiting a fixed behavioral pattern in response to stimuli [7, 108]. Level 0 behaviors are always active while the agent locomotes, but can be inhibited or overridden entirely by higher levels (the state machine or the agent model).

We also add "physics-based" behaviors to this set. Physics-based behaviors make up the subset that implement the agent's physical properties (such as inertia) and affect the perceived realism of the agent's locomotion. We add them at this level because they too are always active and are often affected by higher levels.

level 1: Goal-Based Behaviors

Level 1 behaviors are named after Brooks' subsumption architecture's level 1. They are the set of goal-based behaviors, generally variations of attraction, that allow the agent to go places. As previously discussed in Section 4.1, our novel approach to goal-based behaviors is to allow only one to be active at any time. An agent who is to walk to both the bank and the post office, for example, must choose between the two goals at any given time rather than combine or average them. We relegate goal arbitration and opportunistic responses to the state-machine level.

6.3.2 Memory-Reliant Behaviors

Memory-reliant behaviors require mechanisms for storing local information and for maintaining state. Therefore, we choose the state-machine paradigm as a natural way to implement each behavior's discrete decision processes. The *chasing* behavior is a good example and is described in more detail in Section 7.6.3. Chasing alternates between two states: an attraction to the target when it is visible, and an attraction to the target's last known location when it is occluded. The location-attraction state requires remembering where the target was last seen as well as its last observed velocity.

For each memory-reliant behavior the designer must provide: a set of preconditions (applicability conditions), a set of interruption conditions, a set of termination conditions, and a **behavior state machine** (BSM) for that behavior (Figure 6.5).

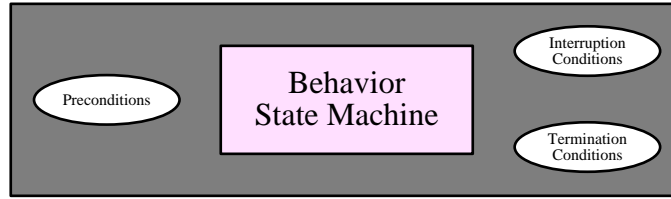


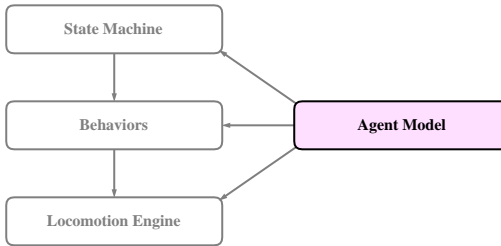
Figure 6.5: The Memory-Reliant Behavior Template

Before the locomotion control system (LCS) executes a BSM, its preconditions must be achieved. When this set is empty, it is achieved vacuously. The sole precondition for chasing in our system is that the target must be visible to the agent. Otherwise, the agent would not know where to begin to search for the target and another behavior, such as path-following, might be more appropriate to the situation.

During execution, if at any time either the set of interruption conditions or the set of termination conditions is achieved, LCS halts the BSM. In the former case, LCS chooses and executes a new BSM based on preconditions. In the latter case, LCS “cleans up after itself” and exits. Interruption and termination conditions are not required, but add to the system’s flexibility. They are a set of arbitrarily complex expressions passed to the system as an optional parameter. An example of an interruption condition is a situation in which the agent’s priorities have changed based on newly acquired information. A condition arises where it is more appropriate for the agent to change its current behavior, but not so drastic a change that termination is necessary. A seeker in the game of hide and seek might be chasing one agent only to discover that another agent is suddenly a better target (maybe it is closer). If the same seeker’s target were to be observed arriving at the safe “home” location, then, although chasing would still be possible, a termination condition would halt the unnecessary chasing and allow the calling user or system to make a new plan.

The BSM schedules and controls the set of level 1 goal-based behaviors to simulate the memory-reliant behavior and simulates memory in the form of local variables. All the information provided here will be used to build a single state-machine that combines these behaviors, and activates and deactivates each of the instantaneous behaviors.

6.4 Creating the Agent Model



A behavioral control system composed of behaviors and state machines is sufficient for generating locomotion. However, this locomotion is unvaried and hence uninteresting. Mataric's groups of robots use identical software, but they behave differently due to their slightly varied physical properties [104]. In the case of identical physical or simulated agents, something else is needed to differentiate the agents. Our aim is to design a system that includes an agent model, including state and personality attributes over which the user has control.

Without the agent model the agents would have the undesirable tendency to look personality-free, approaching each challenge without variation. With the addition of an agent model the architecture allows for the effects of agent state and personality. Each part of the architecture uses the agent model for parameterization. In this way we are able to encode a visual impression of personality as advocated by Badler *et al.* [15] and Perlin [126, 127].

Figure 6.6 illustrates the agent model. The system designer is responsible for choosing a set of system parameters and a set of model attributes, and for establishing a correspondence between the two sets. The user will be required to choose the degrees to which the attributes affect the system. Finally, the system itself calculates the parameter values, based on the established correspondence and chosen degrees, which are then used to adjust various aspects of the agent's locomotion.

6.4.1 System Parameters

The first step in designing the agent model is to choose a set, \mathcal{P} , of n system parameters which, when adjusted, have an observable effect on the agent's locomotion style. This set might include parameters that affect the locomotion's physics such as **speed** or **inertia**, or parameters that affect "cognitive" choices made by the state machine. It is important to restrict the set to parameters that might be affected by the agent's state or personality. Environmental parameters such as **gravity**,

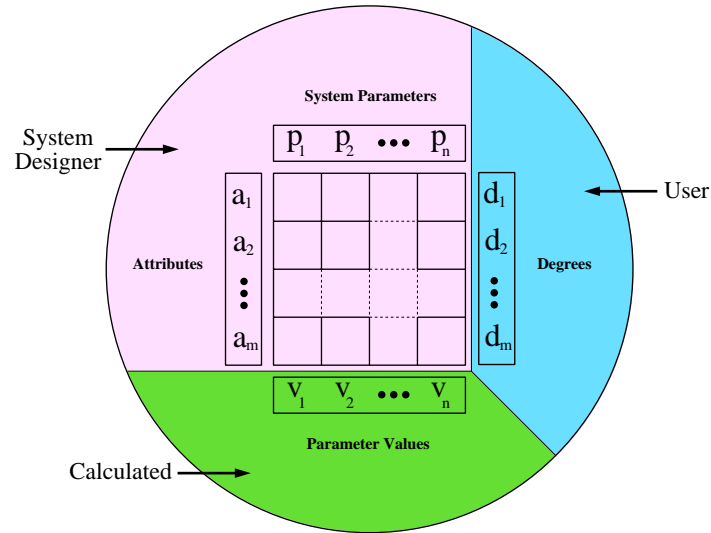


Figure 6.6: The Agent Model

while potentially useful, are inappropriate here.

6.4.2 Attributes

The next step is to choose a set, \mathcal{A} , of m state and personality attributes. Examples include state attributes such as **fatigue** or **intoxication**, and personality attributes such as **curiosity** or **caution**. This set may vary significantly from one domain to another or even from one implementation to another, so we leave the selection process as a design decision. The one piece of advice we offer is to keep it simple. It is unnecessary to keep the set small so long as the user limits his or her choice of *active* attributes to a reasonably small number.

6.4.3 Establishing the Correspondence

Given \mathcal{P} and \mathcal{A} , the system designer's final responsibility is to establish the correspondence between them. Since each attribute affects each parameter in a different way, this involves filling out the matrix from Figure 6.6 as illustrated in Figure 6.7. Each matrix entry contains all the information necessary to produce a recommended parameter value, given the user-defined set, \mathcal{D} , of degrees (Section 6.4.4). We describe the three parts of each matrix entry in the following sections.

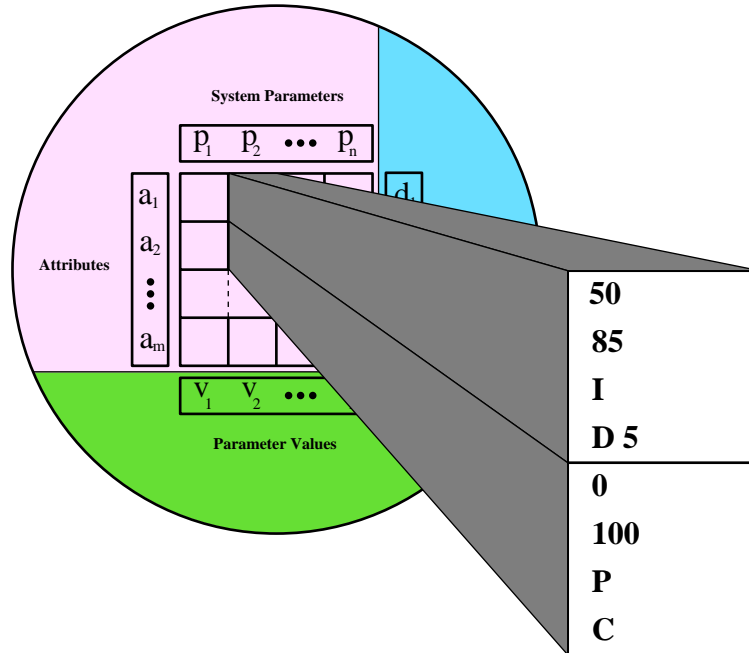


Figure 6.7: Sample Agent Model Entries

Range of Values

For the sake of simplicity and without loss of generality, the agent model treats all system parameter values as being in the range $[0, 100]$ with 50 often being the default. Although the system maps this range, usually linearly¹, to the actual implemented range, we keep this mapping transparent at the agent-model level.

The range of values, made up by the first two numbers in each matrix entry, specifies the subrange of valid parameter values for the entry. The exact value in this subrange is based on the attribute's degree. Consider the speed-parameter/rushed-attribute entry as an example. A reasonable subrange for this combination might be $[55, 90]$ where a slightly rushed agent might travel at a rate of 60 while an agent in a "big" rush might move at a rate of 85 or even 90.

Proportional or Inversely Proportional

As another example, consider the speed-parameter/fatigued-attribute entry. A reasonable subrange for this combination might be $[10, 40]$ where a slightly fatigued agent might travel at a rate of

¹The linear mapping has proven to be sufficiently general while still being intuitive. It may be interesting, however, to experiment with non-linear mappings in the future.

35 while an extremely fatigued agent might move at a rate as slow as 10. This example differs from the *rushed* example in that as fatigue increases the value decreases. In this case we call the relationship “inversely proportional” and specify it as such with an “I” (as opposed to a “P”) as the third datum in the appropriate entry.

Combinatorial or Dominant

The final part of each entry specifies how to arbitrate among it and all the other recommended parameter values in the corresponding parameter’s column. An attribute may have one of three types of effects on a parameter. It may have no effect (N), a combinatorial effect (C), or a dominant effect (D). If all the entries in one column are marked as having no effect, the parameter is assigned the predefined default value, usually 50. Otherwise, if all the contributing values are combinatorial, they are simply averaged, their relative contributions already having been scaled according to their degrees. When one or more entries are dominant, the value of the one with the highest priority (the value next to the “D” in the figures) is taken and the other entries’ values are ignored.

6.4.4 Degrees

As with system parameters, for consistency, we choose to limit each degree to the range (0, 100]. This value represents the “degree” to which the agent is to exhibit the attribute. Note that this set does not include zero. We reserve zero to represent an inactive attribute where the system ignores the corresponding row of entries for parameter value calculations. The system presents the user with a set of attributes and requests a corresponding set of degrees, possibly through an interface including input boxes or slider bars. This is the extent to which the user is exposed to the system. He or she remains insulated from the implementation details.

6.4.5 Calculating the Results

The algorithm for calculating the results, \mathcal{V} , is as follows. First, calculate each recommended value by mapping the row’s degree over the specified range, either proportionally or inversely proportionally.

Given the low and high recommended values and the proportional flag for each entry and the set of degrees:

Set of Low Recommended Values	$\mathcal{L} : \{l_{11}, \dots, l_{mn}\}$
Set of High Recommended Values	$\mathcal{H} : \{h_{11}, \dots, h_{mn}\}$
Set of Proportional Flags	$\mathcal{P} : \{p_{11}, \dots, p_{mn}\}$
Set of Degrees	$\mathcal{D} : \{d_1, \dots, d_n\}$

We wish to calculate the set of recommended parameter values:

$$\text{Set of Recommended Parameter Values } \mathcal{R} : \{r_{11}, \dots, r_{mn}\}$$

In each of the following, $i \in [1, m]$ and $j \in [1, n]$. The algorithm for calculating r_{ij} is as follows. If $p_{ij} = \text{“P”}$, indicating a proportional correspondence, then:

$$r_{ij} = \frac{d_i - 1}{99} (h_{ij} - l_{ij}) + l_{ij} \quad (6.5)$$

Otherwise $p_{ij} = \text{“I”}$, indicating an inversely proportional correspondence:

$$r_{ij} = \frac{100 - d_i}{99} (h_{ij} - l_{ij}) + l_{ij} \quad (6.6)$$

Then, for each parameter, examine the contribution entries. If they are all “N”s (or are all of degree zero and thus inactive), then set the parameter value to its default value. If there is at least one active dominant attribute, set the parameter value to the recommended value of the dominant entry of highest priority. Otherwise, set the parameter value to the average of all the active combinatorial attributes.

As an example, see Figure 6.8. Row 3 is ignored in all three cases because its degree of zero indicates that it is inactive. r_{11} , r_{21} , and v_1 are calculated as follows:

$$r_{11} = \frac{100 - 70}{99} (85 - 50) + 50 = 61 \quad (6.7)$$

$$r_{21} = \frac{15 - 1}{99} (100 - 0) = 14 \quad (6.8)$$

$$v_1 = \frac{61 + 14}{2} = 37 \quad (6.9)$$

	p_1	p_2	p_3	
a_1	50 85 I C	30 90 P C	40 80 P D 2	70 d_1
a_2	0 100 P C	27 100 I D 3	25 99 P D 5	15 d_2
a_3	 N	 N	21 88 I D 7	0 d_3
	v_1	v_2	v_3	

Figure 6.8: Sample Parameter Value Calculations: The Initial Conditions

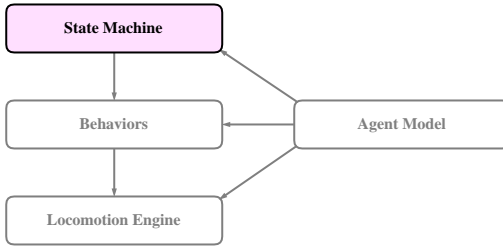
Figure 6.9 shows all the recommended parameter values. Since a_2 is the active dominant attribute of highest priority for parameters p_2 and p_3 , the parameter values are assigned the respective recommended parameter values from row 2. Note that activating attribute a_3 would only affect parameter value v_3 .

	p_1	p_2	p_3	
a_1	61 C	72 C	68 D 2	70 d_1
a_2	14 C	90 D 3	35 D 5	15 d_2
a_3	— N	— N	— D 7	0 d_3
	37	90	35	
	v_1	v_2	v_3	

Figure 6.9: Sample Parameter Value Calculations: The Results

6.5 Building the State Machine

In the previous sections we presented methods for creating a locomotion engine and an agent model. We also outlined methods for choosing a set of behaviors and for partitioning it into



three subsets: level 0 reflexive and physics-based behaviors, level 1 goal based behaviors, and memory-reliant behaviors. Now we explain how to build the state machine that will combine all the parts, implementing the selected set of behaviors and embodying the desired locomotion control system. Figure 6.10 illustrates the design. Because every system will differ, we omit implementation details, describing the design abstractly.

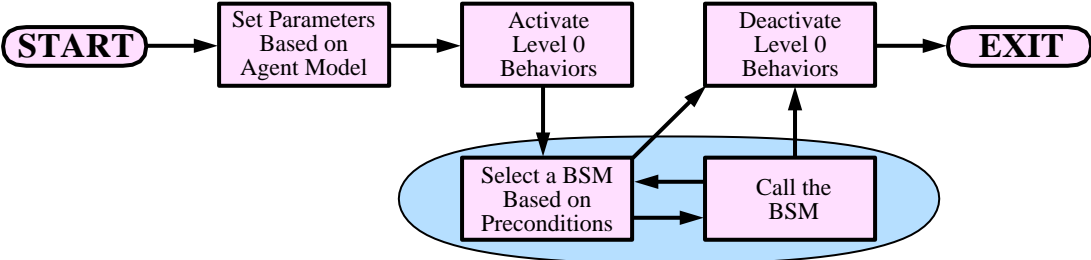


Figure 6.10: The Locomotion Control System Flow Diagram

As part of its initialization process, LCS first adjusts system parameters based on the agent model to give the resulting locomotion an appearance that reflects the selected state and personality attributes. For example, if the agent were intoxicated, both its speed and inertia would be decreased. The other parts of the system should include a way to adjust the necessary parameters based on the agent model. Also as part of system initialization, LCS activates the set of level 0 behaviors which will remain active for the duration of the agent’s locomotion (except when affected by a BSM).

Next, LCS enters its main loop (highlighted in the figure) where it selects and executes BSMs. The selection process is based on the sets of preconditions. Each BSM’s preconditions are evaluated in the current context in a predefined order. The first BSM whose preconditions are satisfied is executed. If none of the preconditions are satisfied, the state machine deactivates the level 0 behaviors and exits reporting its “failure” status. If at any time during the execution of a BSM its termination conditions are achieved, LCS halts its execution, deactivates the level 0 behaviors and exits reporting its “success” status. When a BSM terminates normally or when its interruption

conditions are achieved, LCS halts its execution if necessary and returns to the selection process.

Since there are cases where a level 1 behavior alone will achieve the goal, the set of level 1 behaviors must be made available during LCS's selection process. For consistency, sets of preconditions, interruption conditions and termination conditions, and a simple memoryless BSM must be provided for each one. These networks are then added to LCS's set of BSMs and executed when appropriate.

6.6 Extensibility

Each of the four architectural components of our system can be modified or extended as we will explain in this section. At the lowest level, the locomotion engine can be modified in two ways to support different locomotion styles or different gaits. The footstep-generation module can be modified by changing the shape of the region reflecting where the agent can be in the next simulation step (after applying an operation to the current world state, or after taking the current step). For example, a rolling robot may be able to move in any direction, thus requiring a circular region instead of the fan-shaped region used for walking. Also, the action module can be replaced with another action module that supports a different gait such as running, hopping, or skipping.

Integrating a new behavior into an existing system depends on the type of behavior being added: reflexive, goal-based, or memory-reliant. In each case, the required behavioral elements must be written, *i.e.*, the behavioral function or the behavioral state machine and its associated conditions. To add a new reflexive behavior to the system, it is sufficient to add it to the system state machine's list of level 0 behaviors. It will then get executed whenever the pre-existing level 0 behaviors are executed. To add a new goal-based behavior, it need only be made available for execution. The system state machine is responsible for scheduling this new behavior as it is for any level 1 behavior. The system designer should examine each existing behavioral state machine and decide if it should be modified to accommodate this new level 1 behavior. To add a new memory-reliant behavior to the system, its set of preconditions must be inserted into the list of BSM preconditions considered by the system state machine. During the BSM selection phase, the preconditions will be evaluated in the proper order.

Adding a new system parameter requires inserting parameterizable code at the appropriate

place in the system, and then integrating the new parameter into the agent model. The code can be part of the locomotion engine, the behaviors, the state machine, or any other part of the system so long as there is an accessible parameter to be modified by the agent model. Integration into the agent model requires the system designer to establish the correspondence between the existing attributes and the new system parameter. Adding a new attribute is simpler, only requiring the system designer to establish the correspondence between it and the existing set of system parameters.

6.7 Design Summary

To summarize, the design process consists of these steps:

1. Locomotion Engine

- Implement the S-C-A loop

2. Behaviors

- Choose the set of behaviors

- Partition the set of behaviors into instantaneous and memory-reliant subsets

- Partition the set of instantaneous behaviors into level 0 and level 1 subsets

- Write behavior functions and state machines

3. Agent Model

- Choose the set of system parameters

- Choose the set of attributes

- Establish the correspondence between them

- Provide a mechanism for user interaction

4. State Machine

- Build the state machine

In the chapters that follow we demonstrate the design process by designing and building a human locomotion control system and presenting the results.

Part III

A Concrete Example

Chapter 7

A Human Locomotion Control System

We demonstrate the design process by building a locomotion control system (LCS) for simulated human agents. LCS endows the agents with reflexive avoidance, goal-based attraction, chasing, and path-following behaviors and provides an agent model allowing manipulation of such parameters as agent speed and inertia. LCS adheres to the limited perception policy and exhibits anticipation at every level of the architecture (see Issues, Chapter 8).

7.1 Development Environment

Running on Silicon Graphics Iris (SGI) workstations, *Jack*^{®1} is a human modeling and simulation program under development at the Center for Human Modeling and Simulation at the University of Pennsylvania. We use *Jack*'s framework for general object locomotion. *Jack* features anthropometrically reasonable human models and general, real-time, visually realistic and dynamically sound locomotion [82, 83, 84, 128]. Additionally, *Jack* includes Becket's Behavioral Simulation System [18, 20] and PaT-Nets [13, 85], a facility for writing state machines which run in parallel with simulations. These features combine to produce realistic looking simulations.

¹*Jack* is a registered trademark of the University of Pennsylvania

7.2 Locomotion Engine

Becket's simulation architecture exhibits several features also found in the subsumption architecture [29], nouvelle AI [30], reactive planning [66], situated agents [3, 95], computational neuro-ethology [95], and behavioral control/behavioral animation [71, 129, 130, 157], namely:

- A bottom-up approach
- Motivated by the animal sciences
- Tightly coupled sensors and effectors
- Integrated behaviors which, if used in isolation, have observable, beneficial effects
- *Emergent behavior*

The Behavioral Simulation System (BSS), the motivation for our locomotion engine, provides general locomotion of objects in *Jack*; we use it to generate human locomotion. Central to BSS is a sense-control-act loop. In the *sense* phase simulated sensors sense the environment. This information determines where the agent should step in the *control* phase. The agent takes the chosen step in the *act* phase.

Behaviors determine how an agent acts. In BSS, a sensor (Section 7.3) and a control behavior (attraction or avoidance - Section 7.4) combine to form a behavior (Figure 7.17 - Section 7.5). A behavior maps a potential new foot position to the “stress” of stepping there. From among the possible choices, the control phase leads the agent to take the least stressful step.

A behavior activates when “bound” to an agent. While active, its output contributes to the stress calculations. Behaviors may be bound or unbound at any time during a simulation, and thus activated or deactivated. Locomotion is performed indirectly by binding behaviors to humans. BSS, constantly monitoring the environment, immediately initiates the appropriate locomotion.

7.2.1 The Simulation Sense-Control-Act Loop

Webber and Badler discuss **Sense-Control-Act** (S-C-A) loops in depth [154]. Each iteration of BSS's S-C-A loop includes perception, control, and action phases (Figure 7.1). Three stages compose the perception phase: (1) generation of possible footsteps, (2) polling of active behaviors,

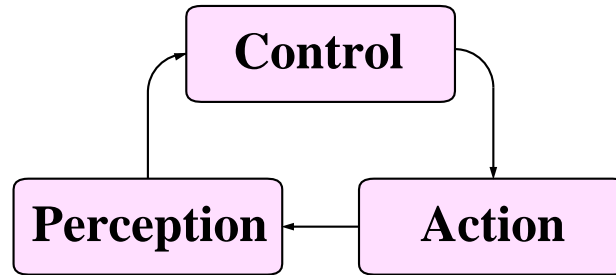


Figure 7.1: The Behavioral Simulation System's Central Simulation Loop

and (3) calculation of overall stress. The first stage determines a set of points to which the agent can step. Then, in the second stage, behavior function evaluation determines stress values for each point; stress is proportional to an agent's desire not to step at a given point. Finally, given the active behaviors, the sum of the stress values for each point determines the overall stress for that point.

The control phase selects the least stressful potential point and directs the action phase to execute the locomotion. Formally, Becket's BSS functions as the locomotion engine described in Section 6.1, so it is a good lowest level for our system. When simulating human locomotion, operations consist of footsteps. Each time through the S-C-A loop the agent takes one step.

7.2.2 Perception

Perception includes the generation of possible steps, the active behavior evaluation, and the calculation of overall stress.

Generation of Possible Steps

Robotics often employs a pure potential-field approach where the robot is viewed as a point-mass subject to attractive and repulsive forces [7, 8, 81, 147]. Unlike robots or more abstract objects which can move in any direction, human locomotion is generally limited to a small set of possible next footsteps. The first part of perception generates candidate foot positions. Three parameters determine the set of possible positions:

- Minimum Step Length (MIN)
- Maximum Step Length (MAX)

- Maximum Turning Angle (θ)

Given the agent's position and orientation, these parameters define a fan-shaped region in front of the agent (Figure 7.2). The agent can step anywhere within this region. The set of the agent's possible new states, \mathcal{P} , corresponds to the set of points to which the agent can step next. The set of points is the intersections of concentric arcs and uniformly spaced radii. The number of arcs and radii can be varied.

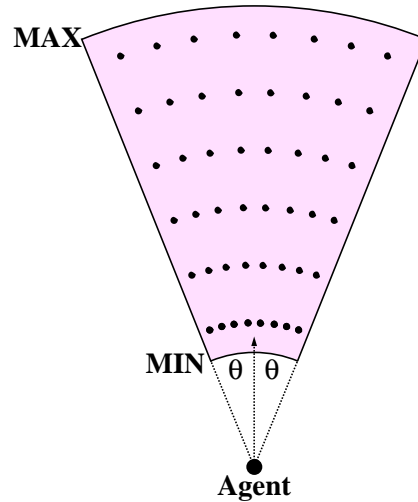


Figure 7.2: Calculating the Set of Possible Next Foot Positions

The agent's current location may also be added to this set. If this point is selected as the point to which to step, the agent does not move. Care must be taken when choosing this option to avoid situations where the agent may get stuck in a local minimum. When side-stepping and backing up are desired as additional gait options, the fan-shaped region can be modified to include regions behind and to the side of the agent. If a point in either of these regions is selected as the point to which to step, BSS selects side-stepping or backing up for the agent as appropriate.

Active Behavior Evaluation

For each state $p \in \mathcal{P}$ the system simulates p and then polls the behaviors. In this way, the system is anticipating the state of the agent one step in the future. Each behavior returns a stress value which is proportional to the undesirability of the state. The stress value is multiplied by a user-defined scalar and raised to a user-defined power giving the user control over the relative contributions

of the behaviors. Sometimes a behavior computes two stress values: one based on the agent's position and one based on its orientation. This requires two sets of user-defined constants.

Behaviors have access to the entire environmental database and are effectively omniscient. For the sake of realism, the designer should limit the amount or type of knowledge extracted. Section 8.1 addresses this issue of limited perception.

Calculation of Overall Stress

The overall stress of a particular state is the sum of the weighted stresses output by each active behavior where the weights are the user-defined scalars and powers.

7.2.3 Control and Action

After computing each state's overall stress, BSS selects the state with the lowest overall stress in the control step. In the action step, invocation of the locomotion system causes the agent to step to the appropriate point. The agent enters a new state and the S-C-A loop continues.

7.3 Basic Sensors

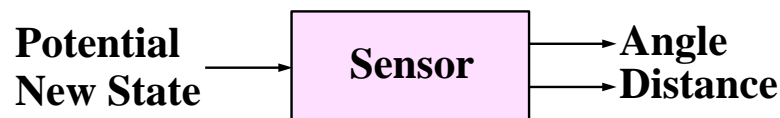


Figure 7.3: The Sensor Function Template

Sensors simulate the perceptual part of a behavior by sensing an element of the environment [18, 20, 70, 85, 113]. They are functions that return the distance and angle to this element from the agent in the potential new state (Figure 7.3). The following sections discuss object, location, line, and proximity sensors.

7.3.1 Object

An object sensor senses a single object in the environment and usually combines with an attract control behavior forming an attraction behavior. The sensing is global; there are no restrictions such as visibility limitations. As a result, care must be taken when using this sensor: the agent will

walk through walls or other objects to get to the object unless bound with the proper avoidances, and realism may be compromised by an attraction to an occluded object. Since an object sensor always senses the object's current location, following or chasing behaviors are possible.

Implementation

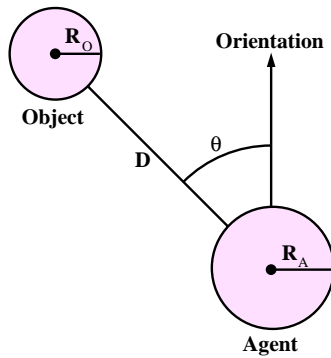


Figure 7.4: The Mathematics of an Object Sensor

In Figure 7.4, R_O and R_A are the radii of the object and agents' bounding cylinders. An object sensor outputs D and θ as indicated. D is the distance between the agent's and object's bounding cylinders and θ is the angle to the object relative to the agent's orientation.

