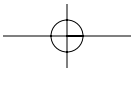


Part I Introduction

In Chapter 1, the phrase "game AI" will be defined, and the more ambiguous uses of the expression will be reassigned to more appropriate language. Some applicable theory will also be discussed, both from the psychological world and from the academic artificial intelligence (AI) world.

In Chapter 2, the foundation systems that make up a general game AI engine will be discussed, along with the elements that need to be considered when designing a new AI engine.

In Chapter 3, the basic example application that later parts of the book will use as an AI implementation test bed will be documented and discussed.

1 Basic Definitions and Concepts

Welcome to *AI Game Engine Programming.* This book is meant to give the game artificial intelligence (AI) programmer the knowledge and tools needed to create AI engines for modern commercial games. But what exactly do we mean by "game AI"? AI as a science is relatively young; some of the earliest work was done in the early 1950s. Games have used real AI techniques for an even shorter time because of the computation and storage space limitations of earlier game machines. Because AI is a rather new concept in games, the definition of game AI is not clear for most people, even those who practice game production. This chapter will define the term *game AI,* identify practices and techniques that are commonly mistaken for game AI, and discuss areas of future expansion. Later in the chapter, relevant concepts from other fields, including mind science, psychology, and robotics, will be discussed regarding game AI systems.

WHAT IS INTELLIGENCE?

The term *intelligence* is fairly nebulous. The dictionary will tell you it is *the capacity to acquire and apply knowledge,* but this is far too general. This definition, interpreted literally, could mean that your thermostat is intelligent. It acquires the knowledge that the room is too cold and applies what it learned by turning on the heater. The dictionary goes on to suggest that intelligence demonstrates *the faculty of thought and reason*. Although this is a little better (and more limiting; the thermostat has been left behind), it really just expands our definition problem by introducing two even more unclear terms, *thought* and *reason.* In fact, the feat of providing a *true* definition of intelligence is an old and harried debate that is far beyond the scope of this text. Thankfully, making good games does not require this definition. Actually, this text will agree with our first dictionary definition, as it fits nicely with what we expect game systems to exhibit to be considered intelligent. For our purposes, an intelligent game agent is one that acquires knowledge about the world, and then acts on that knowledge. The quality and effectiveness of his actions then become a question of game balance and design.

WHAT IS "GAME AI"?

Let us start by giving an academic definition to AI. In their seminal AI Bible, *Artificial Intelligence: A Modern Approach,* Russel and Norvig [Russel95] say that AI is the creation of computer programs that emulate acting and thinking like a human, as well as acting and thinking rationally. This definition encompasses both the cognitive and the behavioral views of intelligence and includes rationality and "humanity" (because being human is sometimes far from rational), but is still considered intelligent (like running into a burning building to save your child).

Game AI is the code in a game that makes the computer-controlled opponents (or cooperative elements) appear to make smart decisions when the game has multiple choices for a given situation, resulting in behaviors that are relevant, effective, and useful. Note the word "appear" in the last sentence. The AI-spawned behaviors in games are very *results* oriented, and thus, we can say that the game world is primarily concerned with the behavioralist wing of AI science. We're really only interested with the responses that the system will generate, and don't really care how the system arrived at it. We care about how the system *acts*, not how it *thinks.* People playing the game don't care if the game is using a huge database of scripted decisions, is making directed searches of a decision tree, or is building an accurate knowledge base of its surroundings and making inferred choices based on logical rules. The proof is really, as they say, all in the pudding as far as game AI goes. So, purely behavioral decisions, such as which attacks an opponent launches, how he navigates the terrain to get to you, how he uses the elements in the environment, and many other details can and are all considered decisions best left to the game's AI system.

Modern game development also uses the term *AI* to describe other game behavior, for instance, the way the interface works with input from the human. Even the algorithms that govern movement and collision (if the game uses animation-driven movement, rather than physics simulations) sometimes fall under this category. As you can see, the term *AI* is a broadly used moniker in the game development world. When discussing AI with someone else in the industry (or even within the company you work at), it's important to know that you both agree on the meaning and scope of the term; otherwise miscommunication can occur if your notion of AI is vastly different (be it simpler or more complex, or just at opposite ends of the responsibility spectrum) than the other person's. When this book refers to AI, it will use the rather narrow definition *character-based behavioral intelligence.* The term *character* refers to the character (or actor, or agent) driven nature of most games. True, many strategy or so-called "God" games have the notion of an "overseer," that is making decisions in the big picture sense, but this could again be considered just another character.

In the old days, AI programming was more commonly referred to as "gameplay programming," because there really wasn't anything intelligent about the behaviors exhibited by the CPU-controlled characters. See Figure 1.1 for an overall game AI timeline. Most coders in the early days of video gaming relied on patterns or some