Discussion

The research that led to our architecture required surmounting many obstacles along the way. Although many authors choose to omit these implementation details, we feel that it is our responsibility to present them in order to aid the reader who wishes to build our system. Therefore we present some of the pitfalls that befell us and how we overcame them.

When people write papers about these things a lot gets shoveled under the rug. The rug often looks like the Himalayas.

– Max Mintz –

Basing D on the distance between bounding cylinders speeds up calculations and is a good approximation in general, however, it introduces severe asymmetries for oblong objects. Figure 7.5

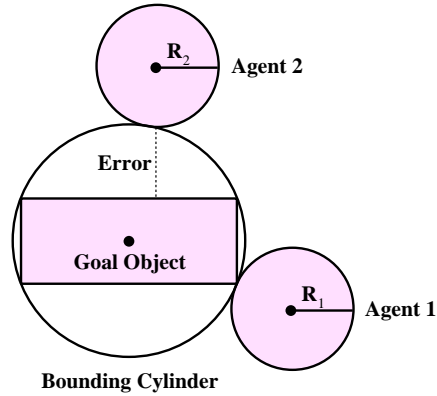


Figure 7.5: Asymmetry Caused by the Use of Bounding Cylinders

shows that, depending on its approach direction, an agent will stop at different distances from the object. This error distance is clearly visible when the goal is large. An agent walking to a car will stop several feet away from the car if it approaches from the side.

Our solution is to associate both bounding cylinders and bounding boxes with each object, both centered at the object's centroid. Distance is approximated based on the bounding shape that better fits the object. This solution increases the system's accuracy far more than it decreases its speed.

7.3.2 Location

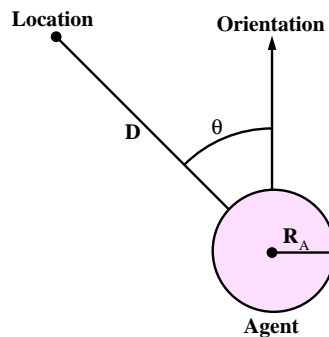


Figure 7.6: The Mathematics of a Location Sensor

A location sensor is nearly identical to an object sensor, usually forming part of an attraction behavior (Figure 7.6). It differs by sensing a fixed point in space, not corresponding to an object.

Since a location is a point, a location sensor does not suffer from the same problems as an

object sensor. However, the original implementation did not allow precise control over the agent's final position. The agent is only guaranteed to arrive within about half a meter (the radius of a human's bounding cylinder) of the goal location.

We found it useful to improve the location sensor's accuracy by basing the sensor's output on the distance between the forward foot and the goal location. This allows more precise control over the agent's final location with little reduction in the system's speed. The agent ends the simulation standing at the goal instead of near it.

7.3.3 Line

A line sensor senses a single line segment which must be associated with an object. If the object moves, the line segment moves, maintaining its position relative to the object. Line sensors typically sense long, narrow objects such as walls, sidewalks, or streets. When combined with an attract control behavior, they can keep an agent on a sidewalk, a path, or on the right side of a hallway; when combined with an avoid control behavior, they can prevent an agent from walking into a busy street or from walking through a wall.

Implementation

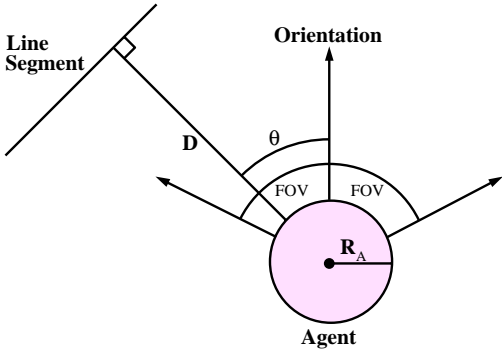


Figure 7.7: The Mathematics of a Line Sensor

In Figure 7.7, D is the distance from the agent's bounding cylinder to the closest point on the line segment. If the angle, θ , to this point is within FOV (the sensor's field-of-view) of the agent's orientation, the sensor outputs D and θ . Otherwise, its output is zero, indicating that the segment is not in the specified field-of-view.

Discussion

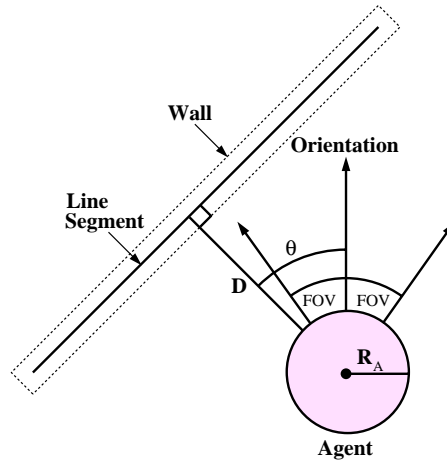


Figure 7.8: An Undetected Wall

When an agent approached a wall at an angle greater than FOV from the wall's normal vector, the wall remained undetected and the agent walked through it (Figure 7.8). Additionally, because of discrete sensing, an agent's bounding cylinder was sometimes on one side of a segment at the end of one step and on the other side at the end of the next step; the agent walked through the segment.

Our first solution to the first problem was to restrict FOV to $\frac{\pi}{2}$ or greater. While this prevented walls from going undetected it also prevented an agent from walking parallel to a wall without getting pushed away. We discovered that a better solution was to develop a collision prevention behavior that output infinite stress when the agent penetrated another object. When combined with a line sensor, this solved the second problem as well.

7.3.4 Proximity

A proximity sensor senses a predetermined set of objects. This sensing is local; the sensor is only sensitive to objects which intersect a fan-shaped region roughly corresponding to the agent's field-of-view (Figure 7.9).

An "intrusion" distance, α , is calculated for each object sensed by the proximity sensor whose bounding cylinder intersects its region of sensitivity. Shown as α_1 and α_2 in Figure 7.9, an object's intrusion distance is the distance from the outer radius of the region to the point on the object's bounding cylinder nearest the agent. A simulated object is calculated as the weighted average of

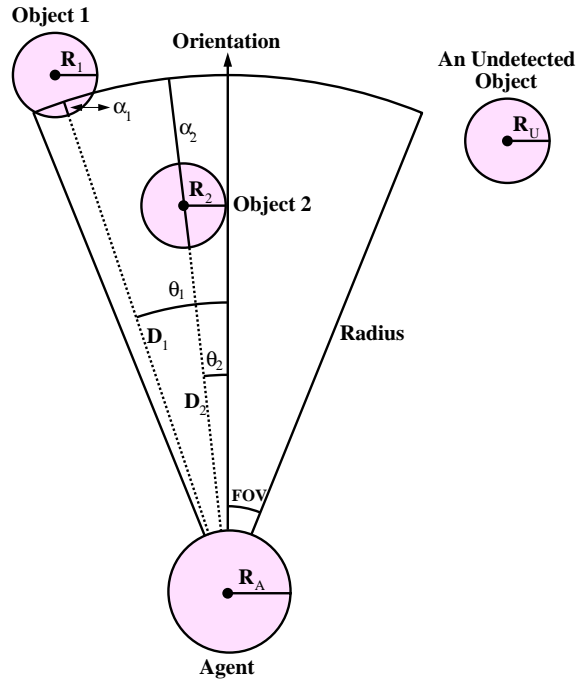


Figure 7.9: The Mathematics of a Proximity Sensor

all the detected objects where the contribution of object i is proportional to α_i (Figure 7.10); an agent is more sensitive to closer objects. The sensor outputs are based on this simulated object and are identical to those of an object sensor sensing the simulated object.

Proximity sensors are used almost exclusively to create avoid behaviors due to the local nature of avoidance. Section 7.4.2 contains an analysis of proximity sensors combined with avoid control behaviors and Section 8.2.4 contains an analysis of the effects of varying the proximity sensor's region of sensitivity including pictorial examples.

7.4 Control Behaviors

A **control behavior**, such as *attract* or *avoid*, is a function that maps the angle and distance to an object or location to a stress value (Figure 7.11).

7.4.1 Attract

An attract control behavior combines with a sensor, usually an object or location sensor, to form an attraction behavior that draws an agent toward a goal. If the goal moves, the point of attraction

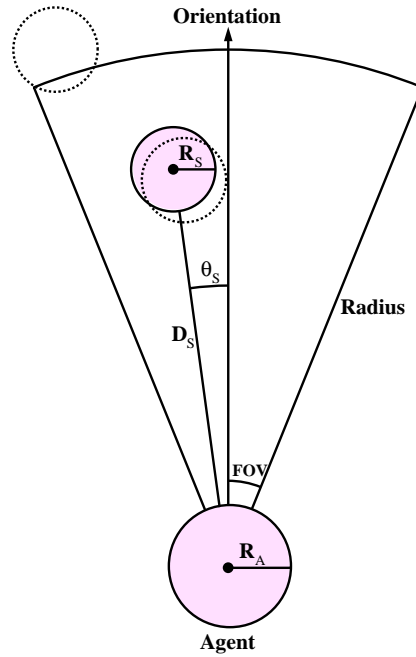


Figure 7.10: The Proximity Sensor's Simulated Object



Figure 7.11: The Control Behavior Function Template

moves appropriately since the distance and angle output by the sensor will change. An attract control behavior's output (stress value) is high when the angle or distance is high. As the agent nears the goal, stress decreases.

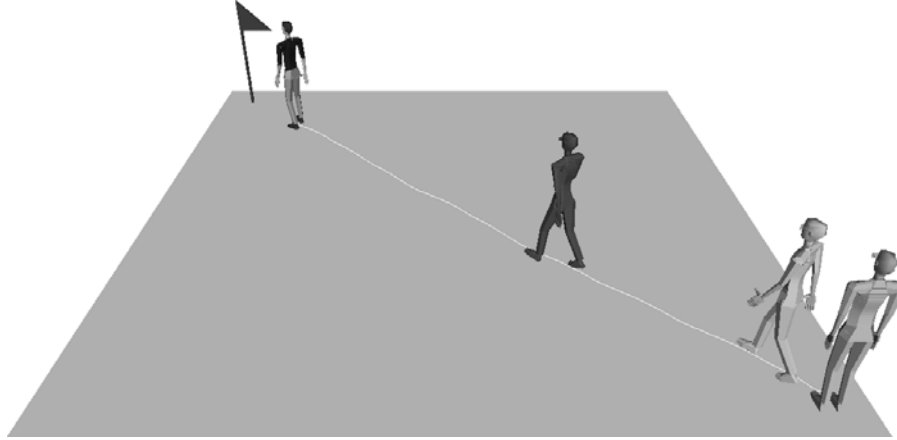


Figure 7.12: An object sensor and an attract control behavior combine to form an attraction behavior which draws the agent to the goal

Figure 7.12 shows an agent walking from the lower right corner of the image to the goal in the upper left. An object sensor and an attract control behavior constitute the attraction behavior which guides the agent. In the absence of other behaviors, the least stressful step always takes the agent closer to the goal.

Implementation

A transfer function converts an angle or a distance into a stress value. Equations 7.2 and 7.4 are the transfer functions for angle and distance respectively.

$$\theta' = \theta - \theta_{min} \tag{7.1}$$

$$Stress_{\theta} = S_{\theta} \left((\theta + 1)^{T_{\theta}} - 1 \right) \tag{7.2}$$

$$D' = D - D_{min} \tag{7.3}$$

$$Stress_D = S_D \left((D' + 1)^{T_D} - 1 \right) \quad (7.4)$$

$S_\theta, T_\theta, S_D, T_D, D_{min}$, and θ_{min} are user-defined constants. $Stress_D$ is based on the distance from the agent's current location to a distance D_{min} away from the goal location or the goal object's bounding cylinder. Thus $Stress_D$ will drop to zero when the agent is within a tolerable distance (D_{min}) and facing within a tolerable angle (θ_{min}) of the goal.

S_θ, T_θ, S_D , and T_D affect the individual stress's contribution to the overall stress of an agent's potential new state. The agent attempts to arrive within D_{min} of the goal, oriented within θ_{min} of the goal.

If the agent is not within D_{min} of the goal or is not oriented within θ_{min} of the goal ($D' > 0$ or $\theta' > 0$), the control behavior's output is the sum of the squares of the distance and angle stress values, otherwise, it is zero.

$$Stress = \begin{cases} Stress_d^2 + Stress_\theta^2 & \text{if } D' > 0 \text{ or } \theta' > 0 \\ 0 & \text{otherwise} \end{cases} \quad (7.5)$$

To summarize, the greater the distance to the object and the greater the angle between the agent's orientation and the goal, the greater the stress.

Discussion

To control the agent's final position, particularly when attracted to a point, a low arrival threshold value is necessary. Unfortunately, using a low arrival threshold may cause the agent to step past the goal and then circle back trying to achieve the goal, due to the discrete nature of BSS in choosing possible next foot locations. In Figure 7.13, the emphasized point is chosen for the next step. If the arrival threshold is less than the distance between the agent in the new state and the goal, the agent has not arrived and will continue walking past the goal.

Ensuring that D_{min} is at least half the maximum of the distance between concentric arcs and adjacent radii prevents the agent from passing the goal. Alternatively, adding the goal location to the set of possible next foot locations when it is within the fan-shaped region where the agent is

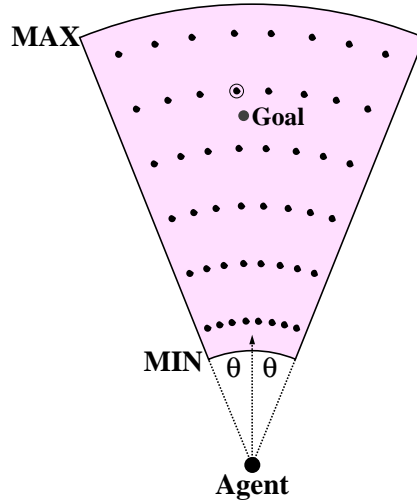


Figure 7.13: Walking Past the Goal

able to step is a better solution. It solves the problem without sacrificing control over the agent's final location.

Using the condition $D' > 0$ or $\theta' > 0$ to determine whether the agent has arrived can cause the agent to spiral in towards the goal, especially when the agent's maximum turning angle is low due to momentum, for example. When the agent is very close, but faces away from the goal, it will take a step that does not bring it significantly closer. It may even step away from the goal as shown in Figure 7.14. The agent will orbit the goal chaotically until it approaches the goal at a low angle.

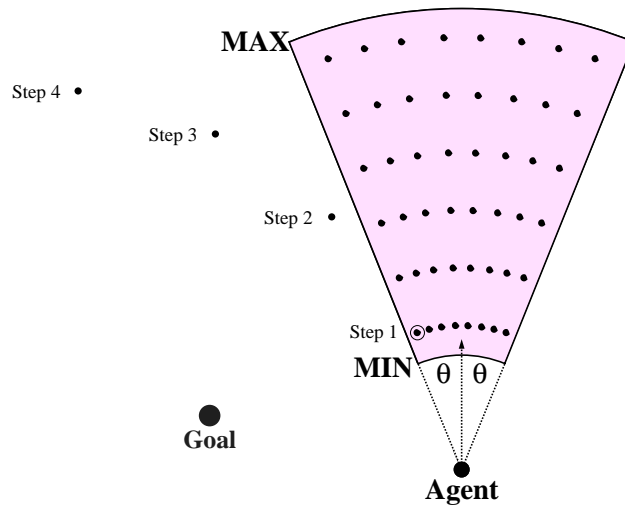


Figure 7.14: An Orbital Approach

Changing the condition to $D' > 0$ and choosing D_{min} sufficiently large prevents the agent from orbiting the goal, however, this solution reduces control over the agent's final position and orientation. A better solution interrupts the S-C-A formalism, performing a higher-level, more appropriate response. PaT-Nets are used to monitor the attraction. If the angle to the goal increases beyond a predefined threshold (when the agent is near the goal), the PaT-Net suspends the agent's controller, turns the agent toward the goal, and then resumes the agent's control system. We note here that this problem only occurs when the maximum turning angle per step is relatively low, significantly less than 90 degrees. More flexible animals or robots do not suffer from this problem.

7.4.2 Avoid

An avoid control behavior combines with a sensor, usually a line or proximity sensor, to form an avoidance behavior. Avoidance behaviors prevent agents from colliding with other agents, objects, or walls.

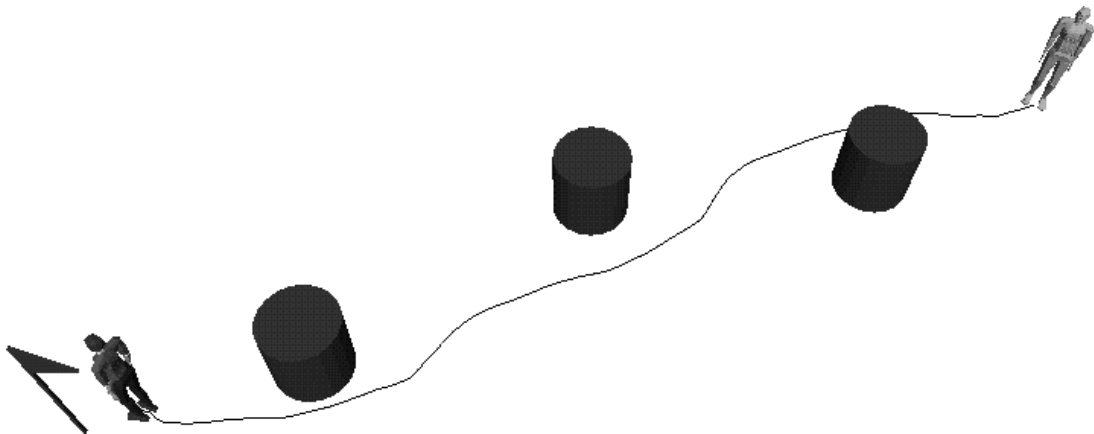


Figure 7.15: A proximity sensor and an avoid control behavior combine to form an avoidance behavior. While an attraction behavior draws the agent to the goal, the avoidance behavior steers it around obstacles.

Figure 7.15 shows an agent walking to the goal on the left of the image. An object sensor and an attract control behavior constitute the attraction behavior which guides the agent to the goal. A proximity sensor and an avoid control behavior constitute the avoidance behavior which guides the agent around the obstacles.

Implementation

The implementation of an avoid control behavior is similar to that of an attract control behavior. Equations 7.7 and 7.9 are the transfer functions for angle and distance respectively.

$$\theta' = \text{signum}(\theta) (\pi - |\theta|) \quad (7.6)$$

Where:

$$\text{signum}(x) = \begin{cases} \frac{x}{|x|} & \text{if } x \neq 0 \\ 1 & \text{if } x = 0 \end{cases}$$

$$\text{Stress}_\theta = S_\theta \left((\theta' + 1)^{T_\theta} - 1 \right) \quad (7.7)$$

$$D' = D_{max} - D + R_A \quad (7.8)$$

$$\text{Stress}_D = S_D \left((D' + 1)^{T_D} - (1 + R_A) \right) \quad (7.9)$$

R_A is the agent's bounding cylinder's radius. S_θ , T_θ , S_D , and T_D affect the individual stress' contribution to the overall stress of a state in the same way as for attract. Stress_D is based on how far an object extends into the avoidance radius, D_{max} ; the more it extends, the greater the stress. Stress_θ is greatest when the agent is facing the obstacle ($\theta = 0$).

Discussion

Adjusting the FOV angle changes the agent's reaction to peripheral activity. When FOV is large, the agent will react to objects it passes. Combining avoidance with such a proximity sensor can result in unrealistic behavior. The agent will attempt to avoid objects already passed or objects that it would have avoided had it continued to walk in a straight line. In coordinated behavior such as group walking, an agent would not want to avoid other agents to its side. Figure 7.16 shows the problem with using a small FOV. The two agents, oriented as shown by the arrows, are unaware of each other and will collide if they both choose to step in the shaded region. Predictive sensors and behaviors that anticipate, described in Section 8.2, do not suffer from these problems.

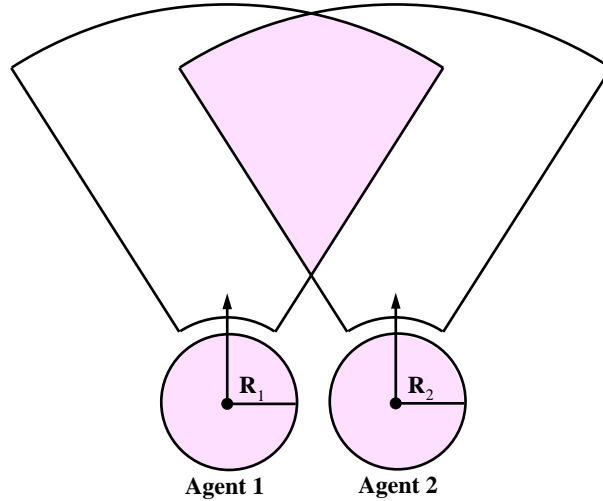


Figure 7.16: Two Agents Colliding

7.5 Behaviors

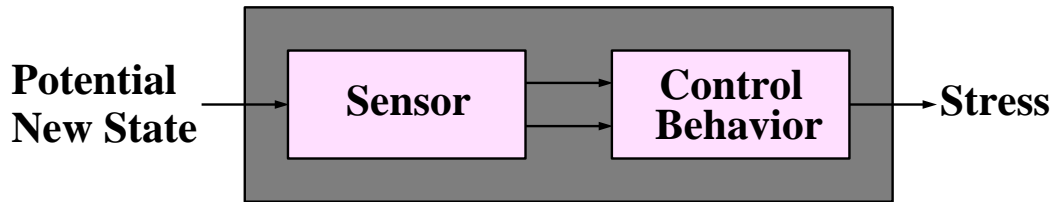


Figure 7.17: The Behavior Function Template

Behaviors influence an agent's actions in specified circumstances or reactions to specified situations (Figure 7.17). In BSS, a behavior combines a sensor and a control behavior. The resulting function maps agent position and orientation in the environment to a stress value; lower values represent more desirable states.

When designing behaviors, realism, generality, and efficiency are important considerations, but robustness and control precision are paramount. Control precision includes specifying exactly where an agent should be at the end of a simulation. BSS originally allowed control precision on the order of one meter. We found this error in the agent's final location to be unacceptable. By making the changes discussed in the previous sections, we were able to increase control precision by a factor of 20. This is important since the agent often locomotes to a goal location as a prerequisite to accomplishing a task at that location. An improperly positioned agent may not even be able to

attempt the task.

The first step in creating the behavior layer is to choose the set of behaviors. We choose the following set:

- Various Avoidances
- Ducking
- Inertia
- Various Attractions
- Turning
- Path Following
- Chasing

The second step is to partition the set of behaviors.

- Instantaneous

Level 0

Reflexive: Various Avoidances

Reflexive: Ducking

Physics-Based: Inertia

Level 1

Various Attractions

- Memory-Reliant

Turning

Path Following

Chasing

The various avoidances include object avoidance (*e.g.*, walls, tables, and trees) and agent avoidance using anticipation, discussed in detail in Section 8.2.4. The various attractions include location, object, and agent attraction, the last of these using anticipation as described in Section 8.2.3. In practice we have found this set of behaviors to be sufficient for nearly all of our human locomotion requirements. Necessity and the additional properties outlined in Section 6.2.2 (locality, robustness, scalability, and simplicity) will be discussed in subsequent sections.

7.5.1 Level 0: Reflexive and Physics-Based Behaviors

Reflexive (instinctive) behaviors are necessary for collision prevention and physics-based behaviors are necessary for added realism. We discuss various avoidance, ducking, and inertia behavior implementations in this section, including the behavioral functions, and examples of the behaviors in use.

Avoidance

We treat avoidance as a default behavior that is always active as the agent explores its environment. This parallels Brooks' approach whose lowest layer of control insures that his robots do not come into contact with other objects: they achieve zero-level competence [28, 29, 30, 32]. For a good discussion on avoidance concerns and techniques, see Reynolds [132].

Both Agre [2] and Wehner [155] observe that animals do not have guaranteed general-purpose collision avoidance strategies. Instead they use simple basic avoidance techniques that are effective most of the time in concert with a small set of special-purpose avoidance techniques for specific cases. When avoidance fails, *collision prevention* halts an agent's locomotion so that it does not penetrate another object. We provide general avoidance behaviors for both stationary and moving objects including anticipating behaviors that predict where an object (or another agent) will be one or more steps in the future. In practice these behaviors are effective. However, when special cases arise, they are handled at the state machine level.

LCS uses three avoidance behaviors: general object avoidance using proximity sensors, oblong object avoidance using line sensors, and avoidance of moving objects using predictive avoidance. Each of the avoidance behavioral functions maps the state of the world through the corresponding sensor to the stress output by an avoid control behavior as described in previous sections (predictive

avoidance is described in Section 8.2.4).

Ducking

Ducking is an example of a zero-level reflexive behavior that we include both for the sake of realism and to demonstrate the architecture’s flexibility. It can support behaviors that are not directly locomotion-based, but serve to enhance a simulation’s perceived realism. A ducking behavior prevents an agent from colliding with low hanging objects such as tree branches or door frames. This behavior differs from the others in that it does not contribute to the choice of the next footstep. It is implemented as a state machine running in parallel with LCS, a child process spawned whenever level 0 behaviors are activated, and killed whenever they are deactivated. The ducking BSM (Figure 7.18) monitors the simulation and makes postural changes when appropriate. An agent ducks by bending its spine forward, though severe ducking includes bending the knees as well.

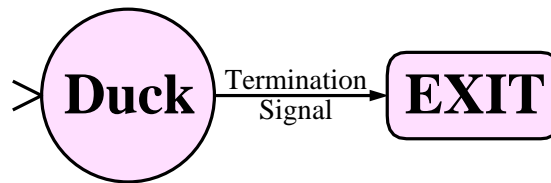


Figure 7.18: The Ducking Behavioral State Machine

A function determines how much the agent should duck to avoid a particular object. Initially, all objects which are potentially “duckable” are tagged. Then, during each simulation cycle, the duck function is applied to each duckable object which returns the height to which the agent must duck. The function first calculates the distance from the agent to the object. If this distance is less than the minimum threshold, the minimum height of the object minus a “clearance” value is returned. If the distance is greater than the maximum threshold, infinity is returned; this object requires no ducking. Otherwise, when the distance is between the minimum and maximum thresholds, the return value is calculated using linear interpolation (Figure 7.19). Once the duck-heights have been calculated for each duckable object, the agent ducks to the minimum height returned by the functions.

Using linear interpolation allows the agent to duck smoothly over time as the distance between it and an obstacle decreases rather than ducking “all at once.” It reacts appropriately when

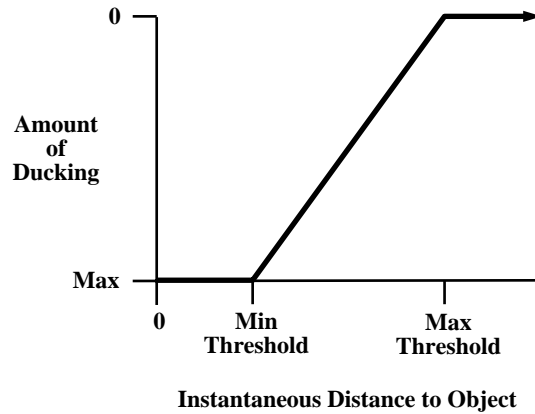


Figure 7.19: The Ducking Function

approaching an object and equally appropriately to an approaching object.

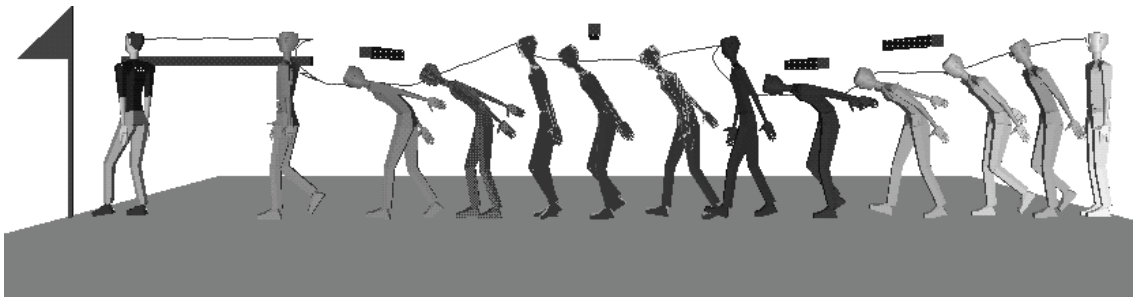


Figure 7.20: An Agent Ducking Under Objects

Figure 7.20 shows an agent ducking under four obstacles while walking to the goal on the left. The fifth, leftmost obstacle is far enough to the agent's right that it does not require ducking. A site on the head was traced to show the path.

Inertia

Inertia is an example of a physics-based instantaneous behavior used for the sake of realism. Binding an inertia behavior to an agent causes the agent to prefer to move in straight lines, suppressing its tendency to meander. Stress is proportional to the angle turned by the agent during the step and is multiplied by a function of velocity. The faster the agent is moving, the more difficult it is to turn significantly during a single step. Figures 7.21a and 7.21b show the inertia functions for a slow and a fast agent, respectively.

We note the similarity between our inertia behavior and Anderson and Donath's *forward*

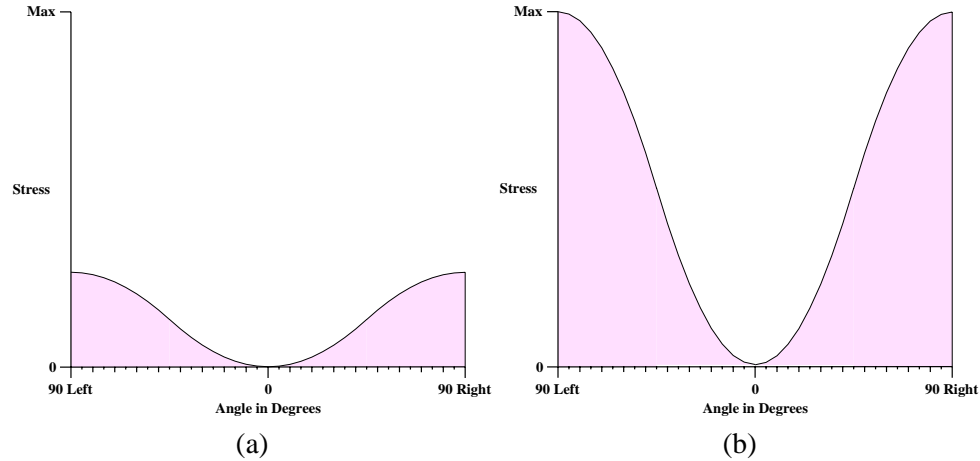


Figure 7.21: The Inertia Behavior Function: A Slow Agent (a) and a Fast Agent (b)

attraction behavior which sustains an agent’s forward motion regardless of detected objects in the environment [7]. The only difference between the two behaviors is the extent to which they contribute to the agent’s observed behavior. Forward attraction tends to be more dominant than inertia.

7.5.2 level 1: Goal-Based Behaviors

Goal-based behaviors are necessary for giving the agent a purpose, for guiding the agent to a goal. LCS uses three variants of attraction as its goal-based behaviors: general object attraction using object sensors, location attraction using location sensors, and attraction to moving humans using predictive attraction. Each of the attraction behavioral functions maps the state of the world through the corresponding sensor to the stress output by an attract control behavior as described in previous sections (predictive attraction is described in Section 8.2.3).

As discussed in Section 6.5, there are cases where a level 1 behavior alone will suffice in achieving the goal, so we make them available to LCS’s selection process by providing BSMs for each one. Attraction’s BSM is illustrated in Figure 7.22. The preconditions are twofold. The goal must be visible, otherwise path-following would be required, and the goal must be stationary (*i.e.*, a location or an object known to be fixed), otherwise chasing would be required. When the preconditions are satisfied, LCS selects and executes the attraction BSM which binds the appropriate attraction to the agent and then transitions into a monitoring state. It remains in this

state until the agent arrives at the goal (or until a user-defined interruption or termination condition is achieved) when it unbinds the attraction and exits. The example in Figure 7.12 was generated using the attraction behavioral state machine.

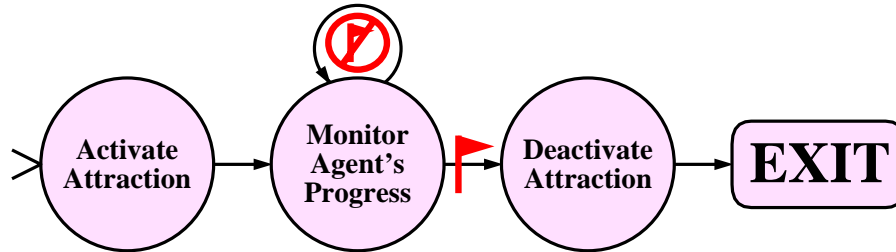


Figure 7.22: The Attraction Behavioral State Machine

7.6 State Machines

Many modern systems include a state machine implementation. Brooks implemented each layer of his subsumption architecture as a LISP augmented finite state machine (AFSM) [29, 30]. Hodgins *et al.* use state machines to enforce control laws in their simulated athletes [74]. Parallel Transition Nets (PaT-Nets) introduce high-level decision-making into our agent architecture [19, 149, 154]. As finite state machines that run in parallel with a simulation, they monitor the S-C-A loop (which can be thought of as modeling instinctive or reflexive behavior) and make decisions in special circumstances.

The observed behavior resulting from the combined use of different behaviors can sometimes fail; for example, an agent may get caught in a local minimum. Actions sometimes fail and unexpected events sometimes occur. A PaT-Net can be used to recognize one of these situations, modify the agent's behavioral mode by binding and unbinding the appropriate behaviors or by changing system parameters, and then return to a monitoring state.

Figure 7.23 shows a sample PaT-Net. PaT-Nets comprise nodes representing states, and transitions between these states. A condition is associated with each transition. When a condition is achieved, the associated transition is made. Each time a state is entered or reentered the PaT-Net performs the action associated with that state. In addition to states and transitions, a PaT-Net may include one or more *monitors*. Regardless of the current state, a monitor will perform an action when a general condition associated with it is achieved.

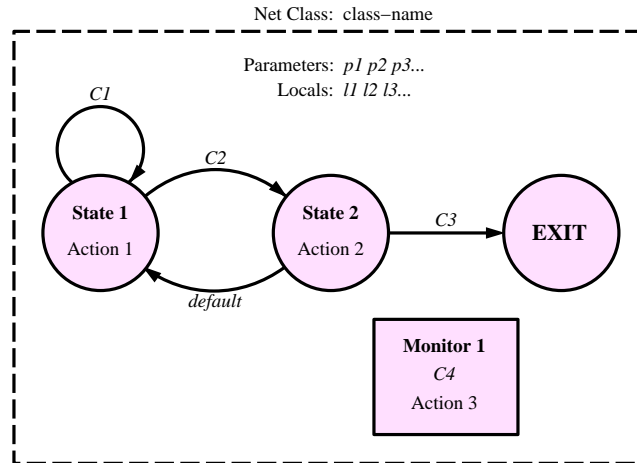


Figure 7.23: A Sample PaT-Net Shown Graphically

The actions performed by PaT-Nets are arbitrary LISP expressions. They may include spawning new nets or killing or communicating with other nets. Communication may be accomplished through the use of semaphores, priority queues, or by waiting until a condition is met. For example, a net may spawn another net and then wait for it to exit before continuing, acting effectively as a subroutine call [49].

Scheduling behaviors is based on the state of the agent and the environment. Given behaviors' state-based nature, behavioral state machines implemented as PaT-Nets are a natural structure for providing a high-level interface to the underlying behaviors. BSMs are responsible for locomotion reasoning and decision procedures for choosing, parameterizing, and controlling appropriate reactive behaviors. They may instantiate other BSMs, either running in parallel with them or waiting for them to exit.

7.6.1 Turning

When an agent walks toward a location to its side or behind it, a turning behavioral state machine turns it to face the location before attraction-based locomotion begins. Otherwise, the agent would walk in an arc to get to the goal due to the limited angle through which it can turn during one step. We do not associate conditions with the turning BSM because, in our implementation, it is only called as a child process by other state machines and its action is considered to be atomic and therefore uninterruptable.

A turning BSM (Figure 7.24) rotates an object or a human agent. An object, human, or location to face is passed as a parameter, or the angle through which to rotate, θ , is specified. If unspecified, the BSM calculates the rotation angle. The rotation direction is the lesser of clockwise or counterclockwise, or may be specified as well.

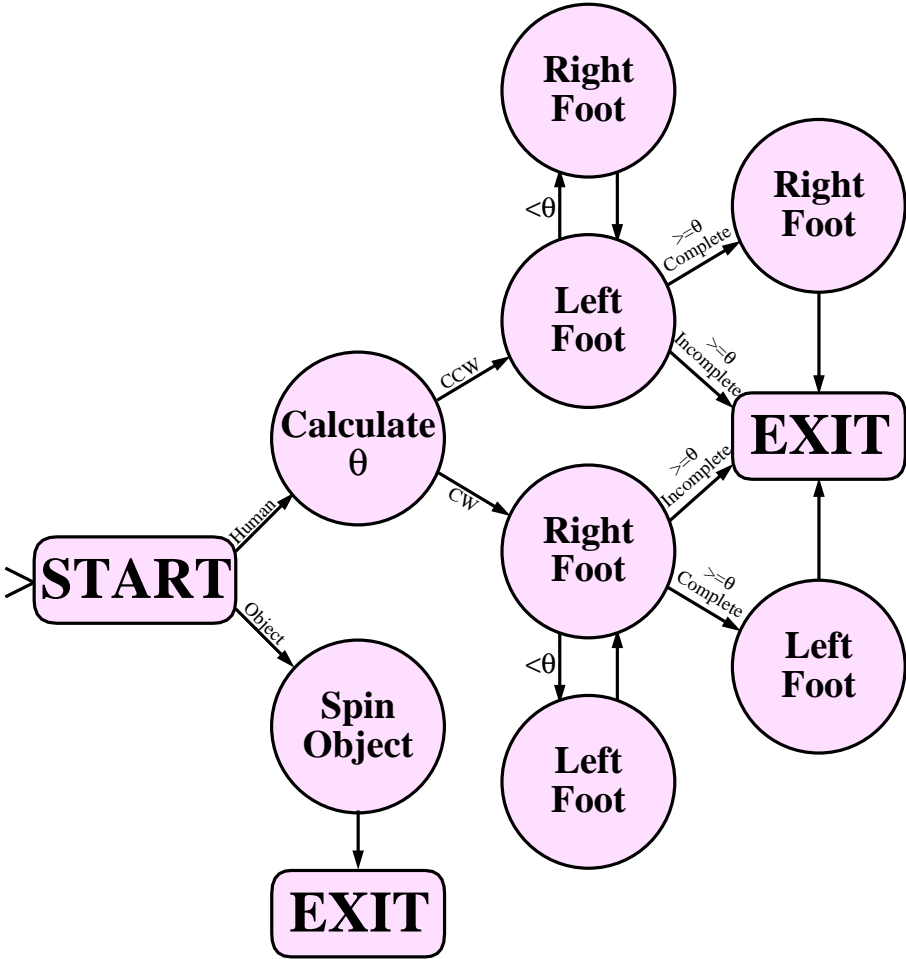


Figure 7.24: The Turning Behavioral State Machine

The turning BSM simply spins objects about their center of mass (unless a rotation point is specified) while human agents are made to step. A human agent begins its turn by rotating its first foot to the right to begin a clockwise rotation, or to the left to begin a counterclockwise rotation. The leg and body follow, maintaining the agent’s balance.² The rotation angle for the first foot is

²Research has show that many motor actions, including human turning, are preceded by eye motion or visual attention. We begin a turn by looking at the object or in the direction we want to face. Turning then occurs top-down with the lower body, legs, and feet following the eyes and head [16].

the minimum of θ and $\frac{\pi}{2}$. The agent then follows the first foot with its other foot bringing its feet together. If θ is greater than $\frac{\pi}{2}$, this cycle repeats until the agent has rotated through θ . An optional argument can specify that the agent not follow with the other foot in the final cycle. It looks more natural when the final cycle is not completed when locomotion is to follow turning.

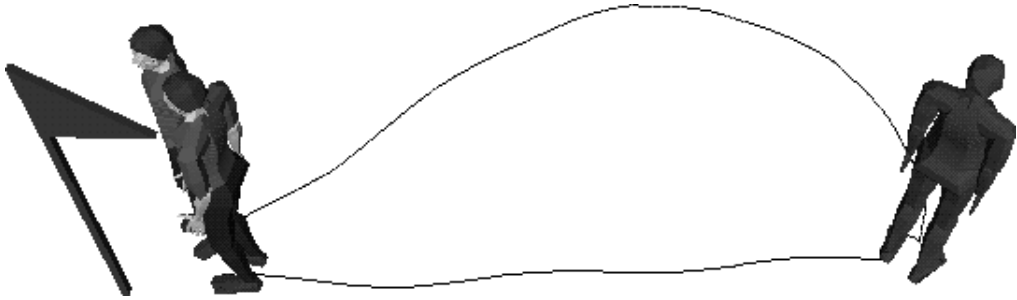


Figure 7.25: Turning to Face the Goal vs. Not Turning

Figure 7.25 shows the difference between an agent who turns to face the goal before walking towards it, and an agent who simply starts walking. Turning first is sometimes necessary as when maneuvering in a restricted area such as a narrow hallway.

7.6.2 Path Following

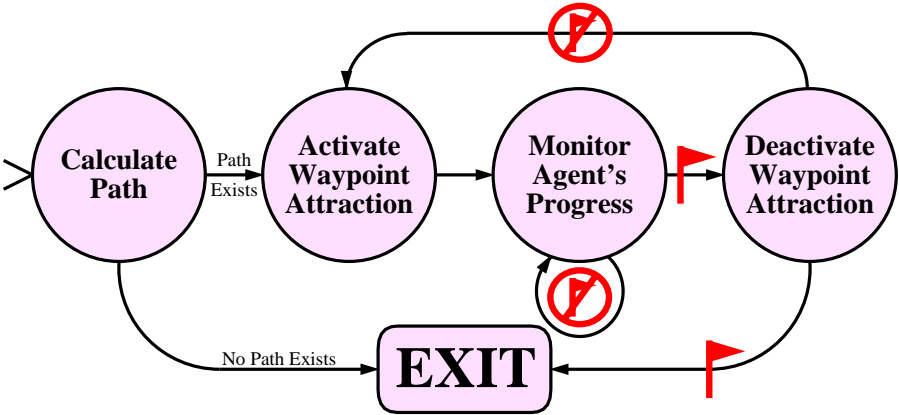


Figure 7.26: The Path-Following Behavioral State Machine

Path-following is useful for realistically guiding an agent to a goal that is not immediately visible. A path-following BSM (Figure 7.26) generates a segmented path from the agent's location to a goal somewhere in the world. Given a set of user-defined constraints, such a path may not exist: in this case, the machine exits reporting this information. Otherwise, an attraction to the first vertex

(intersection of consecutive segments) along the path binds to the agent; the agent begins following the path.

The path has three important properties: it connects the agent to the goal, it avoids obstacles by at least the *obstacle-clearance value* chosen by the user, and it is a path of minimal length given the first two constraints. As the agent approaches a vertex, the attraction drawing it there unbinds and an attraction to the next vertex along the path binds to the agent which smoothly transitions between them. This process repeats until the agent arrives at the goal.

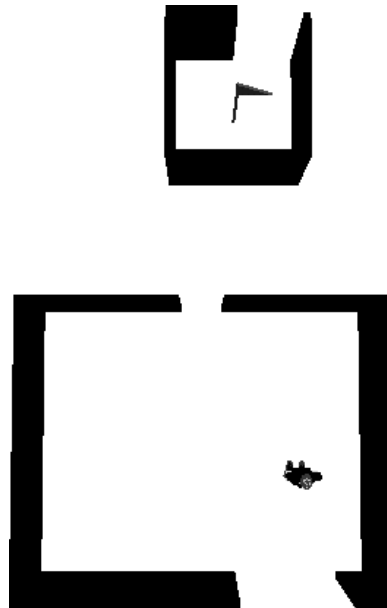


Figure 7.27: The Initial Conditions of the Path-Following Example

Figure 7.27 shows the sample environment we use to demonstrate our path following algorithm. The agent in the large room at the bottom of the figure, is told to walk to the goal location, marked with a flag, in the small room at the top of the figure. LCS performs a visibility test initially and determines that the goal is not visible to the agent, thus path following is required. It selects and executes a path-following BSM.

Given boundary, resolution, and obstacle clearance values (user-defined or default), the system casts rays to generate an obstacle map.³ Figure 7.28a shows the results. The boundary is two meters in the X direction (up-down) and three meters in the Z direction (left-right) beyond the

³To optimize ray casting, we take advantage of the fact that the rays are cast parallel to the Y-axis and the fact that all objects include axis-aligned bounding boxes. In practice it is fast enough not to be a weak link in the system.

environment's farthest extents, making the size of the path-generation region 12 meters (X) by 11 meters (Z). The resolution is 122 in the X direction and 112 in the Z direction - just enough to ensure that walls (which are 10 centimeters thick) are not missed due to floating-point error. The obstacle-clearance value is 30 centimeters, *i.e.*, the path should avoid obstacles by this distance. We satisfy this constraint by expanding obstacles by the obstacle-clearance value in each direction (Figure 7.28b), thus reducing the path-generation problem to the simpler problem of trying to find a minimal-length path connecting the agent to the goal that avoids obstacles. Avoiding the expanded obstacles is equivalent to avoiding the obstacles by the expanded amount [92]. It is interesting to note that the agent's path will take it through the lower doorway since the upper doorway is only 60 centimeters wide and thus not wide enough to walk through given the 30 centimeter clearance constraint.

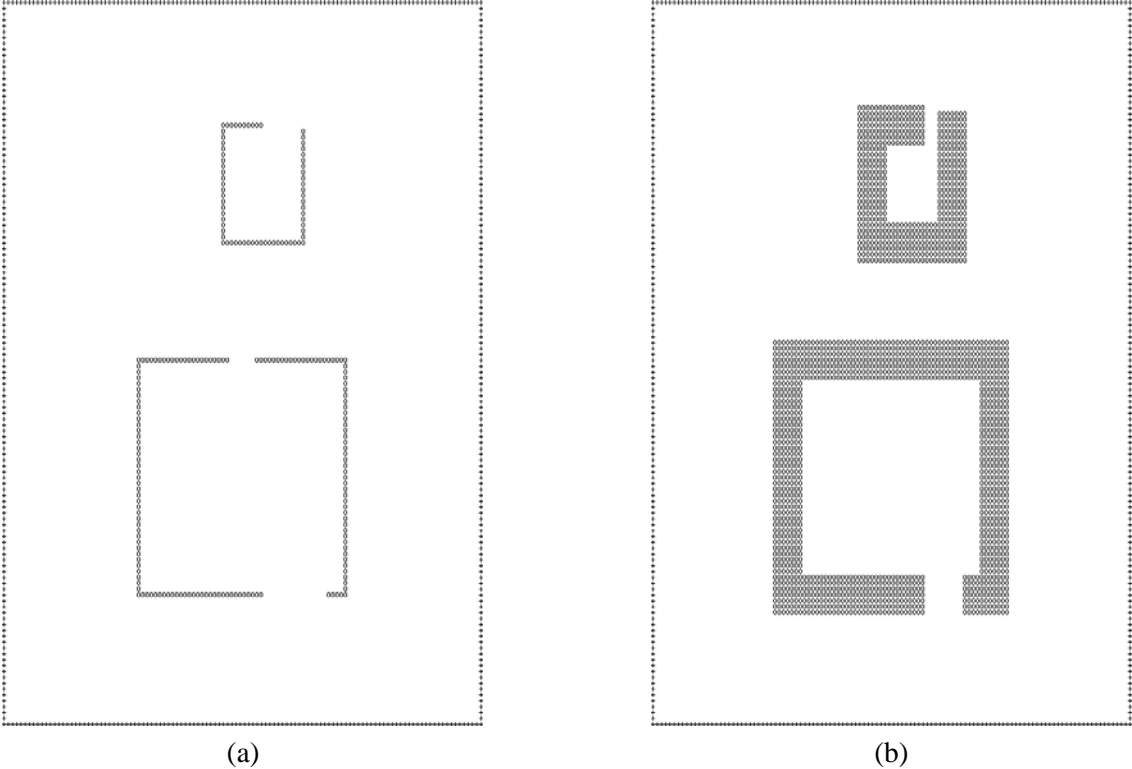


Figure 7.28: The environment obstacle map before (a) and after (b) expansion

Path Generation is accomplished through the use of a wave-propagation algorithm [90] that takes as input the expanded-obstacle map of Figure 7.28b and the agent and goal locations (Figure 7.29). The algorithm can be understood most easily through analogy. Imagine that the goal

location is the source of a viscous fluid expanding uniformly and filling the obstacle map. The fluid cannot penetrate obstacles or the outer boundary. When the fluid contacts the agent it stops flowing. The actual implementation begins by discretizing space as indicated by the figures and marking the goal as being “explored” with an associated weight (the shortest distance to the goal) of zero. Then the outer boundary of the explored region is repeatedly expanded into neighboring rectangular grid points, except when those grid points are marked as containing obstacles or are out of bounds.⁴

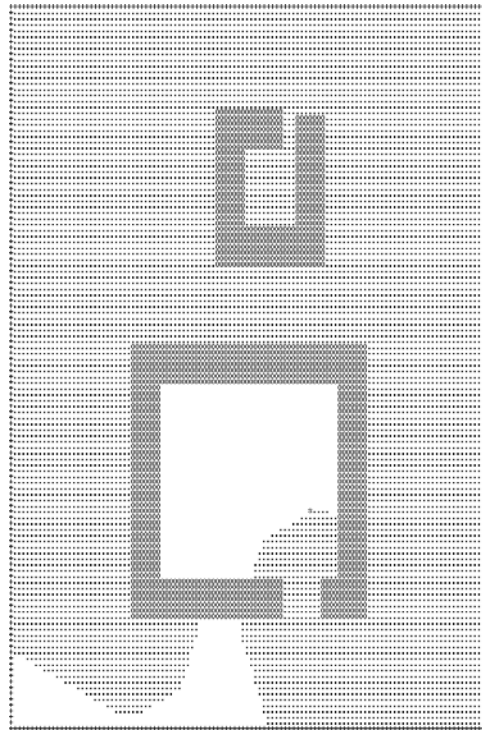


Figure 7.29: The expanded-obstacle map after wave-propagation

The associated weight of any new grid point is the minimum of the weights of its neighbors plus 1 if the neighbor is horizontally or vertically adjacent, or plus $\sqrt{2}$ if the neighbor is diagonally adjacent. Figure 7.30 is an example of a small weight map. The goal is at the center, the agent is in the upper-right corner, and the dark, Xed grid points represent obstacles.

A minimal-length path from the agent to the goal can now be found by starting from the agent

⁴We found it necessary to maintain a sorted list of boundary grid points, always choosing the grid point of minimal weight for exploration. A heap data structure for storage proved efficient given its $O(\log n)$ time complexity for both insertion and deletion.

2.8	2.4	2	2.4	3.4
2.4	1.4	1	X	3.8
2	1	0	X	2.8
2.4	1.4	1	1.4	2.4
2.8	2.4	2	2.4	2.8

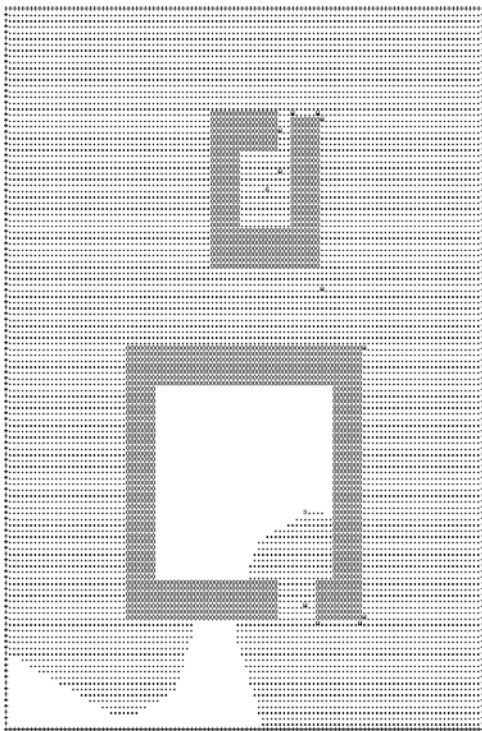
Figure 7.30: The Weight Map

grid point and repeatedly moving in the direction of the neighbor of minimal weight (Figure 7.31). In the case of a tie the previous direction is given preference, producing long, linear runs of grid points. We choose the waypoints to be the ends of each of these runs. Figure 7.32 shows the waypoints in our original example, both on the current map and on the original obstacle map, for clarity.

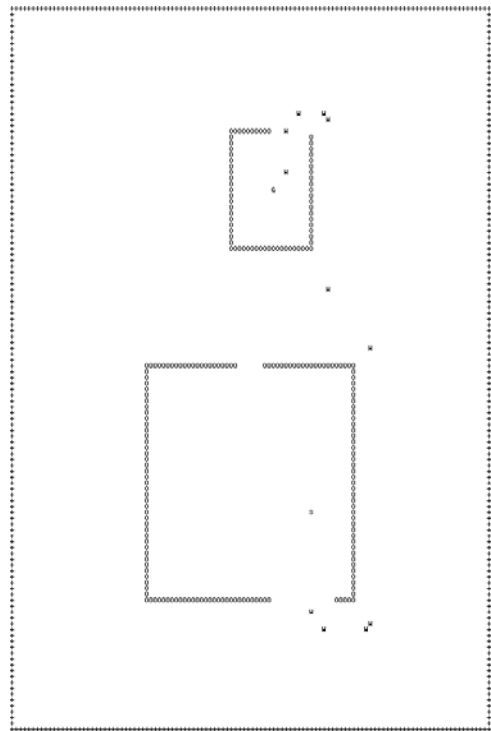
2.8	2.4	2	2.4	3.4
2.4	1.4	1	X	3.8
2	1	0	X	2.8
2.4	1.4	1	1.4	2.4
2.8	2.4	2	2.4	2.8

Figure 7.31: The Path to the Goal

Note that this method produces extraneous waypoints due to the limited choice of direction, discretized at 45° intervals. We remove these unnecessary waypoints by doing visibility tests in map-space using Bresenham's incremental line scan-conversion algorithm [55] for efficiency. For any three consecutive waypoints, if the first and third can be connected by a line that does not intersect an obstacle, then the second is removed. The resulting set of waypoints forms a path that takes the agent to the goal efficiently (Figure 7.33).

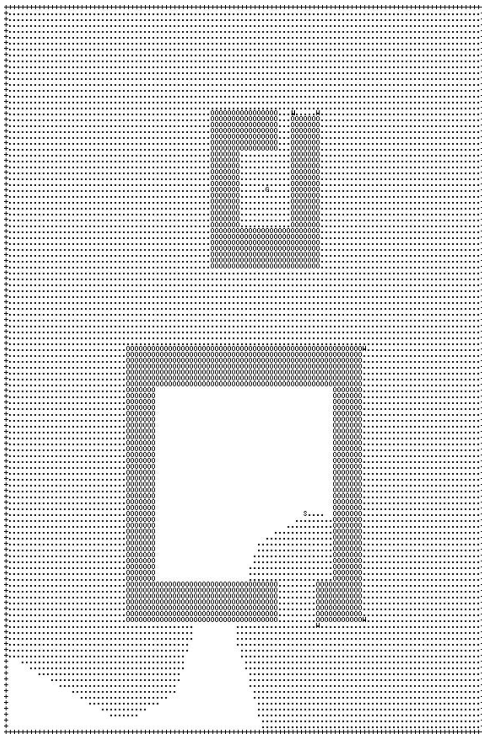


(a)

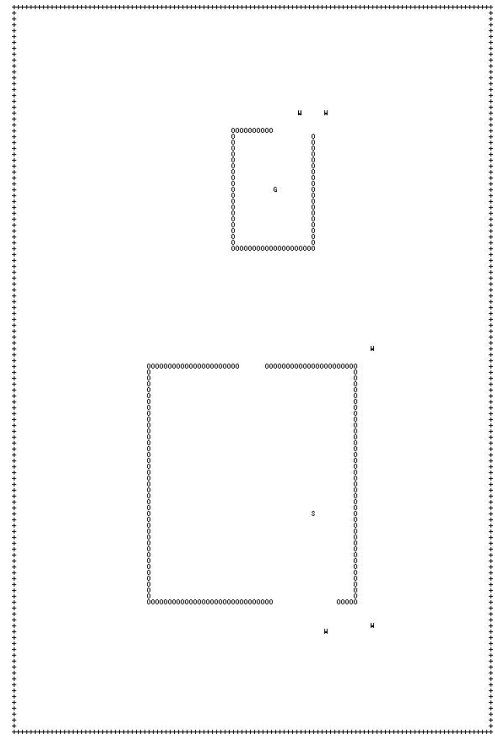


(b)

Figure 7.32: The waypoints (a) - The waypoints drawn on the original obstacle map for clarity (b)



(a)



(b)

Figure 7.33: The necessary waypoints (a) - The necessary waypoints drawn on the original obstacle map for clarity (b)

Figure 7.34a shows the agent's path as calculated by the path-following BSM. Figures 7.34b and 7.34c show the agent following the path, and Figure 7.34d shows the agent successfully arriving at the goal.

Although the system will accept a user-defined path as an optional parameter, in general it is responsible for path generation. In this way it differs from other systems such as Miller's Rocky III which must be given an operator-designated set of intermediate way-points [110].