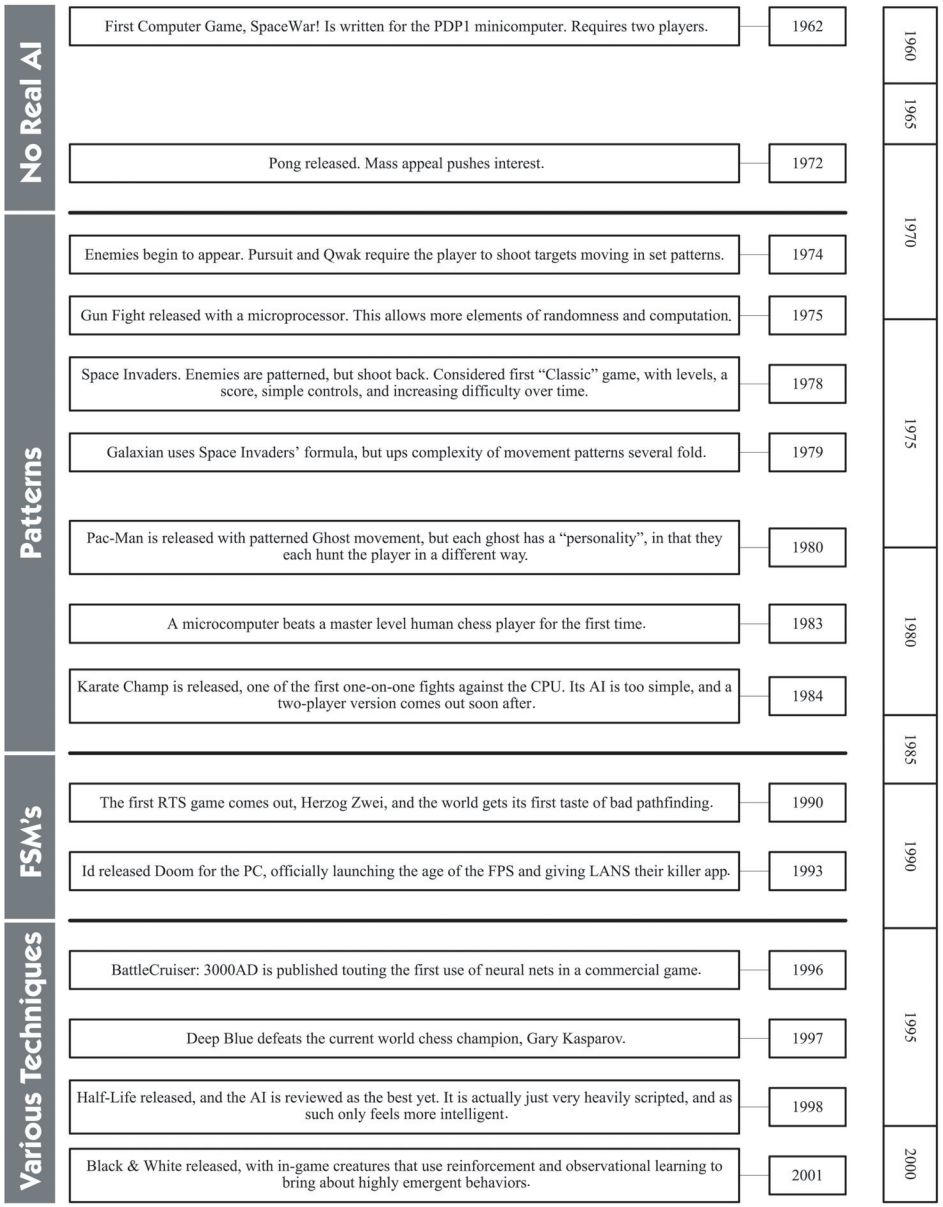


FIGURE 1.1 Game AI timeline.

repetitive motions for their enemies (e.g., *Galaga* or *Donkey Kong*), or they used enemies that barely moved at all but were vulnerable to attack only in certain "weak points" (like *R-Type*). That was the game, to some extent: finding the predetermined behavior patterns so that you could easily beat that opponent (or wave of opponents) and move on to another. This was done because of the extreme restraints of early processor speed and memory storage. Patterns could be stored easily, requiring minimal code to drive them, and required no calculation; the game simply moved the enemies around in the prescribed patterns, with whatever other behavior they exhibited layered on top (for instance, the *Galaga* enemies shooting when you're beneath them while moving in a pattern). In fact, some games used supposed *random* movement, but because the random number generator in these early games used a hard-coded table of pseudo-random numbers, a pattern could eventually be seen in the overall game behavior.

Another commonly used technique in the past to make games appear smarter was to allow the computer opponents to cheat; that is, to have additional information about the game world that the human player does not have and, thus, make decisions about what to do next that seem remarkably smart. The computer reads that you pushed the punch button (before you've even started the punch animation) and responds with a blocking move. A real-time strategy (RTS) cheater might have its workers heading toward valuable resource sites early in the game, when they hadn't explored the terrain to find those resources. AI cheating is also achieved when the game grants *gifts* to the computer opponent, by providing him additional (and strategically timed) abilities, resources, and so forth that it uses outright, instead of planning ahead and seeing the need for these resources on its own. These tactics lead to more challenging but ultimately less satisfying opponents because a human player can almost always pick up on the notion that the computer is accomplishing things that are impossible for him. One of the easier to notice and most frustrating examples of this impossible behavior is the use of what is called *rubber banding* in racing games. Toward the end of a race, if you're beating the AI-controlled cars by too much, some games simply speed up the other cars until they've caught up with you, and then they return to normal. Sure, it makes the race more of a battle, but watching a previously clueless race car suddenly perform miracles to catch up to you borders on the ridiculous.