Path following poses two questions: is it reasonable for the agent to behave in this way, and is this not simply path planning? We argue that it is reasonable to generate a path from one point to another provided that the agent "knows" the area. We use this BSM to take the agent to a place where it has already been, through an area it has already explored.

Path following conforms to Agre and Chapman's *Plans-as-Advice* paradigm [4]. It differs from path planning because the agent is not constrained to the path. Instead, an attraction to the path binds to the agent. This behavior competes with any other bound behavior for control of the agent; if a black cat crossed its path, a superstitious path-following agent would stray from the path to avoid the cat.

7.6.3 Chasing

An agent-following or chasing behavior alternates between object attraction and location attraction behaviors. While the target (the agent being chased) is visible, an attraction to it binds to the chasing agent. While the target is occluded, an attraction to the target's last known location is bound instead.

If the target is not immediately visible, the chasing BSM exits unsuccessfully. Otherwise, the agent turns to face the target and an attraction to the target binds to the agent. The agent walks toward the target until either the agent arrives at the target and the machine exits successfully, or the target becomes occluded (the target may have walked around the corner of a building, for example). In the latter case the existing target attraction unbinds and an attraction to the target's last known location binds to the agent. The agent walks toward this location until either it arrives or the target becomes visible again. In the latter case the agent again turns toward the target and chases it. In the former case the machine transitions to the extrapolation state.

When the agent arrives at the last known location of the target without seeing the target, instead of exiting unsuccessfully, the agent is guided in the direction the target was walking before

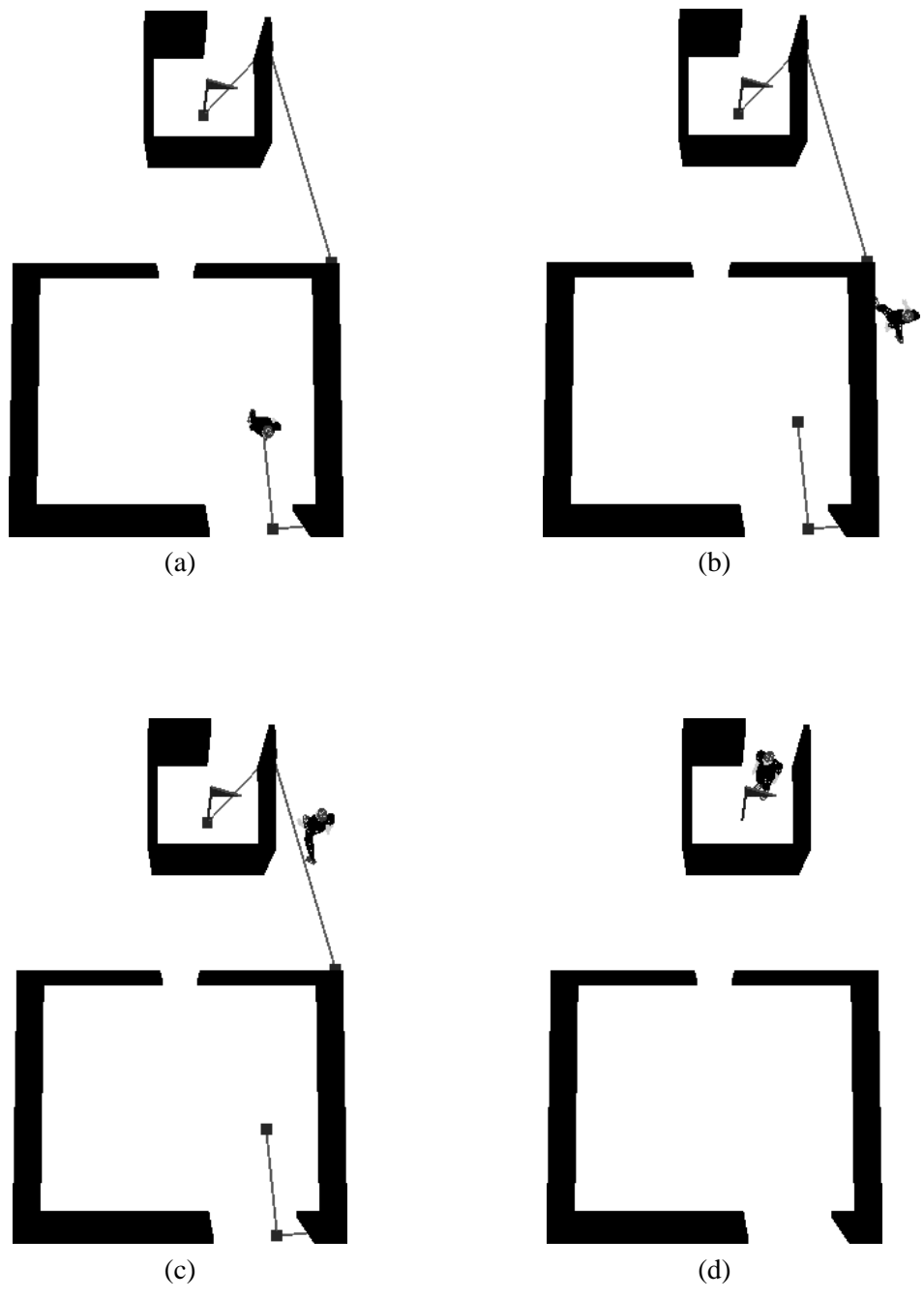


Figure 7.34: The Agent Following the Path

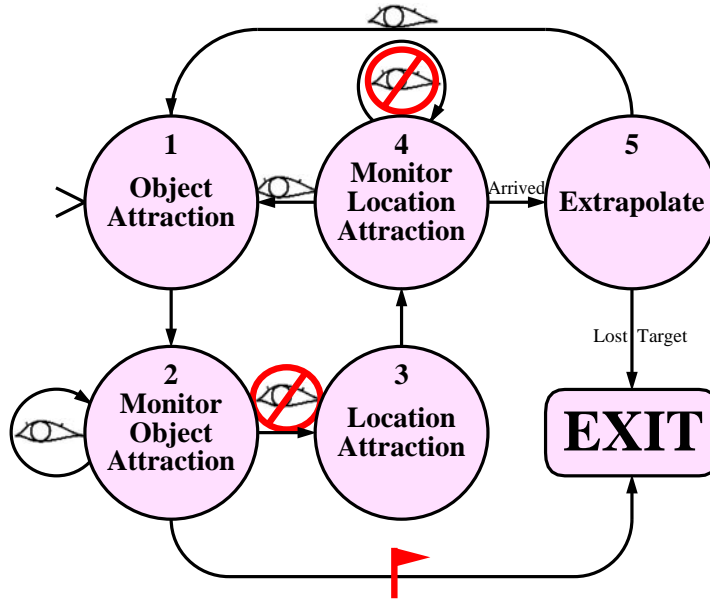


Figure 7.35: The Chasing Behavioral State Machine

disappearing. This requires maintaining two pieces of information, the target’s last known location and its vector velocity. Extrapolation is optional and may be turned off to allow for a more effective higher-level implementation.

To illustrate, we give an example where Tom is chasing Jerry. The chasing BSM shown in Figure 7.35 begins in **state 1**. An attraction to Jerry binds to Tom. As Tom begins to run toward Jerry the BSM passes to **state 2**, a monitoring state. When Jerry is no longer visible to Tom (Jerry may have run around a corner or behind an object), the BSM enters **state 3**. An attraction to the location where Jerry is most likely to be found, Jerry’s last known location, binds to Tom. Tom begins to run toward this location as the BSM transitions to **state 4**. If Tom arrives at this location and does not see Jerry, the BSM transitions to **state 5** and Tom searches in the direction Jerry was last known to be heading. Eventually either Tom will see Jerry again and continue chasing him, or Tom will give up at which point the chasing BSM exits. Section 9.3 contains a detailed example of chasing.

7.6.4 Combining the State Machines

Given the BSMs and preconditions for simple path-following, turning, attraction, and chasing, we combine them into a single state machine (the locomotion control machine, LCM) as described in

Section 6.5. First we examine the preconditions to determine the conditions necessary to implement LCM (Figure 7.36).





				
EXIT	Y	Y	–	–
Path Following	N	–	N	–
Turning	Y	N	–	–
	N	N	Y	–
Attraction	N	Y	Y	Y
Chasing	N	Y	Y	N

Figure 7.36: The BSM Preconditions

The symbols in the figure, from left to right, indicate whether or not the agent, Jack, is at the goal, if he is properly oriented, if he can see the goal, and whether or not the goal is stationary (*i.e.*, unmovable for the duration of the currently requested locomotion). If Jack is at the goal and properly oriented, LCM exits. If he is not at the goal and cannot see it, path-following is required. If he is improperly oriented and either at the goal or the goal is visible, then turning is required. If Jack is not at the goal, properly oriented, and able to see the goal then either attraction or chasing is required depending on whether or not the goal is stationary. The flow diagram in figure 7.37 clarifies this decision-making process. LCM and the encompassing LCS are described in more detail in Section 9.1.

7.7 Agent Model

LCS includes an agent model as outlined in Section 6.4. It includes personality traits such as curiosity and caution, and state information such as the agent’s energy level and alertness. The personality traits and state information affect system parameters throughout the architecture. For example, fatigue affects the agent’s speed in the locomotion engine layer, anticipation affects attraction and avoidance in the behavior layer, and curiosity affects condition evaluation in the state machine layer. The user sets high-level locomotion goals for the agent, from which the system

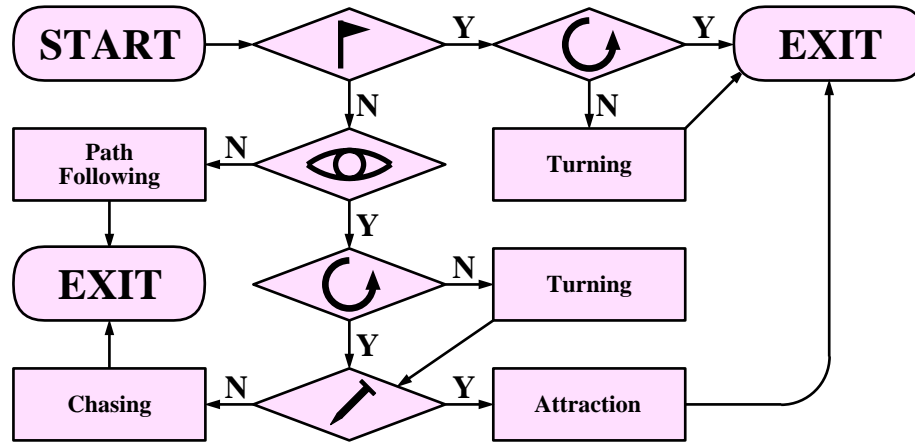


Figure 7.37: The Locomotion Control Machine Flow Diagram

generates locomotion automatically, shaping the locomotion to reflect and convey to an observer the chosen agent state and personality.

As an example, consider an animator who is given the task of modeling two agents: Homer and Marge. In the scene to be animated, Homer has just left the local bar and is on his way home. He is moderately drunk, minimally aware of his surroundings, and generally flirtatious. Marge, Homer’s girlfriend, having just received a call from the bartender, is on her way to intercept Homer. She is in a hurry, knowing Homer’s propensity to get himself into trouble in this state.

The animator opens the agent model interface window and configures the agents beginning with Homer. He clicks on the “Inebriated” button with the mouse and a slider-bar entitled “Intoxication-Level” appears. He chooses a setting of 60 (out of 100). Then he sets Homer’s “Awareness” of his surroundings to 15. Finally, the animator wants to model Homer’s flirtatiousness, but finds that no such trait exists in the system. He clicks on “Add New Attribute” and answers the questions as they appear. He defines a LISP condition which is achieved when Homer can see a woman to whom he is attracted. The probability of reacting to this condition when it is achieved he relates to Homer’s awareness level.

The animator configures Marge in a similar way setting her “Rushed” value to 90. He chooses locomotion goals instructing Homer to go home and Marge to go to the bar. In addition, he adds an interruption condition to Marge’s high-level behavioral schema causing termination when Homer becomes visible. If this happens, a new schema is adopted with the goal “go to Homer” followed by schemata that take Homer and Marge home. The animator tests the configurations by starting

the locomotion control systems and this is what he sees:

Homer begins walking home. He is walking relatively slowly, though his speed varies, and he is staggering. Marge is on her way to the bar. Her walking rate is faster and much more stable. Homer passes a woman sitting on a bench who might distract him under normal circumstances. He walks by the bench without noticing her which is fortunate for him because at that moment Marge comes around the corner into view. She turns towards Homer, they meet, and they walk home together.

The animator shows the animation to a colleague. After a few minutes of discussion and a small amount of parameter adjustment, he is satisfied that the agents appear to have the desired personalities.

Attribute	Locomotion System Parameters Affected				
	Speed	Inertia	Anticipation	Condition	Probability
Fatigued	X				
Laden	X				
Rushed	X				X
Inebriated	X	X	X		X
Aware			X		X
Alert to Danger				X	
Curious				X	X
Distracted					X

Figure 7.38: How the Agent Model Affects the Agent

Figure 7.38 shows a subset of the attributes that make up the agent model and some of the ways each one affects the agent’s behavior. **Speed**, or walking rate, is one of the factors regulated by the “rushed” and “fatigued” attributes. A fatigued or laden agent walks slowly while an agent who is in a rush walks quickly or runs. An inebriated agent walks slowly and at an inconsistent rate, varying his velocity an amount proportional to the level of intoxication. Figure 7.39 shows the effects of our model of intoxication on an agent. The two simulations are identical except that the agent on the left is sober and the agent on the right is drunk.⁵

Figure 7.40 shows the inertia behavior function for a drunk agent. This function encourages

⁵When we discuss agent states or attributes such as intoxication, or drunkenness, we do not claim to have fully modeled the state or attribute in question. Instead, we have approximated its effects on the agent’s locomotion. Our drunken agent may be thought of as exhibiting “drunk-like” behavior.

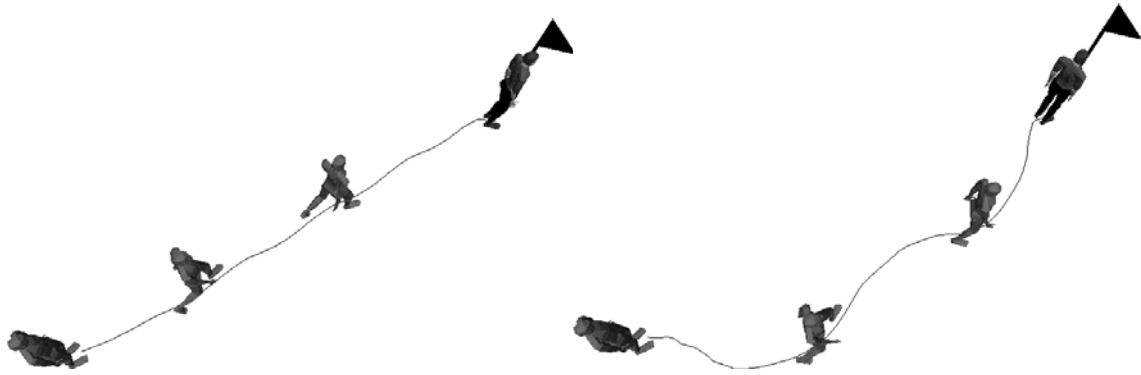


Figure 7.39: An agent exhibiting “drunk-like” behavior through the use of the agent model

the agent to step to the side rather than straight towards the goal, effectively causing the agent to stagger.

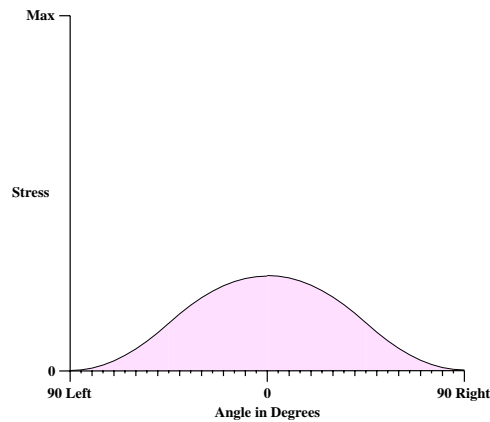


Figure 7.40: The Inertia Behavior Function: An Intoxicated Agent

The combination of several behaviors, as is necessary with behavioral control, typically yields excessively wandering paths. In an attempt to straighten locomotion paths and also to realistically model the effects of inertia, we add an “inertia” behavior to an agent’s behavior set which attracts the agent to the forward direction. An inebriated agent is given a low **Inertia** value and tends to meander as a result.

When we walk around in the presence of other people we attempt to react to them based not only on their current location, but also on a guess of where they will be one or more steps in the future. Consciously or unconsciously, we anticipate their potential locations in the near future and base our locomotion reasoning on this prediction. An inebriated agent has a lower **Anticipation**

value than normal, while an agent who is particularly aware of its surroundings will anticipate more.

Condition is a special system feature implemented at the state-machine level. It is a *hook*, effectively an interruption, which allows the user to implement behaviors, particularly those involving sub-goals, and make decisions that would otherwise be unsupported by the architecture. In this way it differs from the weighted summation implementation of the other parameters. When an arbitrary condition is achieved, the state-machine takes an appropriate action, possibly stopping the agent and exiting. This allows a higher-level system to replan, taking advantage of the opportunity. A condition, for example, might be achieved when an enemy is in the agent's field-of-view as in this scenario. We see a burglar walk past a police officer. The burglar notices the officer and starts running (his sense of urgency increases). He tries to find a good hiding place while the officer pursues him. Our system is capable of simulating this scenario.

Although the burglar passes the officer, in real life, there is no guarantee that they would notice each other. The **Probability** parameter reflects the probability that an agent will react to the condition being achieved, *i.e.*, that the burglar notices the officer or *vice versa*. The probability is high when the agent is aware of its surroundings or curious, but low when he is distracted, inebriated, or rushed, for example.

An advantage of this agent model implementation is its modularity. To add a new attribute, all that must be done is to establish the correspondence between the attribute and the system parameters as described in Section 6.4. The system adapts automatically, taking the new attribute into account when calculating parameter values.

Chapter 8

Issues

In this chapter we discuss the issues of limited perception and anticipation, both of which are required for simulation realism.

8.1 Limited Perception

Since realism is one of the goals of this work, the concept of limited perception is an important concern [112, 113]. Behaviors have access to the entire environmental database. A behavior's designer filters this information in such a way that an agent's behavior depends only on reasonable information. Reasonable information is that which a real agent in an equivalent situation would be able to perceive, usually visually. For example, in Tu and Terzopoulos' fish controller, an object is only considered to be "seen" if any part of it enters the fish's view volume and if it is not fully occluded by another object [150].

Filtering can be performed at either the behavior level or the higher state-machine level. At the behavior level, avoidance combined with a proximity sensor approximates the effects of limited perception. An agent only avoids nearby objects, those of which the agent is aware. Attraction combined with an object sensor produces a behavior that attracts an agent to an object regardless of distance or visibility. The state machine level repairs this behavior by activating and deactivating the attraction to simulate realistic path-following or chasing behaviors as described in sections 7.6.2 and 7.6.3.

We give an example (Figure 8.1) of how enforcing the limited perception policy affected

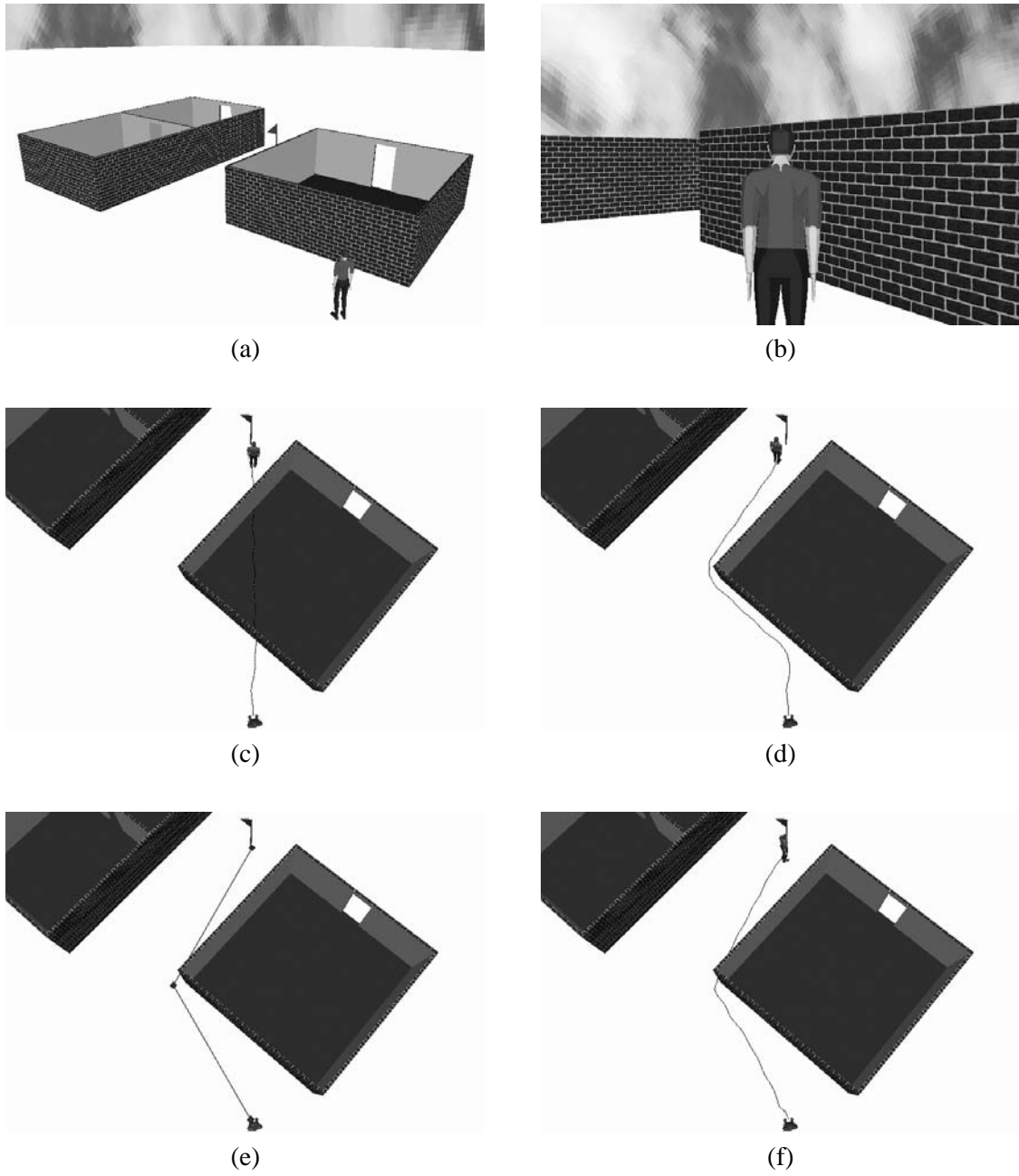


Figure 8.1: Enforcing the Limited Perception Policy: (a) The environment (b) The agent's view (c) Attraction only (d) Attraction and wall avoidance (e) The path (f) The agent arriving at the goal

system design. In the environment shown in Figures 8.1a and 8.1b, we wanted the agent to walk to the goal (the flag). Our first thought was to use an object sensor to attract the agent to the goal. Figure 8.1c shows the results of the first test. Figure 8.1d shows the results after adding avoidances. The agent was “dragged” along the wall until the way to the goal was clear.

Analysis of our results revealed that the unrealistic agent dragging was due to an attraction to a non-local, obscured object. While it was reasonable for the agent to be aware of an object it could not see, it was unreasonable for the agent to walk directly towards that object, relying on avoidances to push it away from the intervening walls.

Two alternatives to this approach presented themselves, both of which adhere to the limited perception policy. One possibility was to simply disallow this behavior, requiring the user or planner level to break this navigation down into shorter, local “sub-navigations.” The second possibility was to consider what we would do in a similar situation and add that behavior to the system’s repertoire.

Our solution was to develop the path-following behavior previously described in Section 7.6.2. Figure 8.1e shows the results of the path-generation algorithm and Figure 8.1f shows the agent arriving at the goal after following the path. The agent’s starting point and path are also shown. This solution has the advantages of increasing the system’s overall realism, flexibility, and ease of use.

8.2 Anticipation

LCS employs anticipation at every level and supports planner-based anticipation through condition evaluation and system interruption; the locomotion engine bases its calculations not on the agent’s current location, but on where the agent will be one step in the future. At the behavioral level the agent anticipates the world through sensor-shape adjustments, through lookahead exhibited by the terrain awareness behavior, and through explicit anticipating versions of attraction and avoidance behaviors. The state machine level includes anticipation in the chasing behavior’s extrapolation state.

8.2.1 Locomotion Engine

A velocity vector resulting from the blended output of all active behaviors could be used to determine the next footstep; however, as Becket observes [20], this would result in severe instability around threshold boundaries. This occurs because thresholds in sensors and control behaviors create a discontinuous potential field space. Taking a discrete step based on instantaneous information may cross a discontinuity in field space.

For example, consider the situation in Figure 8.2a where a goal attracts the agent, who avoids the hot radiator up to some threshold distance. If the agent is initially located at position p1, it will choose to step directly toward the goal and will end up at p2. Now within the threshold distance for the radiator, the agent will step away from the wall and end up at p3, outside the threshold. This process repeats until the agent clears the radiator, producing an extremely unrealistic sawtooth path about the true gradient in the potential field.

In calculating the stress values of all possible next agent-states instead of relying on instantaneous information, the system looks one step into the future.¹ This anticipation results in more realistic locomotion as shown in Figure 8.2b, particularly when the local environment is static. When the local environment is dynamic, it is also necessary to predict as many changes in the environment (next world-states) as possible to maintain reasonable realism (Section 8.2.3).

8.2.2 Terrain Awareness

A terrain awareness behavior evaluates an agent's position and orientation in the environment based on the local terrain. A weight proportional to the stress of walking through terrain of a particular type is associated with that terrain type; easily navigable terrain such as grass receives a low weight, and difficult terrain such as mud receives a high weight. This is easily extended to terrain with changing elevation by factoring in to the stress calculations the agent's metabolic energy expenditure in taking a step.

A terrain awareness behavior does more than simply return a stress proportional to the weight of the terrain where the agent is stepping. It predicts the agent's next N steps (assuming the agent

¹Theoretically, the system would benefit from a lookahead of more than one step. However, we limit it to one because of the $O(x^n)$ time complexity of lookahead, where x is the number of potential new states being examined and n is the number of steps of lookahead. Consider, for example, a system where 50 possible next footsteps are examined. Looking one step into the future requires 50 sets of calculations while looking two steps into the future requires 2500. Due to the time required for each set of calculations, it is not feasible to look more than one step into the future.

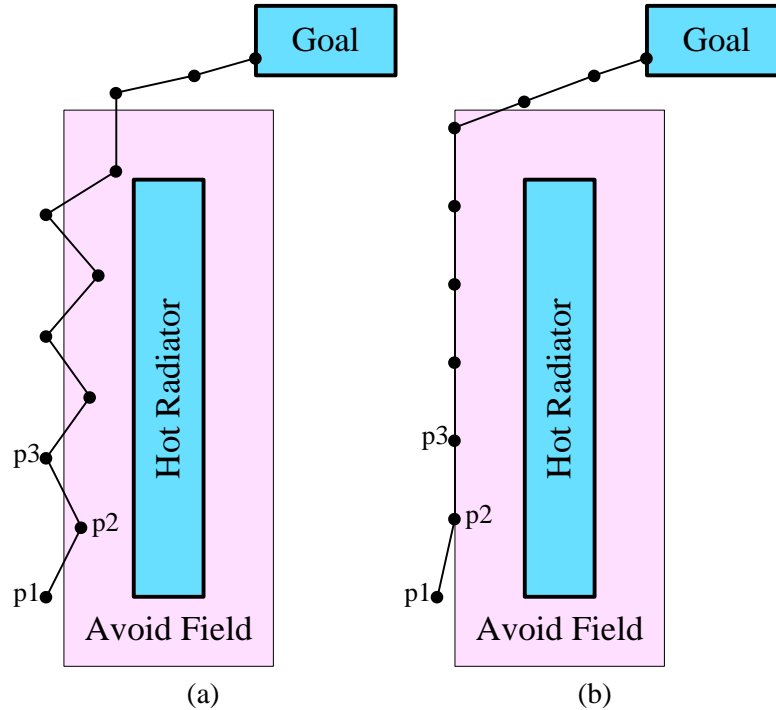


Figure 8.2: (a) A sawtooth path due to potential field discontinuities, and (b) Better results through anticipating the state of the world after the agent has taken the step

continues to walk in a straight line) where N is a parameter. The terrain is sampled at each of these steps and the associated weights are multiplied by a function that decreases with each step. Stress is proportional to the sum of these values. This method's advantage is that the agent will step into a region of difficult terrain if the region beyond is easily navigable.

Figure 8.3 is an example of real-time terrain traversal by three simulated agents. The figure shows the agents' starting positions at the bottom of the image and their paths through the terrain. Attraction guides the agents toward the goal in the upper left corner, avoidance prevents the agents from colliding with obstacles and each other, and terrain-awareness causes the agents to avoid water (the dark patches) whenever possible. The effect of the terrain awareness behavior can be seen most clearly where the rightmost agent chooses to turn slightly to cross the stream instead of walking straight toward the goal. Figure 8.4 shows the effects of the terrain-awareness behavior in more detail.

The winding exhibited by the center agent resulted from trying to avoid the leftmost agent who was slightly ahead of it. This observed behavior can be eliminated by replacing avoidance with

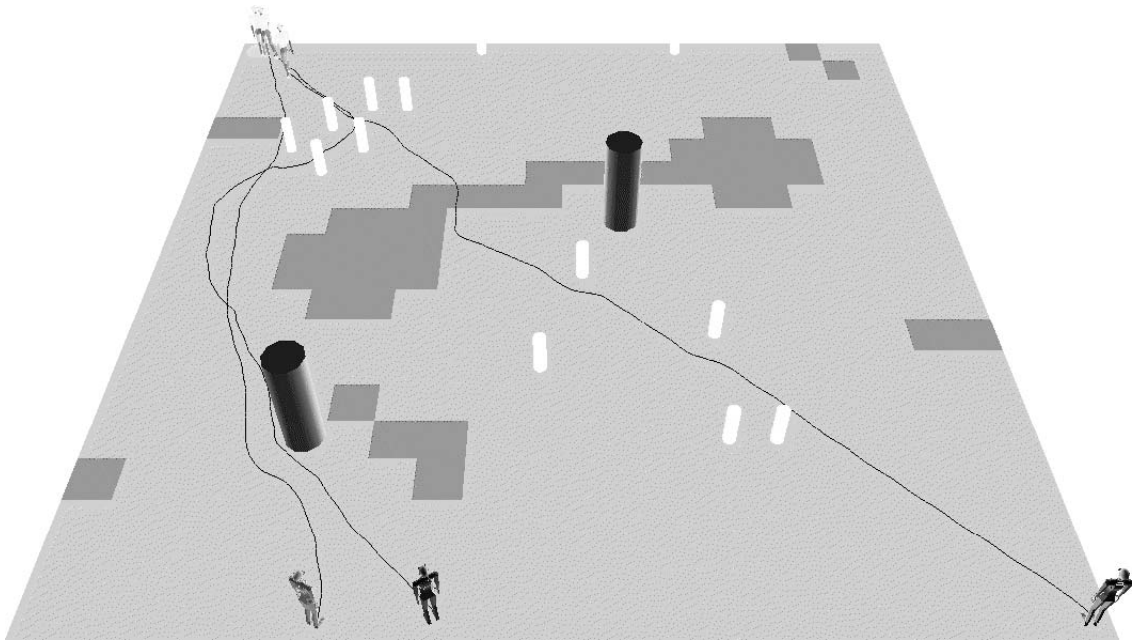


Figure 8.3: Attraction, avoidance, and terrain awareness behaviors combine to draw the three agents to the goal in the upper left corner

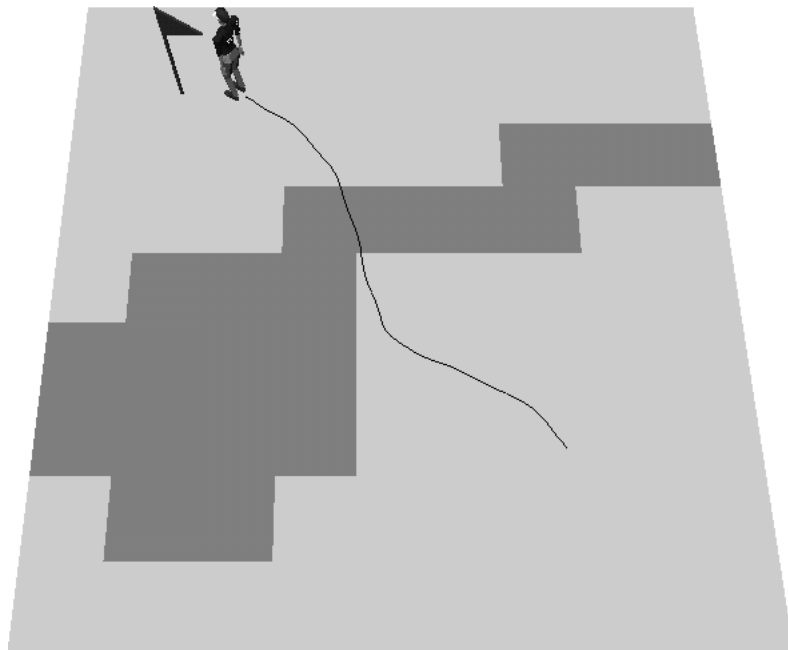


Figure 8.4: A close-up view of an attract behavior combining with a terrain awareness behavior to guide the agent to the goal

predictive avoidance. Toward the end of the simulation the leftmost agent was standing between a tree and a puddle (upper left). The combination of obstacle and agent avoidance and terrain-sensor output resulted in the central agent walking around the tree.

8.2.3 Attraction

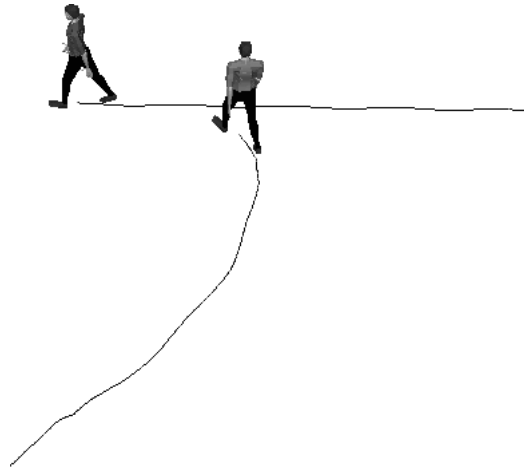


Figure 8.5: The lower agent is attracted to the upper agent using a traditional attraction behavior

Our desire for an accurate predictive attraction behavior came from a dissatisfaction with traditional techniques where an agent is attracted to a target's current location. Figure 8.5 shows what happens when a traditional attraction behavior is bound to the chasing agent coming from the bottom of the figure. The chasing agent's path is curved to such a degree that it eventually falls behind the target agent. This example motivated us to replace the naïve object attraction behavior whenever the target is moving.

First Implementation

Predictive attraction evolved from traditional attraction in two stages. First we experimented with an attraction to the location in which the target is predicted to be one step in the future. An object sensor and an attract control behavior compose an object attraction behavior, so stress is reduced when the agent to which it binds turns toward and approaches the object. Predictive attraction replaces the object sensor with a predictive object sensor that differs by reporting the location where the object is expected to be one step in the future.

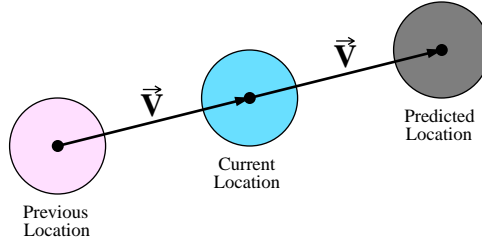


Figure 8.6: The Mathematics of the Predictive Attraction Behavior

Figure 8.6 illustrates the implementation. The predicted location is calculated as:

$$\vec{L}_{Pred} = 2\vec{L}_{Curr} - \vec{L}_{Prev} \quad (8.1)$$

where \vec{L}_{Prev} , \vec{L}_{Curr} , and \vec{L}_{Pred} are the target's previous, current, and predicted locations respectively. The model assumes that the agent will maintain its current velocity, \vec{V} .

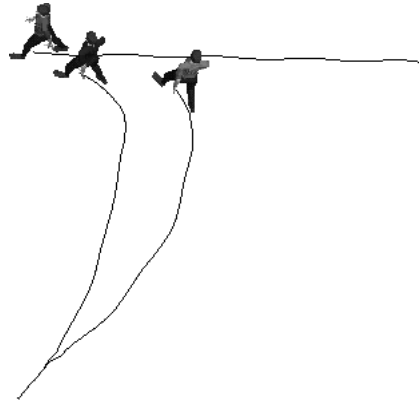


Figure 8.7: The First Attempt at Predictive Attraction

Figure 8.7 shows that, although the anticipating agent's path is better (the agent on the left, approaching from the bottom), the agent is not anticipating the target appropriately. Let N be the number of steps in the future for which we are predicting the agent's location. Until now we only considered the case where $N = 1$. Experiments with $N \geq 2$ were no more encouraging than for $N = 1$. The problem is that anytime the anticipating agent is drawn to a point that is fixed relative to the position of the target (regardless of the distance of the fixed point to the target), the anticipating agent's trajectory will be curved. An optimum solution is one in which the anticipating agent walks in a straight line.

Best Implementation

We obtained the best results by choosing the direction for the agent that brought it within a threshold distance of the target as quickly as possible (again assuming constant velocities). Effectively, the agent followed an intercept course.

The stress of a potential footstep is calculated as follows. Both the agent and the target (*Agent'*) have an initial position and velocity. From this, and assuming both agents will maintain their velocities, the goal is to determine how long it will take them to be within a threshold distance of each other. This step's stress value is proportional to the calculated time, measured in seconds. We will also consider the cases where the agents will not come within the threshold distance of each other and where they are diverging.

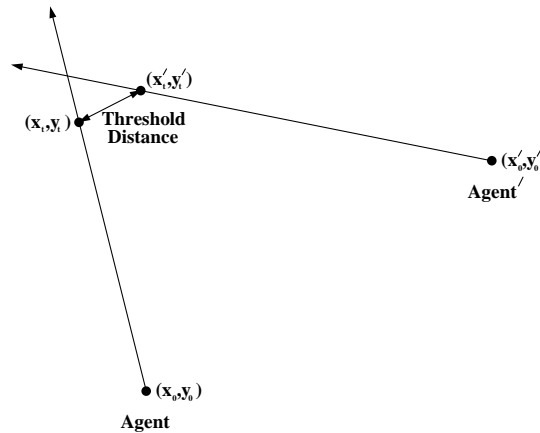


Figure 8.8: Initial Conditions for Predicting an Intersection

Figure 8.8 shows the initial conditions. We want to solve for (x, y) and (x', y') , which are functions of time, t .

$$x = x_0 + \vec{V}_x t \quad (8.2)$$

$$y = y_0 + \vec{V}_y t \quad (8.3)$$

$$x' = x'_0 + \vec{V}'_x t \quad (8.4)$$

$$y' = y'_0 + \vec{V}'_y t \quad (8.5)$$

Using the distance formula:

$$\sqrt{\Delta x^2 + \Delta y^2} = \text{Distance}$$

We want to solve for the time when the agent and the target come within the threshold distance, T , of each other:

$$\sqrt{(x' - x)^2 + (y' - y)^2} = T \quad (8.6)$$

$$(x' - x)^2 + (y' - y)^2 = T^2 \quad (8.7)$$

Substituting from Equations 8.2 through 8.5 we get:

$$\left((x'_0 + \vec{V}'_x t) - (x_0 + \vec{V}_x t) \right)^2 + \left((y'_0 + \vec{V}'_y t) - (y_0 + \vec{V}_y t) \right)^2 = T^2 \quad (8.8)$$

Substituting again using the following:

$$\alpha = x'_0 - x_0 \quad (8.9)$$

$$\beta = y'_0 - y_0 \quad (8.10)$$

$$\gamma = \vec{V}'_x - \vec{V}_x \quad (8.11)$$

$$\delta = \vec{V}'_y - \vec{V}_y \quad (8.12)$$

We get:

$$(\alpha + \gamma t)^2 + (\beta + \delta t)^2 = T^2 \quad (8.13)$$

Finally, if we substitute:

$$a = \gamma^2 + \delta^2 \quad (8.14)$$

$$b = 2(\alpha\gamma + \beta\delta) \quad (8.15)$$

$$c = \alpha^2 + \beta^2 - T^2 \quad (8.16)$$

We get a quadratic equation:

$$at^2 + bt + c = 0$$

With solutions:

$$t = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a} \quad (8.17)$$

If the discriminant ($b^2 - 4ac$) is negative then there is no solution. The agents will never come within the threshold distance of each other. In this case we calculate the minimum distance they will achieve and use this value as the threshold distance as we solve the problem again. Thus we calculate the time it will take the agents to achieve their minimum distance from each other. If the discriminant is positive (or zero) then we solve for the two values of t . When both values are negative, the agents are diverging, so we revert to traditional attraction, reporting a stress value based on distance. When one value is negative and the other is positive (or when both are zero) the agent has achieved the goal; the agents are within the threshold distance of each other and the stress is zero. When both values are positive, stress is proportional to the minimum of the two values.

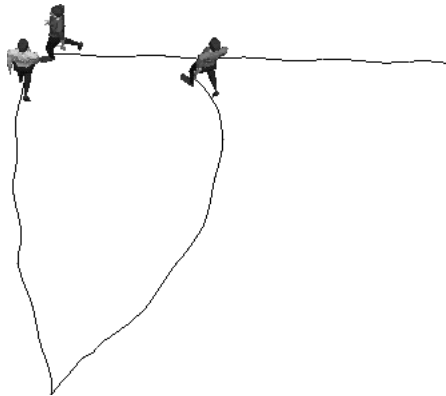


Figure 8.9: The Second Attempt at Predictive Attraction

Figure 8.9 shows that the final implementation yields a realistic predictive attraction behavior. Figure 8.10 compares all three approaches with the final implementation on the left and the traditional approach on the right. Note that if the agent being chased were to turn, speed up, slow

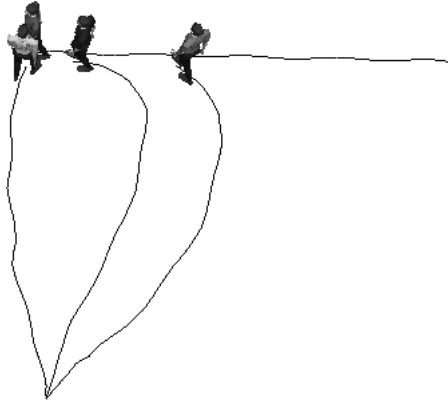


Figure 8.10: A Visual Comparison of All Three Attraction Behaviors

down, or attempt to evade it, the chasing agent would successfully adapt and adjust its pursuit because of the recalculation that leads to each footstep (a side-effect of the sense-control-act loop architecture).

8.2.4 Avoidance

We introduced anticipation to avoidance behaviors in two ways. First we did so by making adjustments to the shape of the proximity sensor's region of sensitivity. Then, driven by the need for more realistic predictive avoidance, we took an approach similar to that of predictive attraction where the agent reacts to objects' expected locations one or more steps in the future.

Proximity Sensor Adjustments

An object avoidance behavior typically combines a proximity sensor and an avoid control behavior. Determining the radius (R) and field-of-view (FOV) parameters when using proximity sensors is difficult. Slight variations of these parameters often have dramatic effects.

Figure 8.11 shows the path of an agent using a proximity sensor of the shape shown combined with an avoidance control behavior. Although the agent successfully arrives at the goal and avoids the obstacles, the path was inefficient and unrealistic. Figure 8.12 shows the path of the same agent using a proximity sensor with a longer, narrower region of sensitivity. This path is better; the agent meandered less.

Increasing R increases the amount of time the agent has to react to an obstacle. When R is

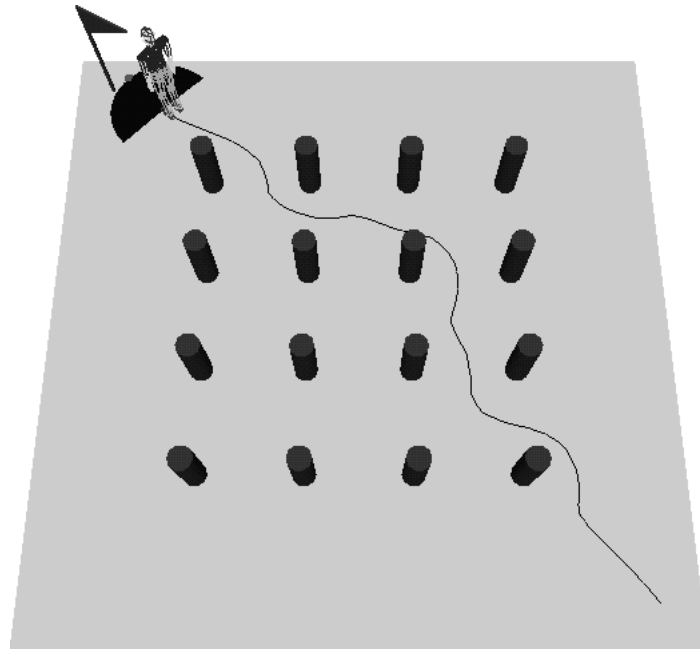


Figure 8.11: The Results of a Short, Wide Avoidance Sensor

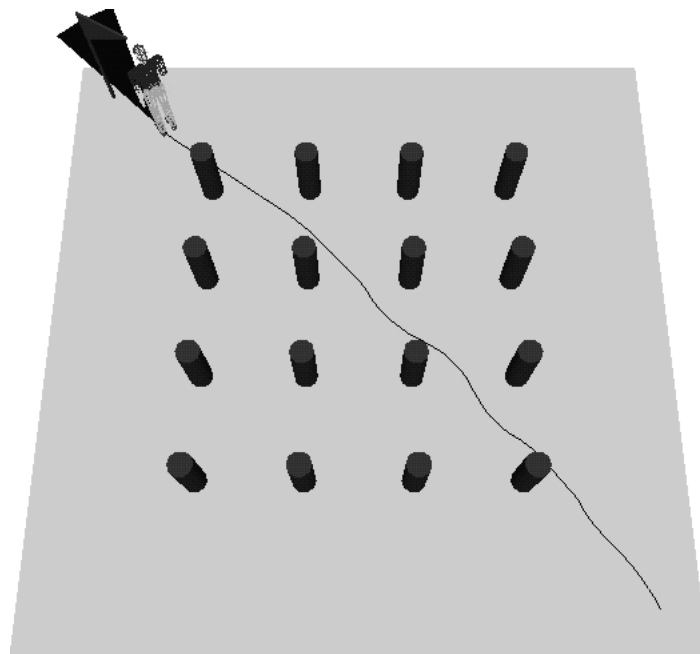


Figure 8.12: The Results of a Long, Narrow Avoidance Sensor

small, an agent will walk up to an obstacle, turning to avoid it only when very near. This behavior is realistic only when the agent's perception is limited by atmospheric conditions such as fog or smoke, or due to insufficient lighting. Increasing R to compensate often causes other problems: if a moving obstacle crosses the agent's path, far enough ahead of the agent so that there is no danger of collision, the agent should not turn to avoid the obstacle. While this approach of using long, narrow regions of sensitivity in proximity sensors is an improvement over basic object avoidance and exhibits features suggested by Reynolds [130], it still results in unrealistic behavior at times motivating the development of predictive, anticipating avoidance.

Predictive

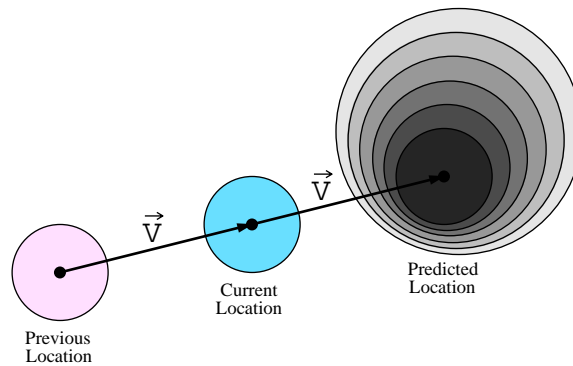


Figure 8.13: The Mathematics of a Predictive Avoidance Behavior

Predictive avoidance outputs a stress value proportional to the probability that the agent will collide with an object during the current step. First, the probability distribution is calculated. Figure 8.13 shows the probable location of a moving object (or agent). The darkest regions are regions of highest probability. The probability distribution is determined using bias values to approximate the likelihood of the object accelerating, decelerating, turning left, or turning right. Bias values can be approximated through observation or experience or acquired through learning algorithms. The stress value is proportional to the agent's point of intersection of highest probability.

Further refinement of the predictive avoidance behavior involves lookahead where the stresses associated with two or more predicted steps are summed (with decreasing weights) similar to the lookahead exhibited by terrain awareness. The behavior's output is the weighted sum of the stresses.

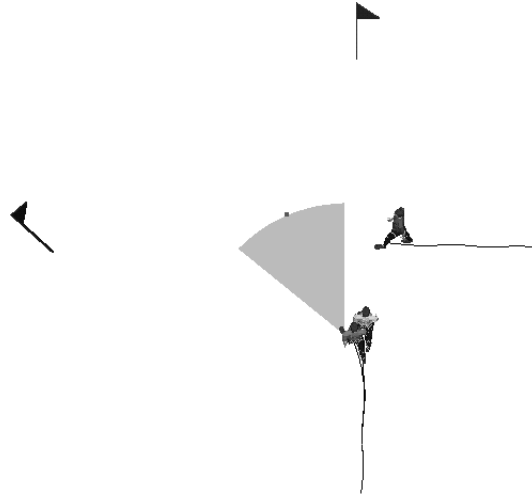


Figure 8.14: A Comparison of Traditional and Predictive Avoidance Behaviors: The Start of the Simulation

We give an example in which three agents must cross to the opposite flags. The agent on the right walks straight across through the use of an attraction behavior. The two agents starting on the bottom, in the same location, are attracted to the goal on the top, but also avoid the agent moving left, one using traditional proximity-sensor-based avoidance, the other using predictive avoidance. The shape of the proximity sensor's region of sensitivity is illustrated. Figure 8.14 shows the avoiding agents beginning to diverge. The traditional agent must swerve left to keep the avoidance target out of the proximity sensor's region of sensitivity while the anticipating agent, predicting that no collision will occur, continues in a straight line.

Figure 8.15 shows the traditional agent finally able to cut behind the avoidance target after being "pushed" significantly out of its way. Figure 8.16 confirms that anticipation results in a shorter, more realistic path.

A target's predicted behavior may not occur; it may stop moving suddenly and the agent may run into it. To prevent this from happening, in addition to the predictive avoidance behavior we use a passive avoidance behavior to halt forward motion if an object is immediately in front of the agent. According to Anderson and Donath [7] this is equivalent to the freeze or startle behavior observed in animals. Although attempts at preventing collisions may fail, Wilhelms and Skinner point out that "Collision avoidance in real life is quite flawed; one need only consider highway traffic and people in a crowded hallway. It is desirable, however, that objects do not unrealistically

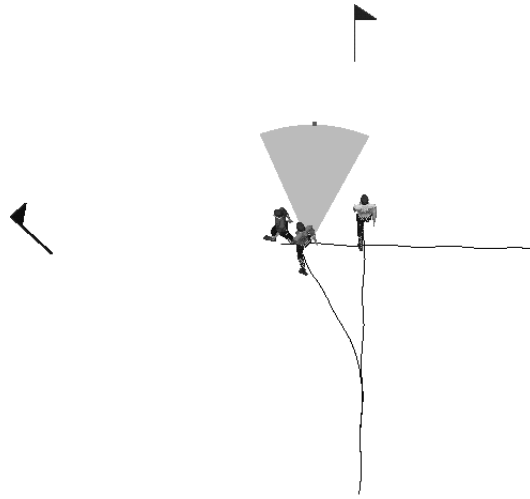


Figure 8.15: The Middle of the Simulation

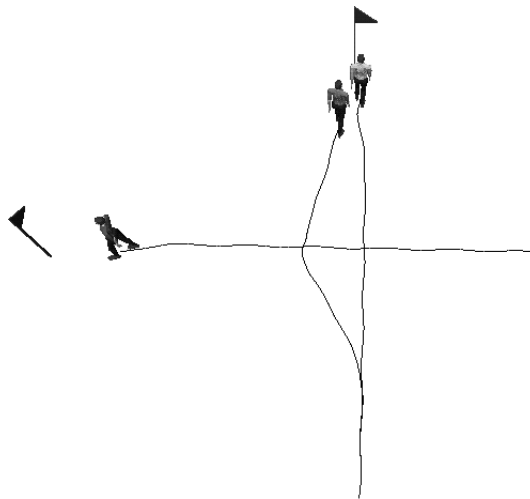


Figure 8.16: The End of the Simulation

interpenetrate when collision avoidance strategies fail.” We added collision *prevention* to the locomotion engine for this reason.

Chasing

Anticipation at the state machine level can be seen in our implementation of the chasing behavior. When an agent arrives at the target’s last known location to find that it is not visible, instead of exiting unsuccessfully, the agent predicts where the target may have gone based on its last known velocity. It extrapolates the target’s path and explores in that direction.

8.2.5 Planner-Based

Although a planner is not a standard part of the locomotion control system, one can be “plugged in” just as easily as a user can configure the system. The Hide and Seek project [149] exemplified this capability by combining a planner with LCS to simulate games of hide and seek. The planner exhibited anticipation when controlling the **seeker** in trying to determine what the **hidiers** would do next. In doing so it was able to make more reasonable decisions for the seeker. While anticipation at the state-machine level is generally constrained to state-based reasoning, anticipation at the planner level can make more complex inferences and can more accurately reflect complexities and subtleties of animal intelligence.

Chapter 9

Results

In this chapter we describe the human locomotion control system. We discuss its adherence to the system requirements previously laid out, demonstrate the system through a fully worked example, and analyze its performance.

9.1 Description

The locomotion control machine (LCM), diagrammed in Figures 9.1 and 9.2, embodies the locomotion control system. Given an agent and a goal, LCM combines avoidance, ducking, inertia, attraction, turning, path following, and chasing to guide the agent to the goal.

Figure 9.1 is an abstract state diagram of the entire system's functionality. Path following is only used when the goal is not immediately visible, and then only until the goal becomes visible (or until the agent arrives). Otherwise, the agent chases the goal, exiting when it either arrives at or loses the goal. The system also includes an agent model for parameter adjustment and a monitor for condition evaluation.

Figure 9.2 is a more detailed flow chart of the locomotion control machine. The machine exits immediately if the agent is at the goal. Otherwise, level 0 behaviors bind to the agent and LCM performs a visibility test to determine whether the agent can see the goal. If not, LCM instantiates a path-following machine to calculate and guide the agent along a path to the goal. If the path-following machine reports that either no such path exists, or that the agent arrived at the goal, level 0 behaviors unbind and LCM exits.

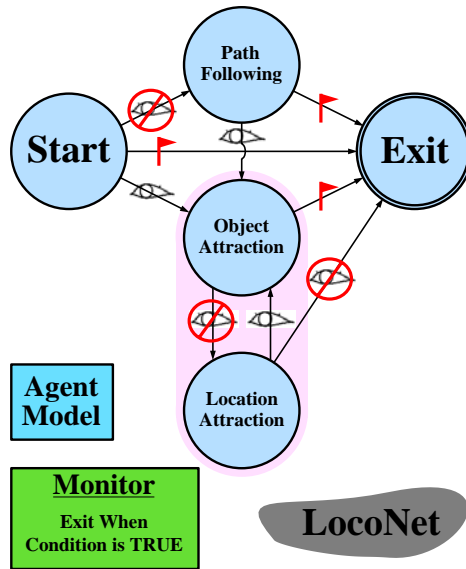


Figure 9.1: The Locomotion Control System

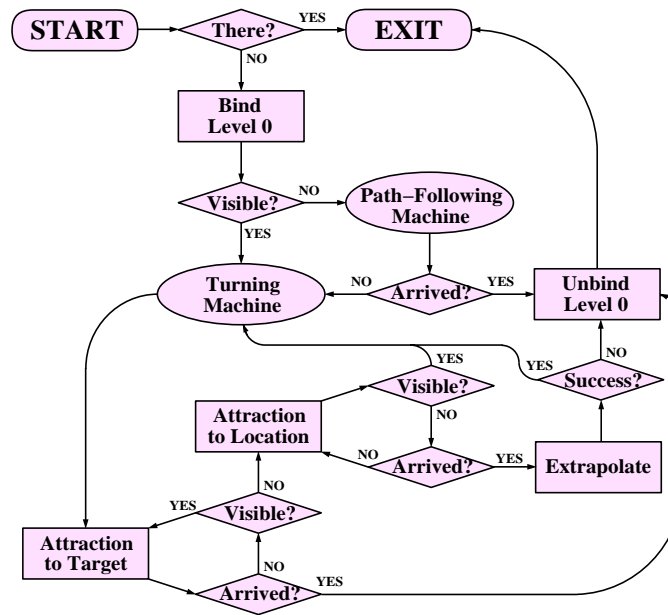


Figure 9.2: The Locomotion Control Machine

Instantiation of a turning machine begins LCM's "chasing" phase and is entered if the goal is immediately visible, or if the agent, nearing the end of a path, finds the goal has moved (but is visible). The agent turns to face the goal, and an attraction to the goal binds to the agent. As long as the goal is visible the agent will walk toward the goal. Upon arrival, level 0 behaviors unbind and LCM exits.