In modern games, the old techniques are being increasingly abandoned. The primary selling point of games is slowly but surely evolving into the realm of AI accomplishments and abilities, instead of the graphical look of the game as it was during the most recent phase of game development. This emphasis on visuals is actually somewhat causal in this new expansion of AI importance and quality; the early emphasis on graphics eventually led to specialized graphics processors on almost every platform, and the main CPU is increasingly being left open for more and more sophisticated AI routines. Now that the norm for game graphics is so

high, the wow factor of game graphics is finally wearing thin, and people are increasingly concentrating on other elements of the game itself. So, the fact that we now have more CPU time is very advantageous, considering that the current consumer push is now for games that contain much better AI-controlled enemies. In the 8-bit days of gaming or before, 1–2% of total CPU time was the norm, if not an overestimation, for a game's AI elements to run in. Now, games that require AI are budgeting 10–35% of the CPU time to the AI system [Woodcock01], with some games going even higher.

This means that today's game opponents can find better game solutions without cheating and can use more adaptive and emergent means—if for no other reason that they have access to faster and more powerful processors driving them. As such, AI as defined in modern games is increasingly showing more *real* intelligence (as defined by academic AI), instead of the old standby of prescripted patterns or behaviors mimicking intelligent behavior. Traditional game AI work is being progressively more infused with techniques from the academic world of AI (heuristic search, learning, planning, etc.), which will only continue as games (and gamer's tastes) become more complex.

WHAT GAME AI IS NOT

As shown in the previous section, game AI is a broad label that is often inaccurately used when referring to anything about the game other than graphics: the collision avoidance (or pathfinding) system, the player controls, the user interface, and sometimes the animation system for a game are all pushed into the AI corner. To some extent, these elements do have something to add to the AI world and are elements that, if done poorly, will make the AI seem "stupider," but they are not the primary AI system in a game. An exception to this rule might be a game in which the gameplay is simple enough that the entire smarts of the enemies are in moving around or choosing the right animations to play.

This book will emphasize this differentiation: game AI makes intelligent decisions when there are multiple options or directions for play. These secondary systems, while making decisions from a pool of solutions/animations/paths, are more "Find the *BEST*" (read: singular) solution for this particular input. The main AI, in contrast, might have many equally good solutions, but needs to consider planning, resources, player attributes, and so on to make decisions for the game's bigger picture.

An alternative way of thinking about this differentiation is that these support systems are much more low-level intelligence, whereas this book will focus more on the high-level decisions that an AI system needs to make. For example, you get out of your chair and walk across the room. The thought in your mind was, "I want

a soda out of the fridge." But look at all the low-level intelligence you used to accomplish the task: your mind determined the right sequence of muscle contractions to get you out of the chair (animation picking), and then started you moving toward the fridge, threading you through all the things on the floor (pathfinding). In addition, you might have lost your balance but regained it quickly (physics) or scratched your head on the way there, in addition to a myriad other minor actions. None of these changed the fact that your entire plan was to go get a soda, which you eventually accomplished. Not all games use a tiered decision system, with an overseeing high-level AI, however. Most just split up the various levels of decision making into separate systems that barely communicate. The point is that these low-level systems do *support* the intelligence of the agent but, for this book's purposes, do not *define* the intelligence of an AI-controlled agent.

Another point to consider is that creating better game AI is not necessarily a result of writing better code. Many programmers believe that AI creation is a technical problem that can be solved purely with programming skill, but there's much more to it than that. When building game AI, a good software designer must consider balancing issues from such disparate areas as gameplay, aesthetics, animation, audio, and behavior of both the AI and the game interface. It is true that a vast number of highly technical challenges must be overcome by the AI system. However, the ultimate goal of the AI is to provide the player with an entertaining experience, not to be a demonstration for your clever code. Gamers will care not about your shiny new algorithm if it doesn't feel smart *and* fun. Game AI is not the best code; it is the best use of code and a large dollop of " whatever works (WW)." Some of the smartest looking games in the past have used very questionable methods to achieve their solutions, and although this book is not advocating poorly written code, nothing should be thrown away if it helps to give the illusion of intelligence and enhances the fun factor of the game. Plus, some of the most elegant game code in the world started out as a mindless hack, which uncovered a clever algorithm on retrospection and cleanup.

The reason the WW mentality has permeated the community is because of a common situation that comes into play very late in a game's development and is usually a sign of poor scheduling. Because AI concerns were so low on the totem pole in years past, the desperate AI programmers of old were somewhat forced to use these questionable methods to cram smarts into their systems. This *cramming* may have lead to better-looking behaviors, but it also led to difficulty debugging, maintaining, and extending game code. Luckily for us, this type of thing is becoming minimal because AI is becoming more and more important to the success of the game and is therefore given the forethought of extended technical design, as well as the time necessary to polish. If you are trying to use a generalized AI algorithm that is clean and elegant, but finding that it is either limiting the types of things your

engine can accomplish, or is causing undue pressure on the production staff by requiring a lot more secondary resources, then the issue of utility needs to be brought up. The world of game AI is just like the rest of the universe—there isn't always an elegant solution for everything.

On a less serious note, game AI is also not some kind of new life form, a disconnected brain that will eventually take over your PlayStation® and command you to feed it regularly. Hollywood tells us that this is what AI has in store for us, but the truth is likely far less dramatic. In the future years of AI research, we will most likely have access to a truly generic AI paradigm that could learn to competently play any game, but for now this is not the case. Nowadays, game AI is still very game specific, and very much in the hands of the coders that work on it. The field is still widely misunderstood by the nonprogramming public, however, and even by those people working in game development who don't regularly work with AI systems. As such, care must be taken not to fall prey to the whims, suggestions, and "game ideas" from management or game designers who don't fully understand the limitations of the AI system in a particular game.