If the goal becomes occluded, the object attraction unbinds and an attraction to the goal's last known location binds in its place. The agent walks toward this location until either the goal becomes visible again or it arrives at the location. In the former case the location attraction unbinds and LCM transitions back to the turning machine state where the chase continues. In the latter case the location attraction unbinds and the agent walks in the direction the goal was last known to be heading. If the goal becomes visible the chase continues. If not, level 0 behaviors unbind and LCM exits.

9.2 Adherence to System Requirements

The locomotion control system adheres to the system requirements laid out in Chapter 5.

Aware: The agent is made aware of its environment through the use of a network of simulated sensors. LCS's S-C-A loop includes relatively little computation and is therefore fast, so the agent remains aware of its environment throughout the simulation.

Dynamic: A major advantage of behavioral S-C-A-based architectures over planner-based architectures is that they are able to cope with a dynamic environment in a timely fashion. Because the environment is resampled each time through the loop, the agent is made aware of, and can react to, changes in the world.

Easy to Use: Figure 9.3 illustrates the system architecture with the addition of a high-level user interface (Figure 9.4). Through a combination of input boxes, sliders, buttons, and menu items, the user configures the simulation, often in a matter of minutes. Knowledge of the underlying architecture or implementation is unnecessary. Most simulations only utilize a small fraction of the system's capabilities, so reasonable defaults help to speed up the process. Also, the system includes a "verbose" mode where detailed debugging information

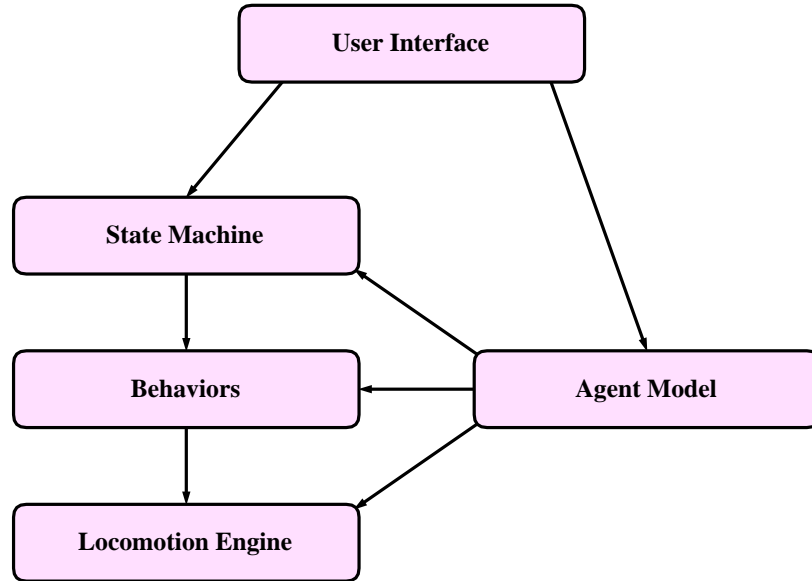


Figure 9.3: The Architecture and the Interface

is displayed to aid the user in reconfiguring the simulation if something goes wrong or if he or she is unhappy with the resulting animation.

Efficient: We satisfied the efficiency requirement by choosing a behavioral control strategy, by keeping the behaviors simple with minimal computation, and by exploiting efficiencies whenever possible. For example, the interface provides a “stationary” checkbox which, when checked, allows the system to take advantage of the fact that the goal, albeit a moveable object, will be stationary for the duration of the simulation. When it is known that the goal is stationary, in general, less computation is required to guide the agent to that goal. What we refer to as stationary objects conform to Sloman’s notion of *passive* objects such as trees or poles as opposed to *active* objects such as other agents [140].

Extensible: LCS is extensible at every level. It supports the addition of new sensors and behaviors of all types as well as new personality attributes and state information. In this way it is useful both as an animation tool and as a research tool.

Fault Tolerant: Arbitrary condition evaluation and action execution provides the agent with the ability to anticipate and prepare for contingencies, making the system fault tolerant. The user may supply any number of condition-action pairs. The conditions are constantly evaluated

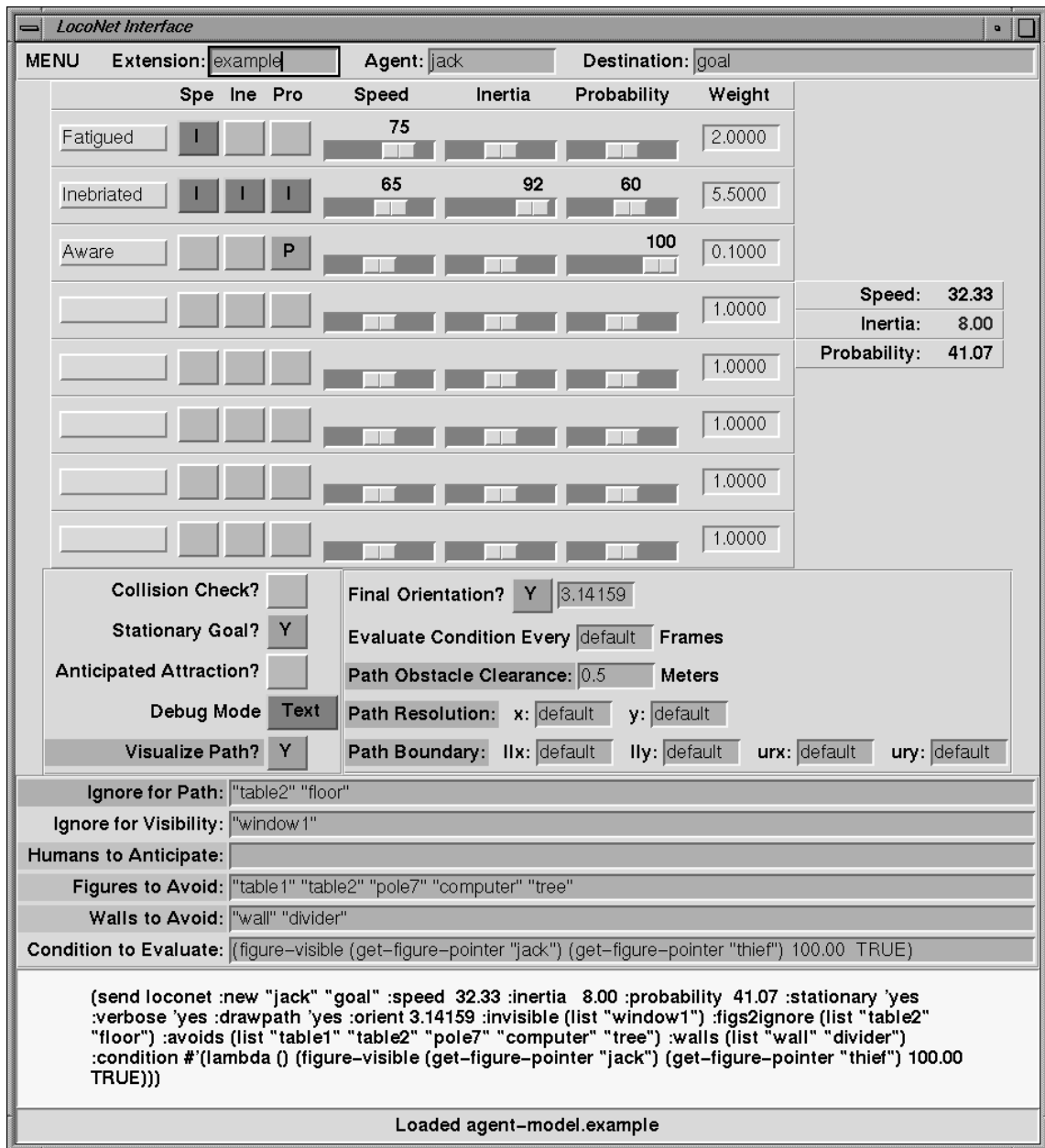


Figure 9.4: The User Interface

in parallel by a monitor process. If one of the conditions is achieved, the simulation is interrupted, the corresponding action is executed, and then the simulation continues. The conditions are prioritized, so if more than one is achieved simultaneously, the order in which the actions are executed is deterministic. Moreover, each condition is rechecked before its action is executed in case it was obviated by an earlier action.

Flexible: Flexibility is achieved through the combination of a behavioral level with a state machine level. In this way we support potential-field-based reasoning as well as state-based reasoning. The combination is powerful enough to simulate a wide range of behaviors. When more complex reasoning is required, a planner or other higher-level module can perform this reasoning and interface with the system as required through a LISP interface with all the power of the interactive user interface.

Modular: LCS was designed to be modular, to support adding, removing, and replacing pieces. Vision can be seen as an example of this. LCS required simulating vision through ray casting. Although the existing method of ray casting is sufficient for our simulations, the vision module can be replaced easily when a different vision simulation method is desired. The same is true of other parts of the system such as the path-following path-generation module and the chasing extrapolation algorithm. They too can be replaced in order to experiment with other cognitive theories.

Opportunistic: Arbitrary condition evaluation provides the agent with the ability to take advantage of opportunities. The approach incorporates cognizant failure in that the user anticipates things that might go wrong or opportunities that may arise (things that may go right) and plans for them. This is implemented as a set of condition-action pairs passed as an optional parameter. A monitor continually evaluates each of these conditions. When one is achieved, the system executes the associated action. For example, as an agent follows a path that takes it through a closed door, a condition can be checked that is achieved when the agent gets to within reaching distance of the door. The action passed with this condition can cause the agent to reach out and open the door as it passes through the doorway, releasing the door when it is through. Condition evaluation can also be used to dislodge an agent that is stuck in a local minimum, a common problem in behavioral control systems.

Parameterizable: LCS includes parametric control over the agent's speed, inertia, and the probability it will react to an opportunity (effectively its awareness).

Purposeful: Our agents are goal-based, their goals being to locomote to chosen locations, objects, or human figures. An agent's purpose is to choose an appropriate path through the environment and an appropriate locomotion style that properly reflects the attributes specified in the agent model.

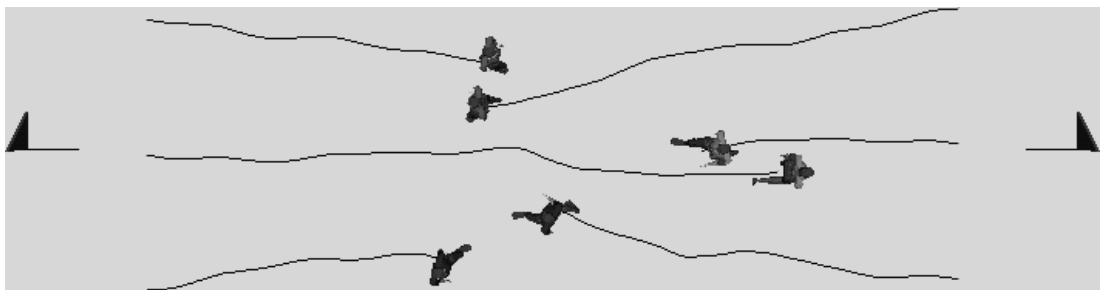
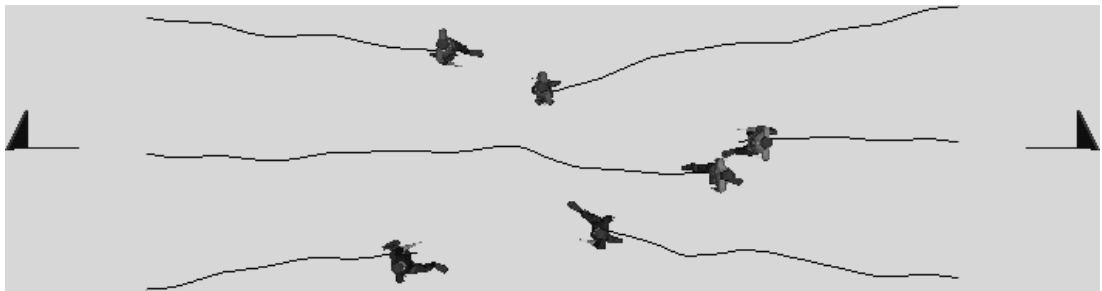
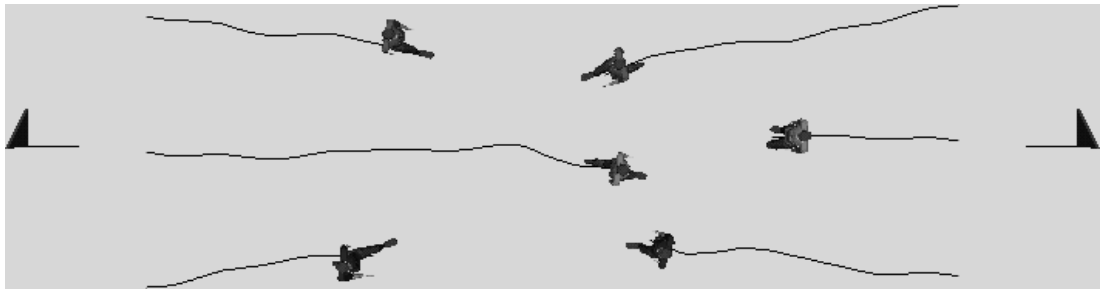
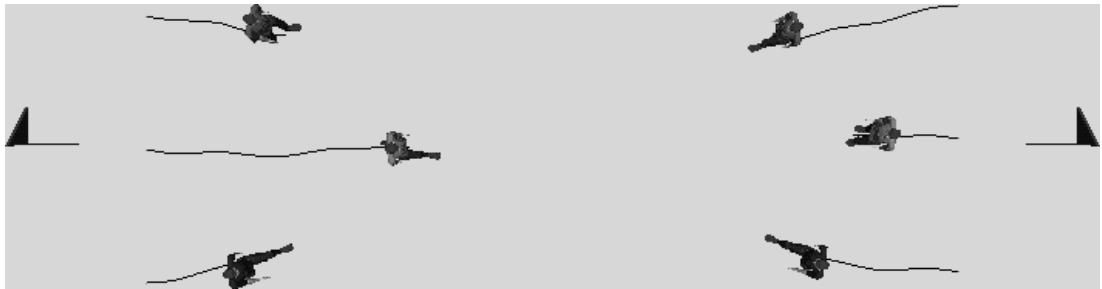
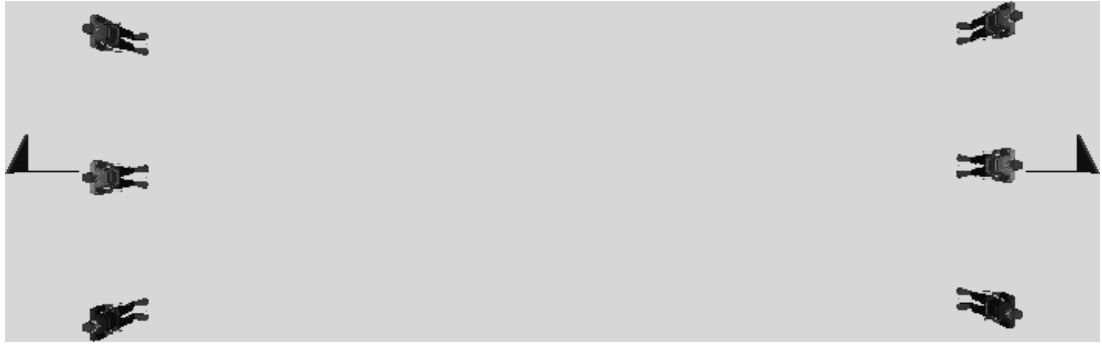
Realistic: *Jack's* accurate human models combine with Ko's locomotion control system [82, 83, 84] to produce extremely realistic looking simulations. Additionally, our strict enforcement of the limited perception policy and introduction of anticipation throughout the system serve to enhance this realism.

Robust: Robustness is achieved through generalization. We try to design sensors that detect an entire class of object rather than just one specific object whenever it is reasonable to do so. For example, an agent can be told to avoid all humans in the environment. If humans are added or removed, our agent continues to behave properly, avoiding any that are nearby.

9.3 Examples

We demonstrate the complete system at work with two examples. The first involves six agents in two groups of three, located at either side of the environment (See the figures on the next two pages). Each agent's goal is the flag on the opposite side of the field, requiring only two behaviors to achieve: basic attraction to the goal, and predictive avoidance of each of the other five agents. In addition, to avoid symmetry, each agent's **rushed** value was adjusted to be a unique multiple of five in the range [60, 85]. The 10 images show the agents successfully navigating to their goals without colliding with each other.

The second example of the complete system involves three agents. The agents exhibit attraction, avoidance, chasing, path-following, and turning behaviors and demonstrate system features such as limited perception, anticipation, and the use of the agent model. Figures 9.5a through 9.5h are eight frames from the simulation.



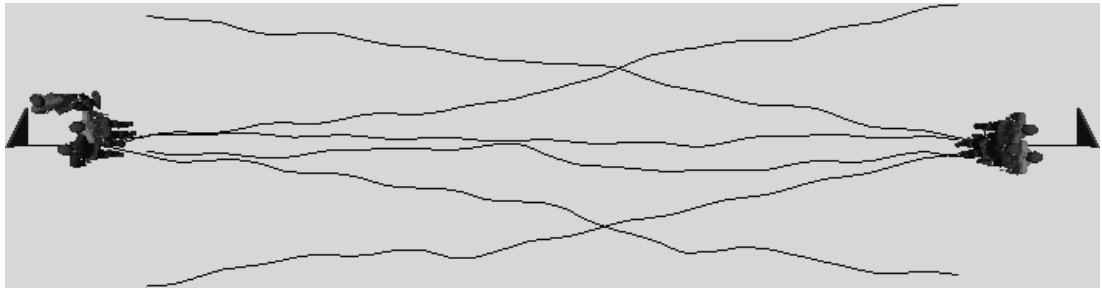
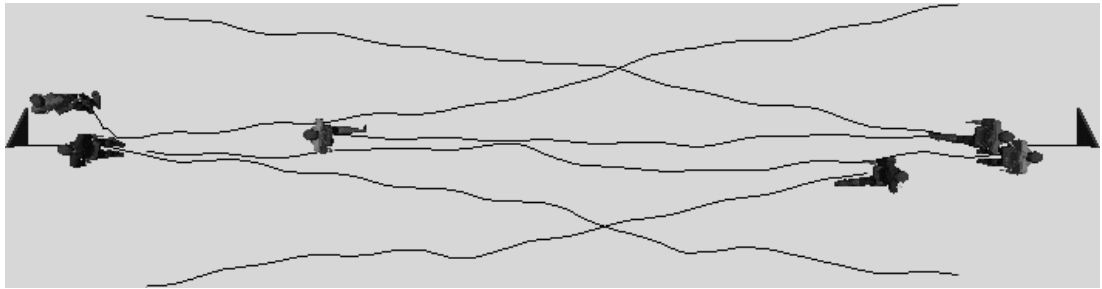
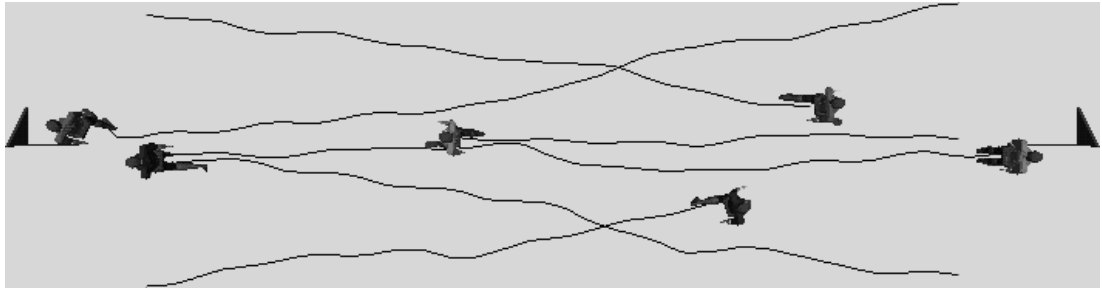
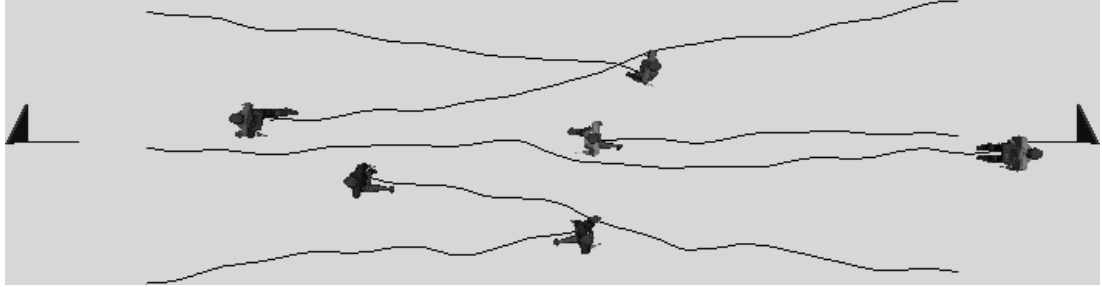
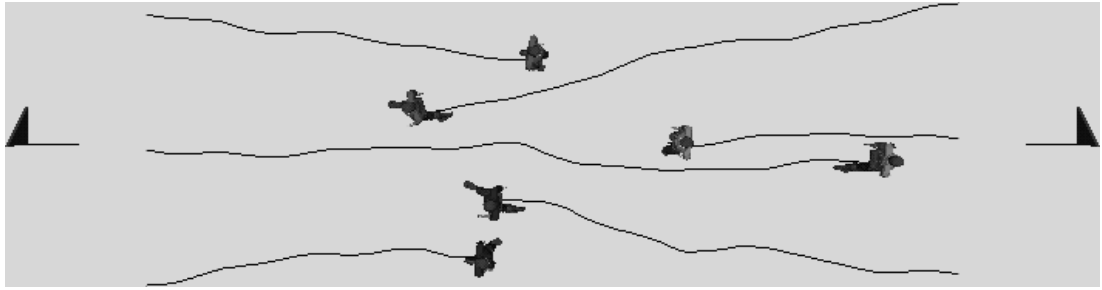


Figure 9.5a: Our simulation environment is made up of two cylindrical obstacles, two cubical obstacles, a flag, grassy terrain, and three human agents. From left to right, we refer to the agents as one, two, and three. All three agents are configured to avoid the cylinders and boxes using standard avoidance techniques and to avoid each other using predictive avoidance (anticipation). If path-following becomes necessary, all three agents will plan a path that avoids only the boxes, *i.e.*, the cylinders, humans, flag, and terrain are not to be considered obstacles for the purposes of path planning. Finally, all three systems are told to ignore cylinders during visibility tests. The cylinders are treated as simplifications of other objects through which the agents would be able to see.

Agent one's goal is agent three. Agent two's goal is a location to the right of the two boxes. Upon arrival at the goal, agent two is to turn to face west (left). Agent three's goal is the flag which is known to be stationary thus allowing the system to simplify its calculations and reasoning. Furthermore, agent three's rushed attribute is set to a relatively high value, so it will walk more quickly than the other agents.

Figure 9.5b: Agents one and two both begin to walk toward their goals. Agent three plans a path to the flag and begins to follow it. Note that the path avoids the boxes but not the cylinders, as expected.

Figure 9.5c: Agent one, having lost visual contact with its goal, is now walking toward agent three's last known location. Agents two and three are avoiding a collision with each other with predictive avoidance. Agent two slows down to let agent three pass by and agent three swerves slightly to its left in an attempt to avoid the collision.

Figure 9.5d: Agent one can now see its goal again, so it switches from last-known-location attraction back to object attraction and again heads toward agent three. Agent two continues toward its goal as does agent three by returning to the path.

Figure 9.5e: Agents one and three approach obstacles and begin to swerve around them. Note that agent one is still attracted to agent three even though the cylinder obscures agent three thanks to our initial instructions to the system. Agent two approaches its goal.

Figure 9.5f: Agents one and three circumnavigate the cylinders while agent two arrives at its goal.

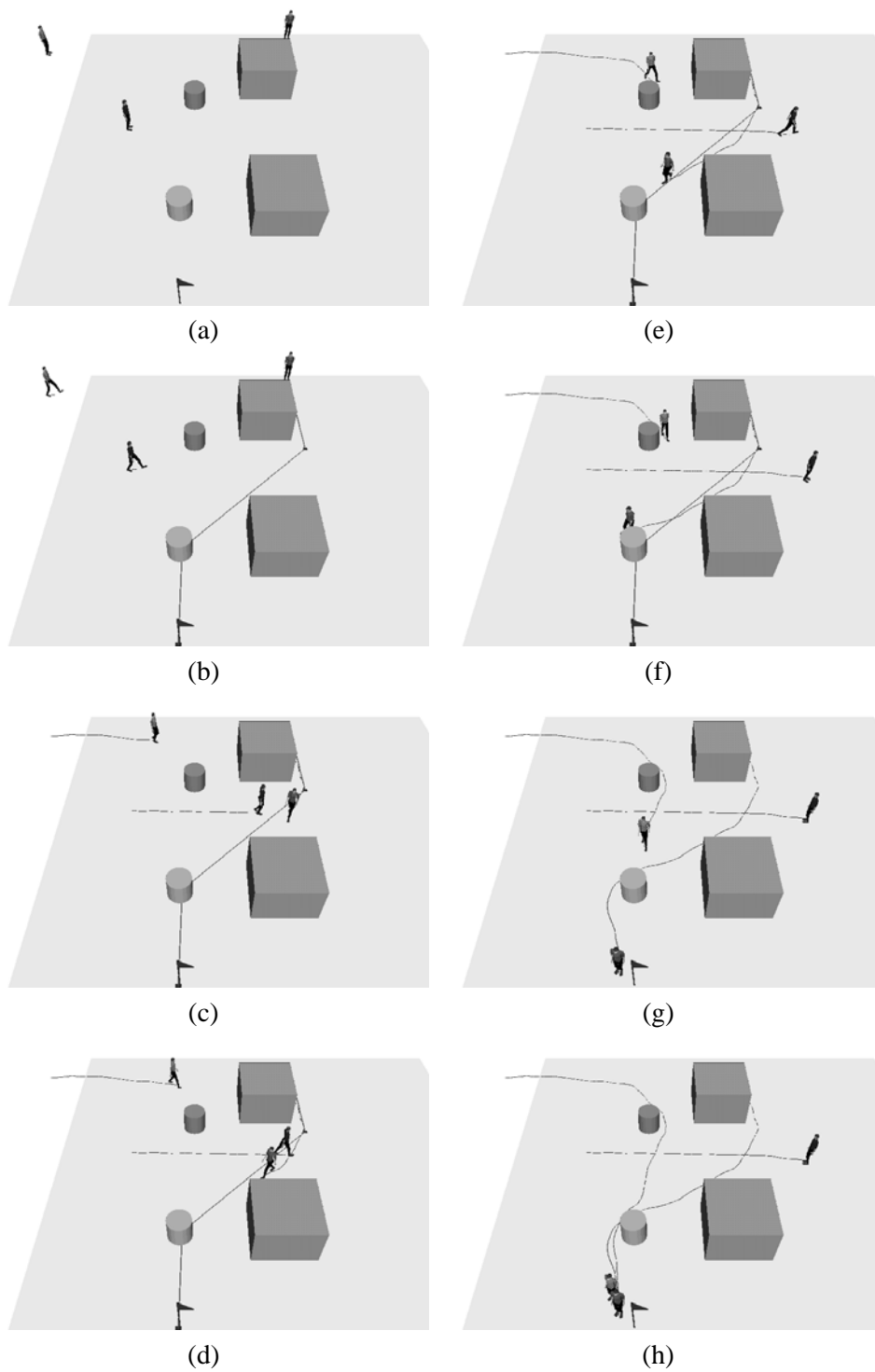


Figure 9.5: An Example

Figure 9.5g: Agent one continues chasing agent three. Agent two completes its turn and is now facing left. Agent three arrives at its goal.

Figure 9.5h: All three agents have arrived at their goals. The simulation ends.

Once the environment was built, it took us less than one minute to configure each agent’s animation using the locomotion control system user interface described in Section 9.2 (Figure 9.4). The simulation then ran in real time on a Silicon Graphics Reality Engine II with a 250 megahertz R4400 processor. The total elapsed time for the simulation was 55 seconds. The total elapsed time from the start of configuration to the end of the simulation, including loading and saving files and other overhead, was under five minutes. These times are typical for our simulations.

9.4 Performance Analysis

This section contains time complexity analyses of various parts of the system which can be used to determine the incremental cost of adding agents, obstacles, and behaviors to a simulation. In each of the following tables, a is the number of agents in a simulation, n is the number of obstacles, f is the number of potential next footsteps for an agent, b is the number of behaviors bound to an agent, and s is the number of steps of lookahead used by the predictive avoidance behavior.

Sensor	Time Complexity	Weak Link
Object	$O(1)$	
Line	$O(1)$	
Location	$O(1)$	
Proximity	$O(n)$	*

Figure 9.6: Sensor Time Complexity Analysis

Figure 9.6 shows the time complexity analysis of the system’s sensors. Object, line, and location sensors each probe and perform calculations on only a single part of the environment; therefore, they operate in constant time. An object sensor, for example, will add the same overhead to a system with 10 obstacles as it will for one with 10,000. A proximity sensor, however, performs calculations proportional to the number of objects it currently detects. Note that although a proximity sensor can theoretically be used to detect every obstacle in the environment, hence

the $O(n)$ time complexity, in practice, one is typically used to detect only a small subset of the obstacles.

Control Behavior	Time Complexity	Weak Link
Attract	$O(1)$	*
Avoid	$O(1)$	*

Figure 9.7: Control Behavior Time Complexity Analysis

Attract and avoid control behaviors are fixed equations based on constants, thus they operate in constant time (Figure 9.7), so when combining sensors and control behaviors into behaviors, sensors are the weak link.

Behaviors	Time Complexity	Weak Link
Object Attraction	$O(1)$	
Location Attraction	$O(1)$	
Predictive Attraction	$O(1)$	
Line Avoidance	$O(1)$	
Proximity Avoidance	$O(n)$	*
Predictive Avoidance	$O(s)$	*
Inertia	$O(1)$	

Figure 9.8: Behavior Time Complexity Analysis

Figure 9.8 shows the results of combining basic sensors with control behaviors and also analyzes three more complex behaviors. Object attraction, location attraction, and line avoidance each combine a constant-time sensor with a constant-time control behavior, resulting in a constant-time behavior. Predictive attraction and inertia operate in constant time as well. Proximity avoidance combines a linear-time sensor with a constant-time control behavior, and so is linear time itself. Predictive avoidance, although based on one agent, involves calculations that are performed once for each of the s steps of lookahead. Its time complexity is therefore $O(s)$.

We can now analyze the time complexity of the overall simulation in terms of agents, footsteps, behaviors, obstacles, and lookahead (Figure 9.9). Since each agent has associated with it an instance of LCS (which includes an instance of the locomotion engine), every phase includes the $O(a)$ term. The simulation run-time is linear relative to the number of agents.

Locomotion Engine Phase	Time Complexity	Weak Link
Footstep Generation	$O(af)$	
Sense	$O(afb(n + s))$	*
Control	$O(afb)$	
Act	$O(a)$	

Figure 9.9: Simulation Time Complexity Analysis

Footstep generation is clearly proportional to the number of footsteps. Control requires computing the sum of all the stresses output by the behaviors. One stress is output by each behavior, for each possible next footstep of each agent. Once an action has been selected for an agent, corresponding to the new state of minimal stress, that action can be carried out in constant time.

Our analysis shows that sensing is the system's weak link. It is linear in the number of agents, footsteps, behaviors, and the maximum of the number of obstacles and steps of lookahead. We observe that simulations can be parallelized, executing one LCS on each processor of a multiprocessor machine. The effect on simulation time complexity is to reduce all $O(a)$ terms to $O(1)$. Given a machine with enough processors, it is actually possible to parallelize any or all of these calculations, reducing each one to constant time complexity. It should be noted that if this were the case, data collection among processors would add at least $O(\log n)$ to the time complexity of each parallelized calculation. We suggest experimenting with parallelization as future research.

Part IV

Conclusion

Chapter 10

Discussion

The current trend in designing autonomous agents is away from the *deliberative thinking* paradigm and toward a more direct coupling of perception to action. Flexibility, distributivity, parallelization, and dynamic interaction with the environment are emphasized more and more [94]. An approach that exhibits these features, behavioral control of animation or robot locomotion has become increasingly popular over the last decade.

Locomotion reasoning through behavioral control has several desirable properties. The iterative approach allows for a dynamically changing environment which includes changing terrain, obstacle, and goal locations. Since only a few local decisions need to be made for each step, the technique is fast and often implementable in real-time. Moreover, it is versatile. New types of behaviors may be designed and easily integrated; additional layers of behavioral control may be overlaid on the architecture. This, combined with its modularity and extensibility, makes it a useful cognitive research tool.

Purely reactive systems are limited; realistic tasks require a certain amount of stored information. State machines provide an additional layer of control and are a mechanism for introducing locomotion reasoning and high-level decision-making into an architecture. They expand the agent's behavioral vocabulary by allowing for state-based and memory-reliant behaviors. Chasing illustrates this point, clearly requiring reasoning and decision-making beyond the scope of the S-C-A loop or pure potential field-based behaviors. Our locomotion control architecture allows for this reasoning.

We have presented a complete agent architectures and a design methodology based on a

theoretical formalism. This allows a researcher not only to understand the system from a theoretical point of view, but also to build the system with relative ease. We have addressed many of the concerns that are unique to human locomotion, though we have done so without loss of generality. Though only demonstrated for walking, our locomotion control system is designed to support the wide variety of locomotion styles exhibited by different types of animals or robots.

We have made the agent's locomotion variable and more interesting by basing it on an agent model. The agent model maps agent state and personality attributes to system parameters. Different attribute settings result in different locomotion styles. Two agents with the same goals might achieve their goals differently depending on their agent models.

Finally, we have enforced a limited perception policy and introduced anticipation to enhance the system's overall realism. The agent's access to the environmental database is filtered to prevent it from acquiring information it is not able to "perceive" visually. Furthermore the agent bases its actions not only on the current state of the world, but also on the predicted state of the world one or more steps into the future.

Chapter 11

Research Summary

The most merciful thing in the world, I think, is the inability of the human mind to correlate all its contents.

– *H. P. Lovecraft, The Call of Cthulu* –

This chapter summarizes the research we have completed and discusses its contributions to science and the field of behavioral control. We provide a complete, structured architecture for behavioral control of locomotion and the design methodology required to build behavioral locomotion systems. We explain the system requirements to which the architecture adheres, give a formal description of the locomotion engine, supply behavior selection and placement criteria, provide agent model design and integration specifications, and finally show how to put everything together in a state-based framework.

We validate the architecture by implementing a human locomotion control system giving detailed descriptions of our results. Simulations of fully-articulated walking human models in a three-dimensional environment serve as the testbed, rather than abstract objects, robots, or creatures that fly using jet propulsion. As a result, we address additional complexities specific to human locomotion.

Our research includes a variety of implementations of the following behaviors implemented either through the locomotion engine or the state-machine layer:

Various Basic Avoidances	Turning	Predictive Avoidance
Ducking	Path Following	Attraction to Deep Spaces
Inertia	Chasing	Field-of-View Avoidance
Various Basic Attractions	Predictive Attraction	Terrain Awareness

We discuss and enforce limited perception as a policy and we introduce the notion of anticipation or predictive sensing where a probabilistic model of future states of the environment influences locomotion decisions. Limited perception and anticipation combine to make simulations based on animal behavior models more realistic. The simulated agents only acquire and react to information that a real agent in an equivalent situation would be able to acquire. Furthermore, the simulated agents anticipate changes in the world such as the movement of other agents, making decisions based on this prediction rather than on the current state of the world alone. In this way, the simulated agents can more closely approximate behaviors observed in real animals.

We achieve the system requirements of Chapter 5, most notably the following. The architecture allows for arbitrary condition-based interruption making it opportunistic. We minimize the set of active behaviors to solve the behavior weighting problem, easing the burden on the system designer, and making the system easier to use. The design methodology provides for extending the behavioral set, the state machine, and the agent model. The agent model allows the user to modify agent state and personality attributes reflected in its observed behavior.

The system’s extensible and modular design make it a useful cognitive research tool. Modules can be swapped in and out and the system can be extended easily allowing for testing of cognitive theories. As an example, an independent user modified her system to include a model of visual attention. Before beginning to locomote, her agent turned its head and focused its gaze in the direction of the goal. Then, during locomotion, its gaze wandered based on attractions and avoidance, among other factors.

The system employs a layered approach making it versatile. It allows for both behavioral and state-based reasoning, and even supports higher-level “plug-ins” such as a planner. In the Hide and Seek project, the system was controlled by a planner exclusively. There was no user interaction whatsoever. In addition, each of these types of behaviors are parameterizable and are affected by the agent model. The system parameters we use most often are: the agent’s speed, the agent’s inertia, the amount of anticipation, the condition being evaluated, and the probability the agent will notice and react to a certain condition being achieved. The attributes we use most often are:

fatigue, intoxication level, and awareness.

While other researchers have designed complete architectures, though with different features, they tend to gloss over or completely ignore the implementation details and issues required to make use of their architectures. Our biggest contribution to the field of behavioral control is the detailed design methodology we present. This thesis contains theoretical background information, a full architectural description, and a complete recipe for building a locomotion control system capable of the reasoning and decision-making necessary to realistically locomote a real or a simulated agent to an arbitrary goal.

Chapter 12

Future

Do you hear the people sing,
Say, do you hear the distant drums?
It is the future that they bring
When tomorrow comes.

– Les Misérables, Do You Hear the People Sing? –

Our research has motivated many possible avenues of future research. We list the more interesting possibilities here.

Audio cues: Sound might contribute to locomotion decisions. Although not visible, an agent might avoid another agent at a blind intersection based on the sound of the other agent approaching. This behavior has been observed in humans, particularly at blind intersections such as those found in buildings.

Behavioral Modifications: Sometimes a path which intersects obstacles is chosen for the agent. When this is the case the agent is forced to make behavioral changes to successfully navigate through the region. These modifications would be level 0 reflexive behaviors implemented as state machines running in parallel with the locomotion control system, similar to ducking. When one of these situations occurs, the corresponding state machine makes the appropriate change to the agent's posture or locomotion.

- **Low Obstacles:** When the agent encounters a low obstacle such as a boulder or a fallen tree, one of three things happens: the agent climbs over, jumps over, or steps on the obstacle depending on the size of the obstacle and the agent's current behavior. The agent only jumps if its current velocity is above a predefined threshold.
- **Narrow Widths:** When the agent encounters a narrow width it will squeeze through it. This may require turning sideways and side-stepping.

Better Sensors: The line sensor described in Section 7.3.3 would not be required and the object sensor described in Section 7.3.1 would be improved if a new sensor were implemented to return the distance between figure *geometries* rather than bounding volumes. If properly implemented, this new sensor might replace location and proximity sensors as well. In designing it, the greatest difficulty would be the increase in computational complexity, but a machine of sufficient speed or an algorithm of sufficient efficiency might overcome this difficulty.

Gait Variability: Although we do not address the issue of variable gait, our system provides the necessary flexibility to parameterize and control an agent's gait. The system state machine can be used to vary the parameters based on the agent's changing state or motivation. The agent model can be used to set the parameters based on the agent's personality in a manner consistent with Bruderlin and Calvert's model [35].

Group Locomotion Control System: With a few modifications the locomotion control system can be made to simulate group behavior. Whether a single LCS instance is to control a single agent or all the agents in the group, the implementation requires the addition of grouping behaviors. Compiling the results of the observations made by Hodgins and Brogan [73], Kube and Zhang [86], Reynolds [130], Shaw [136], and others, we suggest the following group behavior set: collision avoidance, group attraction, and velocity matching.

Collision Avoidance: Simply a non-interference behavior, collision avoidance prevents collisions between members of the group. This is best implemented as a level 0 instinctive behavior and should be combined with standard obstacle avoidance.

Group Attraction: The group members are attracted to the center of the flock, or they attempt to stay close to their nearby neighbors.

Velocity Matching: Each member of the group attempts to match the velocity of its neighbors. Hodgins and Brogan use an algorithm that computes a desired velocity for an individual based on the location and velocity of its visible neighbors.

When managing a group, the system has two additional responsibilities. It requires a mechanism for determining whether or not an agent is in the group, switching back and forth between an “in the group” and a “not in the group” state as appropriate. Group behaviors are only applied when an agent is in the “in the group” state. Additionally, some groups have leaders and followers. In such a group, the system only applies group behaviors to the followers. The leader or leaders are controlled by a standard locomotion control system; the rest of the group naturally follows.

Optical Flow Sensor: An optical flow sensor might be designed, combined with an avoid control behavior, and attached to a fast moving agent such as a bird. This optical flow behavior would prevent the agent from colliding with anything in its field-of-view by considering optical flow vectors.

Path Replanning: In the existing system, a path is generated for the agent which follows the path until either it reaches the end, or it is interrupted. The path is static; once generated it does not change. It may be more realistic to replan the path at times, but at which times? Experiments will indicate at which times it is appropriate to replan the path, *e.g.*, constantly, periodically, when the agent’s deviation from the path exceeds a threshold value, or when a particular condition is achieved. To be consistent with the limited perception policy, it may be necessary to assume the goal is in the same place unless the agent is made aware of a change.

Simulated Vision: The agent’s actual vision process can be simulated through the use of the z-buffering hardware. An approach similar to that of Boulic, Noser, and Thalmann’s [25], or Renault, Thalmann, and Thalmann’s [129] might prove successful where vision is the only channel of information between the agent and its environment. All sensors would be rewritten to use simulated vision only.

- **Infrared Sensor:** If simulated vision is used then night-vision could be simulated simply by replacing the visible light values with infrared radiation values.

Temporal Constraints: Parameters could vary with simulation time rather than remain constant.

The desire to achieve a goal may eventually override other desires, or an agent may suffer from increasing fatigue, for example. When the simulation is turned on, a floating point time value is passed. The default is zero. After each step a fixed number of seconds are added to the simulation time. Parameters may be constant or functions of a *time* variable.

Uncertainty in Object Sensing: Currently, an object sensor provides the exact location of the object it is sensing whether or not that object is actually visible to the agent. The following approach might be used to introduce uncertainty into the simulation and might be used by robotics researchers who simulate their robots before building them: When the simulation begins, a random point is chosen in the circle centered at the object, with radius equal to the uncertainty radius defined by the user. The agent “thinks” the object is at this point. After each simulation step (after each step taken by the agent) the perceived location of the object is moved by a random amount less than or equal to a user-defined distance, in a random direction. This distance might change depending on the number or type of reference points in the local environment. When wandering through an open field an agent’s perception of location or relative location will vary, but if the agent thinks the target object is near a tree visible in the distance, the agent will head straight for that tree. Once the object is visible to the agent (distance, size, and atmospheric effects should be considered) the perceived location is set to the exact location. Introducing uncertainty might allow for more realistic simulations or more realistic simulated robots.

Part V

Appendices

Appendix A

Additional Behaviors

In this appendix, we present our research on two behaviors that were omitted from our sample locomotion control system: attraction to deep spaces and field-of-view avoidance. We include them here because, although we found the existing behavioral set sufficient for our examples, these two additional behaviors are potentially useful in other domains.

A.1 Attraction to Deep Spaces

Agents running or flying through a crowded area, such as a dense forest, tend to move in the direction of greatest depth. A depth sensor combined with an attract control behavior simulates this behavior. A depth sensor detects the direction of greatest (weighted) depth similarly to Anderson and Donath’s open space attraction behavior [7] except that depth (rather than width) is the deciding factor.

A depth sensor uses the output of an array of range sensors covering the agents entire field of view (Figure A.1) to calculate the direction of greatest depth.

A range sensor (Figure A.2) takes the following parameters:

- Minimum Detection Threshold (D_{min})
- Maximum Detection Threshold (D_{max})
- Field-of-View Angle (θ)
- Orientation Relative to Agent (ϕ)

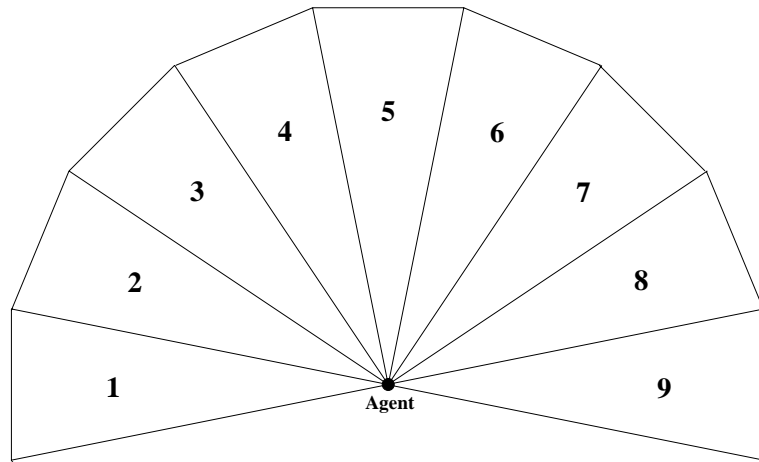


Figure A.1: A Depth Sensor

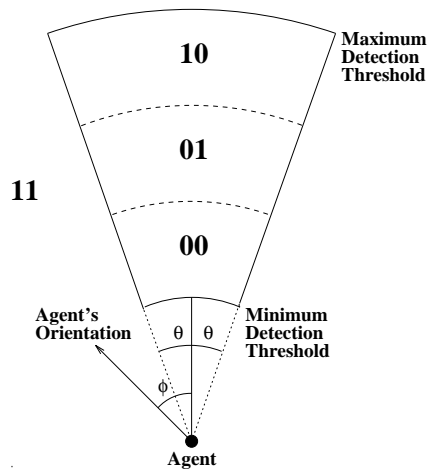


Figure A.2: A 2-Bit Range Sensor

- Resolution in Bits (n)

It detects objects that intersect the fan-shaped region defined by θ and the minimum and maximum detection threshold values. Similar to sonar, a range sensor's output is the distance to the nearest part of the nearest detected object.

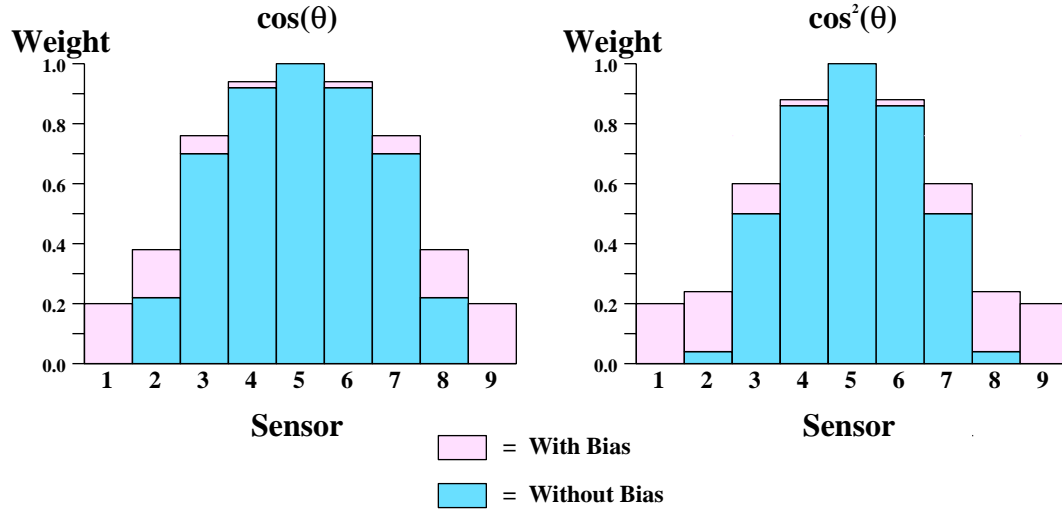


Figure A.3: Range Sensor Weighting Functions

If no object is detected then the code $2^n - 1$ is returned where n is the number of bits of resolution, the number of bits used in the return value. The return value is inversely proportional to the stress value of the range sensor. The stresses of all the range sensors are weighted by a cosine function, either $\cos(\theta)$ or $\cos^2(\theta)$, and then summed, yielding the overall stress value for the depth sensor. A bias insures that there is a minimum weight given to any sensor. In Figure A.3, the bias is 0.2.

Figure A.4 shows an agent working its way out of a simple maze. The depth attraction behavior is the only behavior bound to the agent. Eighteen range sensors, each with a 10-degree field-of-view ($\theta = 5$ deg), compose the depth sensor. The 32-bit range sensors, covering the agent's field-of-view as in Figure A.1, detect objects up to 10 meters away from the agent. In this simulation, the weighting function was \cos^2 and the bias was 0.

Attraction to deep spaces as a method of obstacle avoidance can be more realistic than proximity-sensor-based obstacle avoidance, especially when the agent is moving quickly through a region of densely populated obstacles. For example, an animal running through the woods to

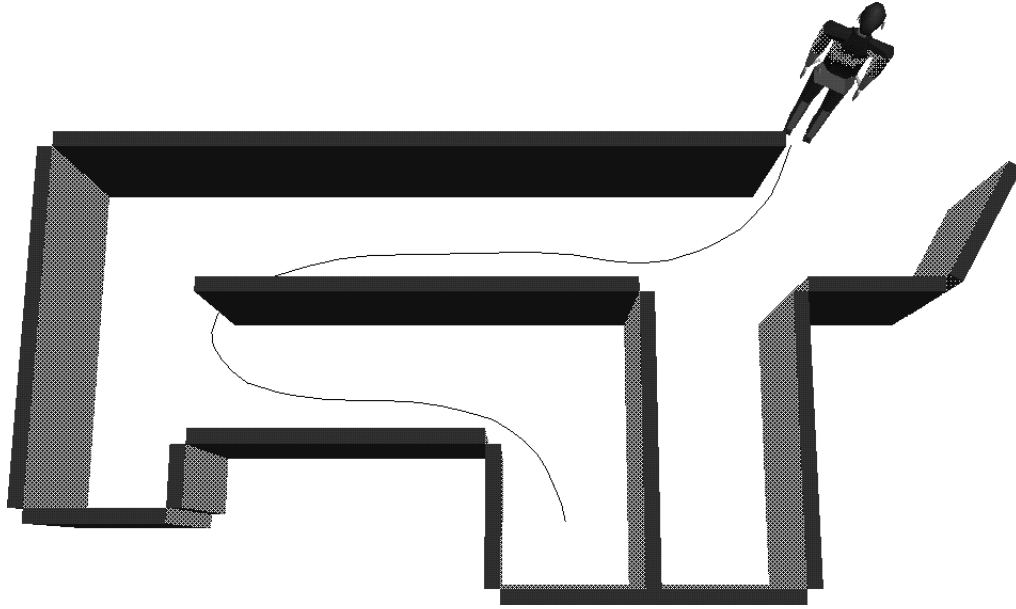


Figure A.4: An agent maneuvering through a maze using only a depth sensor for navigation

avoid a predator, or a human running through a crowd might benefit from this behavior.

A.2 Field-of-View Avoidance

Finding covered or concealed locations with respect to a known opponent or enemy locomotion has applications from military [91] to gaming [113]. In our implementation of this behavior, the agent attempts to achieve the goal while avoiding the gaze of hostile agents (hostiles). Hostiles' views may be obscured by obstacles, structures, or terrain, and attenuated by distance and atmospheric effects such as smoke or fog.

A field-of-view avoidance behavior uses a sensor to determine whether the agent is visible to any of a specified set of hostiles. Stress is proportional to the number of hostiles that can see the agent and inversely proportional to the distances to these agents.

The field-of-view sensor uses a simple vision model. Rays are cast from a point between a hostile's eyes toward the agent. If any of these rays hit the agent before intersecting the environment, the agent is visible to that hostile. Although casting more rays would improve the visibility test's accuracy, ray casting is a computationally expensive operation and the system would slow down significantly as a result. A better vision model might take advantage of z-buffering hardware to

simulate and analyze the hostile's view.

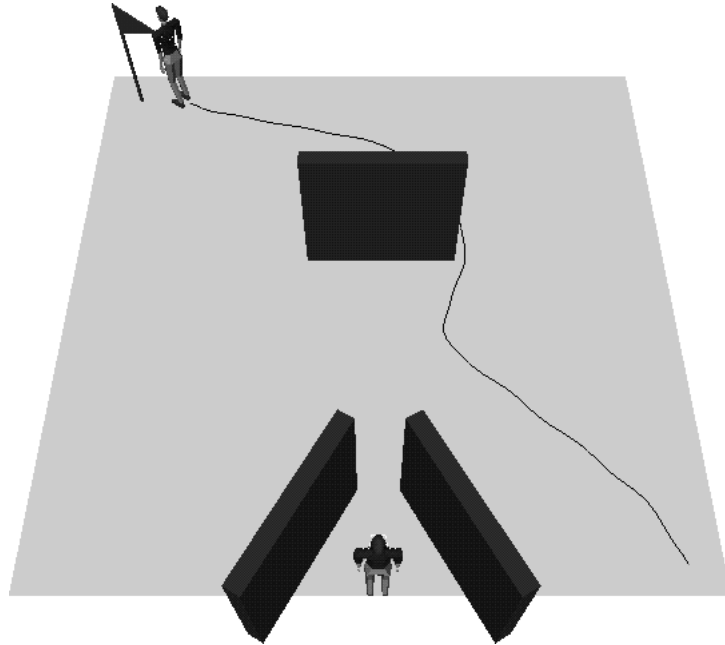


Figure A.5: Attraction, avoidance, and field-of-view avoidance combine to guide the agent to the goal without it being seen by the hostile agent hiding at the bottom

Figure A.5 shows an agent arriving at the goal, having avoided walls and the field-of-view of the stationary hostile at the bottom of the image. Although we were able to effectively implement this behavior, we omit it from our locomotion control system for one important reason. It violates the principle of limited perception. Even when an agent can guess where a hostile might be located in the environment, it is unrealistic for the agent to know that it will be visible to that hostile after taking a particular step. Upon analysis of this behavior we realized that a realistic implementation would have to involve the user or planner level. As an agent moves through the world it remains aware of its local environment through vision simulated by ray casting. When the agent “sees” a hostile, the system terminates the agent’s locomotion and sends a message to the calling system or user. A decision may then be made at that level as to the agent’s reaction.

Appendix B

Groups

...and the thousands of fishes moved as a huge beast, piercing the water. They appeared united, inexorably bound to a common fate. How comes this unity?

– *Anonymous, 17th Century* –

Generating reasonable agent behavior, whether simulated or real, is a difficult problem in itself. Generating reasonable group behavior is even more complex. It combines the single agent problem with the need to form and maintain a coherent group without relying on explicit communication. The goal of group behavior research is to have agents group, herd, flock, school, *etc.* without any form of centralized coordination.

B.1 Why Do Animals Group?

In nature, grouping is advantageous for predators. The group profits from a larger effective search pattern in the quest for food [135], and is able to hunt larger, more powerful animals than those the animals could otherwise overpower as individuals [27, 73]. When the prey attempts to evade the leading animal of the group by swerving when it is close behind, the animals further back are well placed to catch it [6].

Grouping is also advantageous for prey. It is safer in a group. Predators cannot easily surprise a large group. One animal detects the predator and alerts the group through its evasive maneuvers. Grouping benefits the average group member by limiting the average number of encounters with

predators [73, 153] statistically improving survival of the (shared) gene pool from attacks from predators [135]. Predators attack the nearest prey, usually at the edge of the group, so prey with a hereditary tendency to push to the center of any group will be favored by natural selection. In addition, there is a small energy savings as a result of *slipstreaming* or *drafting*, whether running, flying, or swimming. This savings may be critical in a race [6].

Being part of a group also improves the chance for social and mating activities [135].

B.2 Group Research

Natural groups consist of agents that exhibit two balanced, opposing behaviors: a desire to stay close to the group and a desire to avoid collisions within the group [27, 73, 135, 136]. Fish density in a school has been observed as being roughly uniform as though each fish had a sphere around its head with which it wished to contact the sphere of another fish [47]. Birds in a flock tend to exhibit three categories of awareness: themselves, their nearest neighbors, and the rest of the flock [123].

Many researchers have gone beyond the single-agent case in an attempt to model groups of robots or simulated animals, to **swarm intelligence**, such as Beni, Wang, and Hackwood [22, 23], Deneubourg, Theraulax, and Beckers [48], and Matarić [102]. Kube and Zhang use a common goal and non-interference behavior as a simple form of cooperation [86]. Their non-interference simply becomes robot-avoidance in their implementation. Steels describes a simulation of simple robots using the principles of self organization to perform a gathering task [142]. Brooks, *et al.* report a set of simulations in a similar domain, with a fully decentralized collection of non-communicating robots [30]. Arkin describes a schema-based approach to designing simple navigation behaviors to be extended to multiple physical agents [9]. Matarić cites various researchers who study and simulate ant colonies, examples of simulations of simple organisms collectively producing complex behaviors emerging from simple interactions including: Beni and Hackwood [22], Colorni, Dorigo, and Maniezzo [40], Drogous *et al.* [50], and Travers [148].

Matarić proposed the following set of basic behaviors as a basis for the set of grouping behaviors [105]:

- Safe-wandering - Minimizes collisions between agents and environment, keeping the agents moving while maintaining a minimum distance between them.