HOW THIS DEFINITION DIFFERS FROM THAT OF ACADEMIC AI

The world of academic AI has two main goals. First is to help us *understand* intelligent entities, which will in turn help us to understand ourselves. This first goal is also the goal of more esoteric fields, such as philosophy and psychology, but in a much more functional way. Rather than the philosophical, "Why are we intelligent?," or the psychological, "Where in the brain does intelligence come from?," AI is more concerned with the question, "How is that guy finding the right answer?" Second is to *build* intelligent entities, for fun and profit, you might say, because it turns out that these intelligent entities can be useful in our everyday lives. The second goal mirrors the nature of the practical economy (especially in the so-called western world), in that the research that is most likely to result in the largest profits is also the most likely to win funding.

Russel and Norvig [Russel95] define AI as the creation of computer programs that emulate four things: thinking humanly, thinking rationally, acting humanly, and acting rationally. In academic study, all four parts of this definition have been the basis for building intelligent programs. The Turing test is a prime example of acting humanly—if you cannot tell the difference between the actions of the program and the actions of a person, that program is intelligent. Cognitive theories that are helping to blend traditional human mind science into AI creation will lead to more humanlike behaviors and, we can hope, to "think" in the same ways that humans do. Sheer logic systems try to solve problems without personal bias or

emotion, by thinking purely rationally. But it is acting rationally—always trying to come up with the answer—that most defines academic AI methods. Acting rationally is in fact the primary leg of the four-part definition that researchers are striving for in labs across the world.

On this point, the major goal of game AI and the more traditional studies part ways. Academic AI is usually concerned with making rational decisions, that is, the best or most correct (given that there might not be a best) situation. In contrast, game AI focuses on acting "human," with much less dependence on total rationality. This is because game AI needs to model the highs and lows of human task performance, instead of a rigorous search toward the best decision at all times. This is for entertainment reasons, of course.

Say you're making a chess game. If you're making this chess game as part of an academic study, you want it to play the best game possible, given time and memory constraints. You are going to try to achieve perfect rationality, using highly tuned AI techniques to help you navigate the sea of possible actions. However, if you are building your chess game to give a regular human player an entertaining opponent to play against, then your goal shifts dramatically. Now you want a game that provides the person with a suitable challenge, but doesn't overwhelm the human by always making the best move. Yes, the techniques used to achieve these two programs might parallel in some ways, but because the primary goal of each program is different, the coding of the two systems will dramatically diverge. The people who coded *Big Blue* did not care if Kasparov was having fun when playing against it. But the people behind the very popular *Chessmaster* games surely spend a lot of time thinking about the fun factor, especially at the default difficulty setting.

Chess is a somewhat odd example because humans playing a chess program usually expect it to do pretty well (unless they're just learning and have specifically set the difficulty rating of the program to a low level). But imagine the same example used in an AI-controlled *Quake* "bot" deathmatch opponent. If the bot came into the room, dodged perfectly, aimed perfectly, and knew exactly where and when powerups spawned in the map, it wouldn't be very fun to play against. Instead, we want a much more *human* level of performance from a game AI opponent. We want to play against an enemy that occasionally misses, runs out of ammo in the middle of a fight, jumps wrong and falls, and everything else that makes an opponent appear human. We still want competent opponents, but because our measure of competence, as humans, involves a measure of error, we expect shortcomings and quirks when determining how intelligent, as well as how real, something is.

Academic AI systems, on the other hand, generally are not trying to model humanity, although some are in one way or another. They are mostly trying to model intelligence—the ability to produce the most rational decision given all the possible decisions and the rules. This is usually their one and only requirement and, as such, the reason why all our limitations (such as time or memory) are not given

thought. Also, by distancing themselves from the issues of humanity, they don't run into the sticky problems that we as game people have in dealing with questions about what constitutes intelligence and proper problem solving. They just happily chug along, searching vast seas of agreed-upon possibility for the maximum total value.

Eventually, computing power, memory capacity, and software engineering will become so great that these two separate fields of AI research may no longer be dissociated. AI systems may achieve the kind of performance necessary to solve even the most complex of problems in real time, and as such programming them might be more like simply communicating the problem to the system.

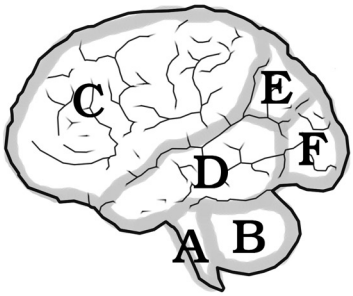
APPLICABLE MIND SCIENCE AND PSYCHOLOGY THEORY

Thinking about the way that the human mind works is a great way to flavor your AI programming with structural and procedural lessons from reality. Try to take this section with a grain of salt, and note that different theories exist on the workings and organization of the mind. This section is more to give you ideas and notions of how to break down intelligence tasks in the same ways that the human mind does it.

Brain Organization

The brain is divided up into subsections, classically (and again, somewhat wrongly) broken into three main groupings: the hindbrain (or brain stem), the midbrain, and the forebrain. Most people have heard these divisions referred to as the reptilian brain, the mammalian brain, and the human brain, but recent research has shown this sort of clear-cut, species-related division to be false. Almost all brains have all three parts, just in different sizes and, in some cases, in dramatically different locations (thus, snakes have a *mammalian* brain region). These brain regions operate independently by using local working memory areas and accessing neighboring synaptic connections to do specific tasks for the organism (fear conditioning is centered in the amygdala, for example). But these regions also are interconnected, some areas heavily so, to perform global-level tasking as well (the amygdala, through the thalamus and some cortical regions, is also a primary first step collection spot for emotional data that will then be sent to the hippocampus for blending with other sensory input and eventual storage). This makes the brain in some way object oriented, with rampant accessor functions and a lack of a real "parent class," however.

The organizational model of the brain has merit when setting up an AI engine, as seen in Figure 1.2, which shows the relative tasking between brain and game systems. By breaking down your AI tasks into specific atomic modules that require



	Organization of:	
	The Brain	A Game AI System
A	Brain Stem - Reflex - Lower Functions/Survival	- Collision - Animation Selection
B	Cerebellum - Motor Center/Sensory Mixing and Coordination	- Physics - Navigation
C	Frontal Lobe - Higher Brain Functions - Emotions - Learning	- Decision Making
D	Temporal Lobe - Memory (Visual and Verbal)	- Learning
E	Parietal Lobe - Sensory Cortex	- Perceptions
F	Occipital Lobe - Visual Processing	- Perception

FIGURE 1.2 Object-oriented nature of the brain related to game AI systems.

little knowledge of the others, you allow other classes to collect the output of these smaller modules together and blend this knowledge into more complex representations that the game character can then use. This also represents the kind of efficiency we should be trying to achieve in our AI systems. Avoid single use calculations and code, or input conditions that are so rare as to be practically hard coded. Alas, inefficiency cannot be completely overcome, but most inefficiencies can be eliminated with clever thinking and programming.

Knowledge Base and Learning

Although the inner workings of the human memory system are not fully understood, or even agreed upon, the common idea is that information is stored in the

form of small changes in brain nerve cells at the synapse connection level (please excuse the heavily simplified neural description). These changes cause differences in the electrical conductivity of different routes through the network and, as such, affect the firing potential of specific nerve cells as well as whole subnetworks. If you use a particular neural pathway, it gets stronger. The reverse is also true. Thus, memory systems use a technique that games could learn a lot from (no pun intended), that of *plasticity.* Instead of creating a set-in-stone list of AI behaviors and reactions to human actions, we can keep the behavior mix exhibited by the AI malleable through plasticity. The AI could potentially keep track of its actions coupled with whether or not the human consistently chooses certain behaviors in response. It could then recognize trends and bias its behaviors (or the requisite counter measures, as a defense) to plastically change the overall behavior mix that the AI uses.

Of course, an AI system would require a dependable system for determining what is "good" to learn, whereas the human brain just stores everything, which can lead to misconception, miscommunication, and even delusion. Although very contextually complex, a filter on AI learning would keep the human player from exploiting a learning system by teaching it misleading behaviors, knowing that the system will respond in kind. Does the AI always use a low block to stop the next incoming punch after the player has punched three times in a row? Then the player will perceive that and punch three times followed by a high punch to get a free hit in on the low-blocking AI.

Another useful lesson from nature is that the rate of memory reinforcement and degradation in the human brain is not the same for all systems. Memories associated with pain aversion, for instance, may never fully extinguish, even if the person only experienced the relation once and never again. This is a good example of nature using *dynamic hard coding.* The plastic changes already mentioned could be "locked in" (by stopping the learning process or moving these changes into a more long-term memory) and thus not be allowed to degrade over time. But like the brain, too much hardcoding, or if used in the wrong place, can lead to odd behavior, turning people (or your game characters) into apparent phobics or amnesiacs.

A further concept to think about is long-term versus short-term memory. Short-term, or working memory, can be thought of as perceptions that can only be held onto for a short time, which are then filtered for importance, and then stored away into longer-term memories, or simply forgotten about. This creates such concepts as attention span, as well as single mindedness. Many games have somewhat digital memory, where the enemy will see you (or even worse, get shot in the arm by you), come after you for some set time period, but if you hide, will completely forget about you and go back to what he was doing. This is classic state-based AI behavior, but it is also very unrealistic and unintelligent behavior. By coding up more analog memory models for our opponent, or just giving him the ability to remember anything longer than a minute or so, he could still go back to his

post, but would be much more sensitive to future attacks, and would probably make it a priority to call for backup, and so forth. For sure, some games do use these types of rules. But there is always room for improvement and expansion of realistic AI behaviors.

The brain also makes use of *modulators,* chemicals that are released into the blood that take a while to degrade. These are things like adrenaline or oxytocin. These chemicals main job is to inhibit or enhance the firing of neurons in specific brain areas. This leads to a more focused mind-set, as well as flavoring the memories of the particular situation in a contextual way. In a game AI system, a modulator could override the overall AI state, or just the behavior exhibited within a certain state. So, conventional state-based AI could be made more flexible by borrowing the concept of modulation. The earlier-mentioned enemy character that you alarmed could transition to an entirely different Alerted state, which would slowly degrade and then transition back down to a Normal state. But using a state system with modifiers, he could stay in his normal Guard state, with an "aggressive" modulator. Although keeping the state diagram of a character simpler, this would require a much more general approach to coding the Guard state.

The human brain learns (which can be thought of as extending your knowledge base of the world through bias and association) by storing things in the different memory centers of the brain. It usually does this in a few, separate ways: exploration or direct experience, imitation, or imaginative speculation. With the possible exception of speculation, which requires quite a sophisticated mental model, game characters may gather information in the same ways. But when you decide that your game is going to use learning techniques, carefully decide how you want the game to come up with its learned data. Keeping statistics on the behaviors that seem to work against the human is one way, and so is recording the things that the human player is doing against the AI opponent and either imitating or trying to deflect those human behaviors.