- Following - Minimizes interference by structuring movement of any two agents, achieving and maintaining a minimum angle between the position of the leader relative to the follower.
- Aggregation - Gathers the agents, achieving and maintaining a maximum distance between them.
- Dispersion - Dissipates the agents, achieving and maintaining a minimum distance between them.
- Homing - Enables the agent to proceed to a particular location, decreasing the distance between the agent and a goal location called “home.”

Matarić’s investigations into emergent behavior and group dynamics of wheeled vehicles (with no explicit communication) demonstrate that combinations of such simple behaviors as attraction and repulsion, in the form of these basic behaviors, can produce complex relationships such as herding, convoying, foraging, gathering, and flocking [102, 103, 104, 105]. Our architecture is capable of endowing agents with these basic behaviors, thus it is capable of simulating the groups that exhibit Matarić’s grouping behaviors.

Bibliography

- [1] M. D. Adams, Huosheng Hu, and P. J. Probert. Towards a real-time architecture for obstacle avoidance and path planning in mobile robots. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 584–589, 1990.
- [2] Philip Agre. *The Dynamic Structure of Everyday Life*. PhD thesis, MIT Artificial Intelligence Laboratory, 1988. Technical Report 1085.
- [3] Philip Agre and David Chapman. Pengi: An implementation of a theory of activity. *Proceedings of the AAAI-87 Conference*, pages 268–272, June 1987.
- [4] Philip E. Agre and David Chapman. What are plans for? In Pattie Maes, editor, *Designing Autonomous Agents*, pages 17–34. MIT Press, 1990.
- [5] Omar Ahmad, James Cremer, Joseph Kearney, Peter Willemsen, and Stuart Hansen. Hierarchical, concurrent state machines for behavior modeling and scenario control. In *Proceedings of 1994 Conference on AI, Simulation, and Planning in High Autonomy Systems*, Gainesville, FL, Dec 1994.
- [6] R. McNeill Alexander. *Locomotion of Animals*. Blackie & Son Limited, 1982.
- [7] Tracy L. Anderson and Max Donath. Animal behavior as a paradigm for developing robot autonomy. In Pattie Maes, editor, *Designing Autonomous Agents*, pages 145–168. MIT Press, 1990.
- [8] J. R. Andrews. Impedance control as a framework for implementing obstacle avoidance in a manipulator, 1983. S.M. Thesis, MIT, Department of Mechanical Engineering.

- [9] R. C. Arkin. Cooperation without communication: Multiagent schema based robot navigation. *Journal of Robotic Systems*, 1992.
- [10] Ronald C. Arkin. Towards the unification of navigational planning and reactive control. In *AAAI Spring Symposium on Robot Navigation Working Notes*, pages 1–5, March 1989.
- [11] Ronald C. Arkin. Integrating behavioral, perceptual, and world knowledge in reactive navigation. In Pattie Maes, editor, *Designing Autonomous Agents*, pages 105–122. MIT Press, 1990.
- [12] N. Badler, B. Webber, W. Becket, C. Geib, M. Moore, C. Pelachaud, B. Reich, and M. Stone. Planning and parallel transition networks: Animation’s new frontiers. In S. Y. Shin and T. L. Kunii, editors, *Computer Graphics and Applications: Proc. Pacific Graphics ’95*, pages 101–117. World Scientific Publishing, River Edge, NJ, 1995.
- [13] N. Badler, B. Webber, W. Becket, C. Geib, M. Moore, C. Pelachaud, B. Reich, and M. Stone. Planning for animation. In N. Magnenat-Thalmann and D. Thalmann, editors, *Interactive Computer Animation*. Prentice-Hall, 1996.
- [14] Norman I. Badler, Cary B. Phillips, and Bonnie L. Webber. *Simulating Humans: Computer Graphics Animation and Control*, chapter 5, pages 137–150. Oxford University Press, 1993.
- [15] Norman I. Badler, Bonnie L. Webber, and Barry D. Reich. Towards personalities for animated agents with reactive and planning behaviors. In Robert Trappl and Paolo Petta, editors, *Creating Personalities for Synthetic Actors*, 1997. Book-chapter, to appear.
- [16] D. H. Ballard, M. M. Hayhoe, F. Li, and S. D. Whitehead. Hand-eye coordination during sequential tasks. In *Philosophical Transactions of the Royal Society of London B*, London, March 1992.
- [17] J. Bates. The role of emotion in believable agents. *Communications of the ACM*, 37(7):122–125, 1994.
- [18] Welton Becket and Norman I. Badler. Integrated behavioral agent architecture. In *Proceedings of the Third Conference on Computer Generated Forces and Behavior Representation*, pages 57–68, Orlando, Florida, March 1993.

- [19] Welton M. Becket. The *jack* Lisp API. Technical Report MS-CIS-94-01, University of Pennsylvania, Philadelphia, PA, 1994.
- [20] Welton M. Becket. *Discrete and Continuous Learning for Behavioral Control of Simulated Autonomous Agents*. PhD thesis, University of Pennsylvania, 1997.
- [21] Randall D. Beer, Hillel J. Chiel, and Leon S. Sterling. A biological perspective on autonomous agent design. In Pattie Maes, editor, *Designing Autonomous Agents*, pages 169–186. MIT Press, 1990.
- [22] G. Beni and S. Hackwood. The maximum entropy principle and sensing in swarm intelligence. In F. Varela and P. Bourguin, editors, *Toward A Practice of Autonomous Systems: Proceedings of the First European Conference on Artificial Life*, pages 153–160. The MIT Press, 1992.
- [23] G. Beni and U. Wang. Swarm intelligence in cellular robotic systems. In *NATO Advanced Workshop on Robots and Biological Systems*, Il Ciocco, Tuscany, Italy, 1989.
- [24] R. Peter Bonasso, H. James Antonisse, and Marc G. Slack. A reactive robot system for find and fetch tasks in an outdoor environment. *Proceedings of the 8th National Conference on Artificial Intelligence*, pages 801–808, 1992.
- [25] R. Boulic, H. Noser, and D. Thalmann. Automatic derivation of curved human walking trajectories from synthetic vision. In *Proceedings of Computer Animation '94*, pages 93–103, Geneva, Switzerland, May 1994. IEEE Computer Society Press.
- [26] Valentino Braitenberg. *Vehicles: Experiments in Synthetic Psychology*. The MIT Press, 1984.
- [27] D. Brogan and J. Hodgins. Group behaviors for systems with significant dynamics. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robot and Systems*, 1995.
- [28] Rodney A. Brooks. Achieving artificial intelligence through building robots. A.I. Memo 899, MIT AI Lab, 1986.

- [29] Rodney A. Brooks. A robust layered control system for a mobile robot. *IEEE Journal of Robotics and Automation*, pages 14–23, April 1986.
- [30] Rodney A. Brooks. Elephants don't play chess. In Pattie Maes, editor, *Designing Autonomous Agents*, pages 3–18. MIT Press, 1990.
- [31] Rodney A. Brooks. Intelligence without reason. In *IJCAI-91*, pages 569–595, August 1991.
- [32] Rodney A. Brooks. Intelligence without representation. *AI Journal*, 47:139–159, 1991.
- [33] Rodney A. Brooks. New approaches to robotics. *Science*, 253:1227–1232, 1991.
- [34] Rodney A. Brooks and Jonathan H. Connell. Asynchronous distributed control system for a mobile robot. In *SPIE's Cambridge Symposium on Optical and Opto-Electronic Engineering Proceedings*, volume 727, pages 77–84, October 1986.
- [35] Armin Bruderlin and Tom Calvert. Knowledge-driven, interactive animation of human running. In *Graphics Interface*, pages 213–221, May 1996.
- [36] David Chapman. Planning for conjunctive goals. *Artificial Intelligence*, 32(3):333–377, 1987.
- [37] David Chapman. Penguins can make cake. *AI Magazine*, 10(4):45–50, Winter 1989.
- [38] David Chapman. *Vision, Instruction, and Action*. The MIT Press, 1991.
- [39] R. Chatila and J. Laumond. Position referencing and consistent world modeling for mobile robots. In *Proceedings of the IEEE Conference on Robotics and Automation*, pages 138–145, March 1985.
- [40] A. Coloni, M. Dorigo, and V. Maniezzo. Distributed optimization by ant colonies. In F. Varela and P. Bourguine, editors, *Toward A Practice of Autonomous Systems: Proceedings of the First European Conference on Artificial Life*, pages 134–142. The MIT Press, 1992.
- [41] Jonathan H. Connell. A colony architecture for an artificial creature. Technical report, 1151, MIT AI Lab, June 1990.

- [42] Jonathan H. Connell. SSS: A hybrid architecture applied to robot navigation. Technical report, IBM Research Report, 1991.
- [43] Ingemar J. Cox. Blanche: an autonomous robot vehicle for structured environments. *IEEE Journal of Robotics and Automation*, pages 978–982, 1988.
- [44] Chris Crawford. Evolutionary game design. *Morph's Outpost on the Digital Frontier*, pages 27–29, April 1995.
- [45] James Cremer, Joseph Kearney, and Yiannis Papelis. HCSM: A framework for behavior and scenario in virtual environments. *ACM Transactions on Modeling and Computer Simulation*, 5(3):242–267, July 1995.
- [46] James Cremer, Joseph Kearney, and Peter Willemsen. A directable vehicle behavior model for virtual driving environments. In *Proceedings of 1996 Conference on AI, Simulation, and Planning in High Autonomy Systems*, La Jolla, CA, March 1996.
- [47] J. M. Cullen, E. Shaw, and H. A. Baldwin. Methods for measuring the three-dimensional structure of fish schools. *Animal Behavior*, 13:534–543, 1965.
- [48] J. L. Deneubourg, G. Theraulax, and R. Beckers. Swarm-made architectures. In F. Varela and P. Bourguine, editors, *Toward A Practice of Autonomous Systems: Proceedings of the First European Conference on Artificial Life*, pages 123–133. The MIT Press, 1992.
- [49] Brett Douville. A theoretical description of PaT-Nets. Technical report, Department of Computer and Information Science, University of Pennsylvania, 1995.
- [50] A. Drogous, J. Ferber, B. Corbara, and D. Fresneau. A behavioral simulation model for the study of emergent social structures. In F. Varela and P. Bourguine, editors, *Toward A Practice of Autonomous Systems: Proceedings of the First European Conference on Artificial Life*, pages 161–170. The MIT Press, 1992.
- [51] Andrew P. Duchon, William H. Warren, and Leslie P. Kaelbling. Ecological robotics: Controlling behavior with optical flow. In *Proceedings of the Seventeenth Annual Conference of the Cognitive Science Society*, 1995.

- [52] A. Elfes. A sonar-based mapping and navigation system. In *Proceedings of the IEEE International Conference on Robotics and Automation*, page 1151, 1986.
- [53] Richard E. Fikes and Nils J. Nilsson. STRIPS: A new approach to the application of theorem proving to problem solving. *Artificial Intelligence*, 8:189–208, 1971.
- [54] Robert J. Firby. *Adaptive Execution in Complex Dynamic Worlds*. PhD thesis, Yale University, 1989.
- [55] James D. Foley, Andries van Dam, Steven K. Feiner, and John F. Hughes. *Computer Graphics: Principles and Practice*. Addison-Wesley Publishing Co., Reading, MA, 2nd edition, 1990.
- [56] Erann Gat. Alfa: A language for programming reactive robotic control systems. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 1116–1121, April 1991.
- [57] Erann Gat. Robust low-computation sensor-driven control for task directed navigation. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 2484–2489, April 1991.
- [58] Erann Gat. *Taking the Second Left: Reliable Goal-Directed Reactive Control for Real-World Autonomous Robots*. PhD thesis, Dept. of Computer Science, VPI, 1991.
- [59] Erann Gat. Integrating planning and reacting in a heterogeneous asynchronous architecture for controlling real-world mobile robots. *Proceedings of the 8th National Conference on Artificial Intelligence*, pages 809–815, 1992.
- [60] Erann Gat. On the role of stored internal state in the control of autonomous mobile robots. *AI Magazine*, pages 64–73, Spring 1993.
- [61] Erann Gat, Joe Fearey, and Joe Provenzano. Semi-automated forces for corps battle simulation. In *Proceedings of the Third Conference on Computer Generated Forces and Behavior Representation*, pages 69–74, Orlando, Florida, March 1993.
- [62] Erann Gat and David P. Miller. Modular, low-computation robot control for object acquisition and retrieval. JPL Internal Report, 1991.

- [63] Erann Gat, Marc G. Slack, David P. Miller, and R. James Firby. Path planning and execution monitoring for a planetary rover. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 20–25, 1990.
- [64] Christopher W. Geib. *The Intentional Planning System: ItPlanS*. PhD thesis, University of Pennsylvania, 1995.
- [65] M. P. Georgeff. Agents and their plans. In *Proceedings of the 14th International Joint Conference on Artificial Intelligence*, Montreal, 1995. Invited Lecture.
- [66] Michael P. Georgeff and Amy L. Lansky. Reactive reasoning and planning. In James Allen, James Hendler, and Austin Tate, editors, *Readings in Planning*, pages 729–734. Morgan Kaufmann Publishers, Inc., 1990.
- [67] Matthew Ginsberg. Universal planning: An (almost) universally bad idea. *AI Magazine*, 10(4), Winter 1989.
- [68] G. Giralt, R. Chatila, and M. Vaisset. An integrated navigation and motion control system for autonomous multisensory mobile robots. In M. Brady and R. Paul, editors, *First International Symposium on Robotics Research*, pages 191–214. MIT Press, Cambridge, MA, 1984.
- [69] R. L. Gould. Making 3-d computer character animation: A great future of unlimited possibility or just tedious? SIGGRAPH 89 Course Notes: 3D Character Animation by Computer, 1989. pages 31-60.
- [70] John P. Granieri, Welton Becket, Barry D. Reich, Jonathan Crabtree, and Norman I. Badler. Behavioral control for real-time simulated human agents. In Pat Hanrahan and Jim Winget, editors, *1995 Symposium on Interactive 3D Graphics*, pages 173–180, April 1995.
- [71] David R. Haumann and Richard E. Parent. The behavioral testbed: Obtaining complex behavior from simple rules. *The Visual Computer*, 4(6), 1988.
- [72] B. Hayes-Roth. Agents on stage: Advancing the state of the art of AI. In *Proceedings of the 14th International Joint Conference on Artificial Intelligence*, pages 967–971, Montreal, 1995.

- [73] J. Hodgins and D. Brogan. Robot herds: Group behaviors for systems with significant dynamics. In *Proceedings of Artificial Life IV*, pages 319–324, 1994.
- [74] J. K. Hodgins, W. L. Wooten, D. C. Brogan, and J. F. O’Brien. Animating human athletics. In *Computer Graphics*, pages 71–78, August 1995.
- [75] Jessica K. Hodgins. Three-dimensional human running. In *Proceedings of the IEEE Conference on Robotics and Automation*, 1996.
- [76] Klaus Immelmann and Colin Beer. *A Dictionary of Ethology*. Harvard University Press, 1989.
- [77] Verne T. Inman, Henry J. Ralston, and Frank Todd. Human locomotion. In Jessica Rose and James G. Gamble, editors, *Human Walking*, pages 1–22. Williams & Wilkins, 1994.
- [78] Leslie P. Kaelbling. Rex: A symbolic language for the design and parallel implementation of embedded systems. In *Proceedings of the AIAA Conference on Computers in Aerospace*, 1987.
- [79] Leslie P. Kaelbling. Goals as parallel program specifications. In *Proceedings of the AAAI-88 Conference*, 1988.
- [80] Leslie P. Kaelbling. An architecture for intelligent reactive systems. In James Allen, James Hendler, and Austin Tate, editors, *Readings in Planning*, pages 713–728. Morgan Kaufmann Publishers, Inc., 1990.
- [81] O. Khatib. Real-time obstacle avoidance for manipulators and mobile robots. In *Proceedings of the IEEE International Conference on Robotics and Automation*, 1985.
- [82] Hyeongseok Ko. *Kinematic and Dynamic Techniques for Analyzing, Predicting, and Animating Human Locomotion*. PhD thesis, University of Pennsylvania, 1994.
- [83] Hyeongseok Ko and Norman I. Badler. Animating human locomotion in real-time using inverse dynamics, balance and comfort control. *IEEE Computer Graphics and Applications*, 16(2):50–59, March 1996.

- [84] Hyeongseok Ko and James Cremer. VRLOCO: Real-time human locomotion from positional input streams. *Presence*, 5(4):367–380, Fall 1996.
- [85] Hyeongseok Ko, Barry D. Reich, Welton Becket, and Norman I. Badler. Terrain navigation skills and reasoning. In *Proceedings of the Fourth Conference on Computer Generated Forces and Behavioral Representation*, pages 219–227, Orlando, Florida, May 4-6 1994.
- [86] C. Ronald Kube and Hong Zhang. Collective robotic intelligence. In *Proceedings of the Second International Conference on Simulation of Adaptive Behaviors*, pages 460–468, Honolulu, Hawaii, December 7-12 1992.
- [87] Benjamin J. Kuipers and Tod S. Levitt. Navigation and mapping in large-scale space. *AI Magazine*, 9(2):25–43, Summer 1988.
- [88] C. G. Langton. *Artificial Life*. Addison-Wesley, 1989.
- [89] C. G. Langton. Computation at the edge of chaos: Phase transitions and emergent computation. *Physica*, D(42):12–37, 1990.
- [90] Jean-Claude Latombe. *Robot Motion Planning*. Kluwer Academic Publishers, 1991.
- [91] Michael J. Longtin. Cover and concealment in ModSAF. In *Proceedings of the Fourth Conference on Computer Generated Forces and Behavioral Representation*, pages 239–247, Orlando, Florida, May 4-6 1994.
- [92] Tomás Lozano-Pérez and Michael Wesley. An algorithm for planning collision-free paths among polyhedral obstacles. *Communications of the ACM*, 22(10):560–570, October 1979.
- [93] Pattie Maes. The dynamics of action selection. In *IJCAI-89*, pages 991–997, Detroit, MI, 1989.
- [94] Pattie Maes. Designing autonomous agents: Theory and practice from biology to engineering and back. In Pattie Maes, editor, *Designing Autonomous Agents*, pages 1–2. MIT Press, 1990.
- [95] Pattie Maes. Situated agents can have goals. In Pattie Maes, editor, *Designing Autonomous Agents*, pages 49–70. MIT Press, 1990.

- [96] Pattie Maes. Agents that reduce work and information overload. *Communications of the ACM*, 37(7), July 1994.
- [97] Pattie Maes. Artificial life meets entertainment: Lifelike autonomous agents. *Communications of the ACM*, 38(11):108–114, 1995.
- [98] Sang Mah, Thomas W. Calvert, and William Havens. A constraint-based reasoning framework for behavioural animation. *Computer Graphics Forum*, 13(5):315–324, 1994.
- [99] D. Marr. *Vision*. Freeman, San Francisco, 1984.
- [100] Maja Matarić. A distributed model for mobile robot environment learning and navigation. Technical Report AI-TR 1228, MIT AI Lab, 1990.
- [101] Maja J. Matarić. Behavior-based control: Main properties and implications. In *Proceedings, IEEE International Conference on Robotics and Automation, Workshop on Architectures for Intelligent Control Systems*, pages 46–54, Nice, France, May 1992.
- [102] Maja J. Matarić. Designing emergent behaviors: From local interactions to collective intelligence. In J-A Meyer, H. Roitblat, and S. Wilson, editors, *Proceedings, From Animals to Animats, Second International Conference on Simulation of Adaptive Behavior (SAB-92)*, pages 432–441. MIT Press, 1992.
- [103] Maja J. Matarić. Minimizing complexity in controlling a mobile robot population. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 830–835, 1992.
- [104] Maja J. Matarić. Kin recognition, similarity, and group behavior. In *Proceedings of the Fifteenth Annual Cognitive Science Society Conference*, pages 705–710, Boulder, Colorado, June 1993. Lawrence Erlbaum Associates.
- [105] Maja J. Matarić. *Interaction and Intelligent Behavior*. PhD thesis, Massachusetts Institute of Technology, May 1994.
- [106] H. R. Maturana and F. J. Varela. *The Tree of Knowledge: The Biological Roots of Human Understanding*. New Science Library, Boston, MA, 1988.

- [107] Michael Mauldin. Chatterbots, tinymuds, and the turing test: Entering the Loebner prize competition. In *Proceedings of the AAAI 1994 Conference*, pages 16–21. MIT Press, 1994.
- [108] D. McFarland, editor. *The Oxford Companion to Animal Behavior*. Oxford University Press, 1987.
- [109] D. P. Miller and M. G. Slack. Global symbolic maps from local navigation. In *Proceedings of the Ninth National Conference on Artificial Intelligence*, Anaheim, California, July 1991. AAAI Press.
- [110] David P. Miller, Rajiv S. Desai, Erann Gat, Robert Ivlev, and John Loch. Reactive navigation through rough terrain: Experimental results. *Proceedings of the 8th National Conference on Artificial Intelligence*, pages 823–828, 1992.
- [111] Michael B. Moore. Search plans. Technical Report MS-CIS-93-56/LINC LAB 250/IRCS-93-29, Department of Computer and Information Science, University of Pennsylvania, 1993.
- [112] Michael B. Moore, Christopher Geib, and Barry D. Reich. Planning and terrain reasoning. In *AAAI Spring Symposium on Integrated Planning Applications*, Stanford, CA, 1995. (also University of Pennsylvania CIS department Technical Report MS-CIS-94-63/LINC LAB 280).
- [113] Michael B. Moore, Christopher Geib, and Barry D. Reich. Planning for reactive behaviors in hide and seek. In *Proc. 5th Conference on Computer Generated Forces and Behavioral Representation*, Orlando, FL, May 1995. Institute for Simulation and Training.
- [114] H. Moravec. Rover visual obstacle avoidance. In *Proceedings of the 7th International Joint Conference on Artificial Intelligence*, pages 785–790, 1981.
- [115] H. Moravec and D. Cho. A bayesian method for certainty grids. In *AAAI Spring Symposium on Robot Navigation*, pages 57–60, March 1989.
- [116] Eadweard Muybridge. *Descriptive Zoopraxography: The Science of Animal Locomotion Made Popular*. The Lakeside Press, Chicago, 1893.

- [117] Eadweard Muybridge. *The Human Figure in Motion*. Dover Publications, New York, 1955.
- [118] Eadweard Muybridge. *Animal Locomotion; the Muybridge Work at the University of Pennsylvania*. Arno Press, New York, 1973.
- [119] Eadweard Muybridge. *Muybridge's Complete Human and Animal Locomotion*. Dover Publications, New York, 1979.
- [120] Eadweard Muybridge. *The Male and Female Figure in Motion*. Dover Publications, New York, 1984.
- [121] Nils J. Nilsson. *Principles of Artificial Intelligence*. Morgan Kaufmann, San Mateo, California, 1980.
- [122] Nils J. Nilsson. Shakey the robot. SRI International Technical Note, No. 325, 1984.
- [123] B. L. Partridge. The structure and function of fish schools. *Scientific American*, pages 114–183, June 1982.
- [124] David W. Payton. An architecture for reflexive autonomous vehicle control. IEEE Robotics and Automation Conference, 1986.
- [125] David W. Payton. Internalized plans: A representation for action resources. In Pattie Maes, editor, *Designing Autonomous Agents*, pages 89–103. MIT Press, 1990.
- [126] Ken Perlin. Danse interactif. *SIGGRAPH Video Review*, 101, 1994.
- [127] Ken Perlin. Real time responsive animation with personality. *IEEE Transactions on Visualization and Computer Graphics*, 1(1):5–15, March 1995.
- [128] Barry D. Reich, Hyeongseok Ko, Welton Becket, and Norman I. Badler. Terrain reasoning for human locomotion. In *Proceedings of Computer Animation '94*, pages 76–82, Geneva, Switzerland, May 1994. IEEE Computer Society Press.
- [129] Olivier Renault, Nadia Magnenat Thalmann, and Daniel Thalmann. A vision-based approach to behavioral animation. *The Journal of Visualization and Computer Animation*, 1(1):18–21, 1990.

- [130] Craig W. Reynolds. Flocks, herds, and schools: A distributed behavioral model. *Computer Graphics*, 21(4):25–34, 1987.
- [131] Craig W. Reynolds. Stanley and Stela in Breaking the Ice. *SIGGRAPH Video Review*, 36, 1987.
- [132] Craig W. Reynolds. Not bumping into things. SIGGRAPH Course 27 notes: Developments in Physically-Based Modeling, 1988. pages G1-G13.
- [133] Gary Ridsdale. Connectionist modelling of skill dynamics. *The Journal of Visualization and Computer Animation*, 1(2):66–72, 1990.
- [134] Jessica Rose, J. Henry Ralston, and James G. Gamble. Energetics of walking. In Jessica Rose and James G. Gamble, editors, *Human Walking*, pages 45–72. Williams & Wilkins, 1994.
- [135] E. Shaw. Schooling in fishes: Critique and review. In L. Aronson, E. Tobach, D. Lehman, and J. Rosenblatt, editors, *Development and Evolution of Behavior*, pages 452–480, 1970.
- [136] E. Shaw. Fish in schools. *Natural History*, 84(8):40–46, 1975.
- [137] Herbert A. Simon. *Sciences of the Artificial*. MIT Press, Cambridge, Massachusetts, 1969.
- [138] Marc G. Slack. Situationally driven local navigation for mobile robots, April 1990. JPL Publication 90-17, California Institute of Technology Jet Propulsion Laboratory.
- [139] Marc G. Slack. Navigation templates: Mediating qualitative guidance and quantitative control in mobile robots. *IEEE Transactions on Systems, Man, and Cybernetics*, 23(2):452–466, March/April 1993.
- [140] Aaron Sloman and Riccardo Poli. Sim_agent: A toolkit for exploring agent designs. In M. Wooldridge, J. P. Muller, and M. Tambe, editors, *Intelligent Agents II (LNAI 1037)*, pages 392–407. Springer-Verlag: Heidelberg, Germany, 1996.
- [141] Monnett Hanvey Soldo. Reactive and preplanned control in a mobile robot. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 1128–1132, 1990.

- [142] Luc Steels. Cooperation between distributed agents through self-organization. In *Workshop on Multi-Agent Cooperation*. North Holland, Cambridge, UK, 1989.
- [143] Luc Steels. Exploiting analogical representations. In Pattie Maes, editor, *Designing Autonomous Agents*, pages 71–88. MIT Press, 1990.
- [144] Luc Steels. Emergent functionality in robotic agents through on-line evolution. In Rodney A. Brooks and Pattie Maes, editors, *Proceedings of the 4th International Workshop on the Synthesis and Simulation of Living Systems ArtificialLifeIV*, pages 8–16, Cambridge, MA, July 1994. MIT Press.
- [145] Arthur Steindler. *Kinesiology of the Human Body Under Normal and Pathological Conditions*. Charles C Thomas, Publisher, 1955.
- [146] Demetri Terzopoulos, Xiaoyuan Tu, and Radek Grzeszczuk. Artificial fishes with autonomous locomotion, perception, behavior and learning, in a physical world. In Pattie Maes and Rodney Brooks, editors, *Proceedings of Artificial Life IV*, Cambridge, MA, 1994. MIT Press.
- [147] Robert B. Tilove. Local obstacle avoidance for mobile robots based on the method of artificial potentials. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 566–571, 1990.
- [148] M. Travers. Animal construction kits. In C. Langton, editor, *Artificial Life*. Addison-Wesley, 1988.
- [149] Thomas S. Trias, Sonu Chopra, Barry D. Reich, Michael B. Moore, Norman I. Badler, Bonnie L. Webber, and Christopher W. Geib. Decision networks for integrating the behaviors of virtual agents and avatars. In *IEEE VRAIS '96*, 1996.
- [150] Xiaoyuan Tu and Demetri Terzopoulos. Artificial Fishes: Physics, Locomotion, Perception, Behavior. *Computer Graphics*, 28:43–50, 1994.
- [151] Toby Tyrell. *Computational Mechanisms for Action Selection*. PhD thesis, University of Edinburgh, 1993.

- [152] S. Ullmann. Visual routines. *Cognition*, 18, 1984.
- [153] S. Veherencamp. *Handbook of Behavioral Neurobiology, Volume 3: Social Behavior and Communication*. Marler, P. and Vandenberg, J. G. (eds.), 1987.
- [154] Bonnie Webber and Norman Badler. Animation through reactions, transition nets and plans. In *Proc. Int'l Workshop on Human Interface Technology*, Aizu, Japan, October 1995.
- [155] R. Wehner. Matched filters - neural models of the external world. *Journal of Computational Physiology*, A(161):511–531, 1982.
- [156] Jane Wilhelms. Toward automatic control. *IEEE Computer Graphics and Applications*, 7(4):11–22, April 1987.
- [157] Jane Wilhelms and Robert Skinner. A 'notion' for interactive behavioral animation control. *IEEE Computer Graphics and Applications*, 10(3):14–22, May 1990.