The problem that games have had with classical AI learning algorithms (as well as the physical model of the brain) is that they usually take many iterations of exposure to induce learning. It is a slippery slope to do learning in the fast-paced, short-lived world of the AI opponent. Most games that use these techniques do all the learning before hand, during production, and then ship the games with the learning disabled, so that the behavior is stable. This will change as additional techniques, infused with both speed and accuracy, are found and made public.

But learning need not be conscious. Influence maps, for instance, can be used by a variety of games to create unconscious learning that will make AI enemies seem much smarter and can be learned with as few as one or two applications. A simple measure of how many units from each side have died on any one spot on the map could give an RTS game's pathfinding algorithm all the information it needs to avoid these *kill zones* where an opponent, human or otherwise, has set up a

deadly trap along some commonly traveled map location. This learning could erode over time or be influenced by attacking units relaying back that they destroyed whatever was making an area a kill zone in the first place. Influence maps are also being used successfully in some sports games. For example, by slightly perturbing the default positions of the players on a soccer field to be better positioned for the passes the human has made in the past, as well as using this same system for the defensive team to allow them to be better able to possibly block these passes.

This type of system allows cumulative kinds of information to be readily stored in a quick and accessible way, while keeping the number of iterations that have to occur to see the fruition of this type of learning very low. Because the nature of the information stored is so specific, the problem of storing misleading information is also somewhat minimized.

Cognition

A flood of data coming from our senses bombards us at all times. How does the brain know which bits of information to deal with first? Which pieces to throw away? When to override the processing it is currently doing for a more life-threatening situation? It does this by using the brain's various systems to quickly categorize and prioritize incoming data. Cognition can be thought of as taking all your incoming sense data, called *perceptions,* and filtering them through your innate knowledge (both instinctual and intuitive) as well as your reasoning centers (which includes your stored memories), to come up with some understanding of what those perceptions mean to you. Think of logic, reason, culture, and all of your personal stored rules as merely ways of sorting out the perceptions you need to be aware of from the background noise. Think of the sheer volume of input coursing into the mind of a person living in a big city. He must contend with the sights, sounds, and smells of millions of people and cars, the constant pathfinding through the crowd, the hawkers, and homeless vying for his attention, and countless other distractions. If his brain tried to keep all this in mind, it would never be able to concentrate sufficiently to perform any task at all.

These perceptions are not all external. The pressures of the modern world cause stress and anxiety that split your attention and fragment your thoughts. Your mind also needs to try to distill the important thoughts inside your own head from the sea of transient, flighty ideas that everyone is constantly engaged in.

In game AI, we don't suffer as much from the flood of data because we can pick and choose our perceptions at any level in the process, and this does make the whole procedure a bit less mystical. In Figure 1.3, you can see a mock-up of a sports game using different perceptions for the various decisions being made by the AI player in the foreground. Make sure, when coding any particular AI subsystem that you only use those perceptions you truly need. Be careful not to oversimplify, or

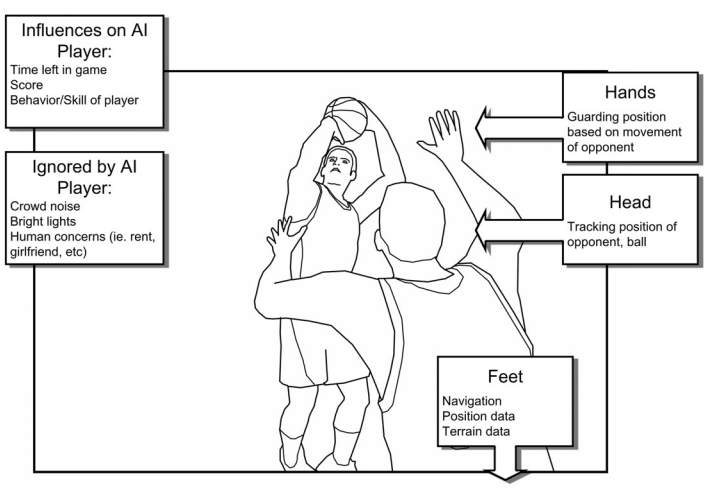


FIGURE 1.3 A visual depiction of various perceptions being taken into account by a game character.

you may make the output behaviors from this subsystem too predictable. An auditory subsystem that only causes an enemy character to hear a sound when its location is within some range to the enemy would seem strange when you set off a particularly loud noise just outside of that range. You should take into account distance and starting volume, so that sounds would naturally trail off as they traveled. You might also want to take into account the acoustics of the environment because sounds will travel much longer distances in a canyon than in an office building, or under water than in open air. These are very simple examples, but you see the notion involved. Perceptions are much more than a single value, because there are usually many ways to interpret the data that each perception represents.

We can think of the systems used in the AI world as filters as well. Whatever technique we are using as our primary decision-making system, to determine the right action to perform, is really just a method of filtering the current game state (by means of the perception variables as registered by the AI) through all the possible things that the AI can do (or some subset of these possibilities, as defined by some rule or game state). Thus, we see the primary observation many people make about AI in general—that it widely comprises focused searching, in some way or another. This is true to some degree. Most AI systems are just different ways of searching through the variety of possibilities, and as such, the topography of your game's possibilities can be used to conceptually consider the best AI technique to use. This topography is generally called the "state space" of the game. If your game's possible outcomes to different perceptions is mostly isolated islands of response, with no

real gray conditions, a state-based system might be the way to go because you're dealing with many more digital possible responses, an almost enumerated state space. However, if the full range of possible responses is more continuous, and would graph out more like a rolling hillside with occasional dips (or another metaphor with more than three dimensions, but you get the idea), a neural net-based system would probably be better because they tend to work better at identifying local minima and maxima in continuous fields of response. We will cover these and the other AI systems in Part III and Part IV of the book; this was merely for illustration.

Theory of Mind

One psychological construct that is again being embraced as a major field of investigation by both behavioralists and cognitive scientists is that of the so-called *Theory of Mind* (ToM). This concept has a good deal of merit in the field of game AI because our primary job is creating systems that *seem* intelligent. A ToM is actually more of a cognitive trait, rather than a theory. It fundamentally means that one person has the ability to understand others as having minds and a worldview that are separate from his own. In a slightly more technical fashion, ToM is defined as knowing that others are *intentional agents,* and to interpret their minds through theoretical concepts of intentional states such as *beliefs* and *desires* [Premack78]. This isn't as complicated as it sounds, however. You could think of this as having the ability to see intent, rather than just strict recognition of action. We do it all the time as adults, and humanize even the most nonhuman of environmental elements. Listing 1.1 shows a bit of code from a Java version (written by Robert C. Goerlich, 1997) of the very early AI program *Eliza,* which, in its time, did a remarkable job of making people believe it was a lot more than it really was.

What does this give us? In human terms, the ability to form a ToM about others usually develops at about the age of three. A commonly used test to determine if the child has developed this cognitive trait is to question the child about the classic "False Belief Task" [Wimmer83]. In this problem, the child is presented with a scene in which a character named Bobby puts a personal belonging, such as a book, into his closet. He then leaves, and while he's away, his little brother comes and takes out the book and puts it in a cupboard. The child is then asked where Bobby will look for his book when he comes back. If the child indicates the cupboard, he reveals that he has yet to develop the understanding that Bobby wouldn't have the same information in his mind that the child does. He therefore does not have an abstract frame of reference, or theory, about Bobby's mind, hence no ToM about Bobby. If the child gives the correct answer, it shows that he can not only determine facts about the world but can also form a theoretical, simplified model of others' minds that includes the facts, desires, and beliefs that they might have; thus providing a theory of this other's mind.

LISTING 1.1 Some Sample Code from a Java Version of Eliza

```
public class Eliza extends Applet

      {

    ElizaChat       cq[];
    ElizaRespLdr    ChatLdr;
    static ElizaConjugate  ChatConj;
    boolean         _started=false;
    Font            _font;
    String          _s;

    public void init()
         {

        super.init();
        ChatLdr = new ElizaRespLdr();
        ChatConj = new ElizaConjugate();

        //{{INIT_CONTROLS
        setLayout(null);
        addNotify();
        resize(425,313);
        setBackground(new Color(16776960));
        list1 = new java.awt.List(0,false);
        list1.addItem("Hi! I'm Eliza.  Let's talk.");
        add(list1);
        list1.reshape(12,12,395,193);
        list1.setFont(new Font("TimesRoman", Font.BOLD, 14));
        list1.setBackground(new Color(16777215));
        button1 = new java.awt.Button
              ("Depress the Button or depress <Enter> to send to
Eliza");
        button1.reshape(48,264,324,26);
        button1.setFont(new Font("Helvetica", Font.PLAIN, 12));
        button1.setForeground(new Color(0));
        add(button1);
        textField1 = new java.awt.TextField();
        textField1.reshape(36,228,348,24);
        textField1.setFont(new Font("TimesRoman", Font.BOLD, 14));
        textField1.setBackground(new Color(16777215));
        add(textField1);
        //}}
```

```
    textField1.requestFocus();
}

public boolean action(Event event, Object arg)
     {
    if (event.id == Event.ACTION_EVENT && event.target ==
                                                          button1)
         {
            clickedButton1();
            textField1.requestFocus();
            return true;
    }
    if (event.id == Event.ACTION_EVENT && event.target ==
                                                      textField1)
         {
            clickedButton1();
            textField1.requestFocus();
            return true;
    }
    return super.handleEvent(event);
}

public void clickedButton1()
     {
    parseWords(textField1.getText());
    textField1.setText("");
    textField1.setEditable(true);
    textField1.requestFocus();

}
public void parseWords(String s_)
     {
    int idx=0, idxSpace=0;
    int _length=0;      // actual no of elements in set
    int _maxLength=200;  // capacity of set
    int _w;

    list1.addItem(s_);
    list1.makeVisible(list1.getVisibleIndex()+1);
    s_=s_.toLowerCase()+" ";
    while(s_.indexOf("' ")>=0)
        s_=s_.substring(0,s_.indexOf("' "))+
                s_.substring(s_.indexOf("' ")+1,s_.length());
```

```
            bigloop: for(_length=0; _length<_maxLength  &&
                             idx < s_.length(); _length++)
                 {
                // find end of the first token
                idxSpace=s_.indexOf(" ",idx);
                if(idxSpace == -1) idxSpace=s_.length();

               String _resp=null;
               for(int i=0;i<ElizaChat.num_chats && _resp == null;i++)
                    {
                   _resp=ChatLdr.cq[i].converse

(s_.substring(idx,s_.length()));
                   if(_resp != null)
                        {
                       list1.addItem(_resp);
                       list1.makeVisible(list1.getVisibleIndex()+1);
                       break bigloop;
                   }
               }
               // eat blanks
               while(s_.length() > ++idxSpace &&
                         Character.isSpace(s_.charAt(idxSpace)));
               idx=idxSpace;

               if(idx >= s_.length())
                    {
                    _resp=ChatLdr.cq[ElizaChat.num_chats-1]
                                          .converse("nokeyfound");
                    list1.addItem(_resp);
                    list1.makeVisible(list1.getVisibleIndex()+1);
               }
           }
       }
       //{{DECLARE_CONTROLS
       java.awt.List list1;
       java.awt.Button button1;
       java.awt.TextField textField1;
       //}}
}
//--------------------
class ElizaChat
     {
```

```
static int          num_chats=0;
private String      _keyWordList[];
private String      _responseList[];
private int         _idx=0;
private int         _rIdx=0;
private boolean     _started=false;
private boolean     _kw=true;
public String       _response;
private String      _dbKeyWord;
public int          _widx = 0;
public int          _w = 0;
public int          _x;
private char        _space;
private char        _plus;

public ElizaChat()
     {
    num_chats++;
    _keyWordList= new String[20];
    _responseList=new String[20];
    _rIdx=0;
    _idx=0;
    _keyWordList[_idx]=" ";

    _space=" ".charAt(0);
    _plus="+".charAt(0);
}

public String converse(String kw_)
     {
    _response = null;
    for(int i=0; i <= _idx - 1;i++){
        _dbKeyWord = _keyWordList[i];

        if(kw_.length()>=_dbKeyWord.length()&&
                _keyWordList[i].equals
                    (kw_.substring(0,_dbKeyWord.length())))
             {

            _widx = (int) Math.round(Math.random()*_rIdx-.5);
            _response = _responseList[_widx];
            _x=_response.indexOf("*");
            if(_x>0)
                 {
```

```
                    _response=_response.substring(0,_x)+

kw_.substring(_dbKeyWord.length(),
                                                        kw_.length());
                    if(_x<_responseList[_widx].length()-1)
                        _response=_response+"?";
                    _response=Eliza.ChatConj
                                        .conjugate(_response,_x);
                    _response=_response.replace(_plus,_space);
                }
                break;
            }
        }

        return _response;
    }

    public void loadresponse(String rw_)
         {
        _responseList[_rIdx]=rw_;
        _rIdx++;
    }

    public void loadkeyword(String kw_)
         {
        _keyWordList[_idx]=kw_;
        _idx++;
    }
}
```

It has been routine in philosophy and the mind sciences in general to see this ability as somewhat dependent upon our linguistic abilities. After all, language provides us a representational medium for meaning and intentionality; thanks to language, we are able to describe other people's and our own actions in an intentional way. This is also probably why Alan Turing gave forth his famous test as to a true measure of intelligence exhibited by a computer program. If the program could communicate successfully to another entity (that being a human), and the human could not tell it was a computer, then it must be intelligent. Turing's argument is thus that anything we can successfully develop a ToM toward must be intelligent—great news for our games, if we can get them to trigger this response within the people who play them.

Interestingly, further studies in chimpanzees and even some lower primates have shown their remarkable abilities toward determining intention and prediction

toward each other and us without verbal communication at our level. So, the ability to form ideas about another's mindset is either biologically innate, can be determined with visual cues, or possibly something else entirely. Whatever it may be, the notion is that we do not require our AI-controlled agents to require full verbal communication skills to instill the player with a ToM about our AI.

This is precisely the kind of inherent nature we want to instill in our games. If we can get the people playing our games to see not a creature in front of them with X amount of health and Y amount of strength, but rather a being with beliefs, desires, and intent, then we will have really won a major battle. This superb leap in suspension of disbelief by the human player can only be achieved if the AI system in question is making the kinds of decisions that a human would make, in such a way as to portray these higher traits and rise above the simple gameplay mechanic involved. In effect, we must model minds, not behavior. Behavior should come out of the minds that we give our AI creations, not from the programmers' minds

Note that this does not mean we need to give our creations perfect problem-solving abilities to achieve this state. People are instinctively seeking to create a ToM about other entities they are dealing with to try to anticipate other entities' actions and thoughts. As game AI programmers, we can use this trait to our advantage when creating entities that we want humans to attribute certain qualities to. In effect, knowledge of this fundamental, low-level goal (that of creatures trying to create a ToM about each other) can help give the programmers and designers guidelines about what types of information to show the player directly, what types to specifically not show, and what types to leave ambiguous. As the illusionists say, "The audience sees what I want it to see."

Take for example, an AI-controlled behavior from a squad combat game. In Figure 1.4, we see the layout of a simple battlefield, with the human player at the bottom of the map, and four CPU enemies closing in on him, moving between many cover points. The simple combat rules for these enemies are the following:

- If nobody is shooting at the player, and I'm fully loaded and ready, I will start shooting. Note that only one player can shoot at a time in this system.
- If I'm out in the open, I will head for the nearest unoccupied cover position, and randomly shout something like "Cover me!" or "On your left!" or even just grunt.
- If I'm at a cover position, I'll reload, and then wait for the guy shooting to be finished, maybe by playing some kind of scanning animation to make it look like he's trying to snipe the player.

Now imagine how this battle will look to the human player. Four enemy soldiers come into view. One starts firing immediately, while the other three dive for cover. Then, the one that was firing stops, shouts "Cover me!," and runs forward for cover

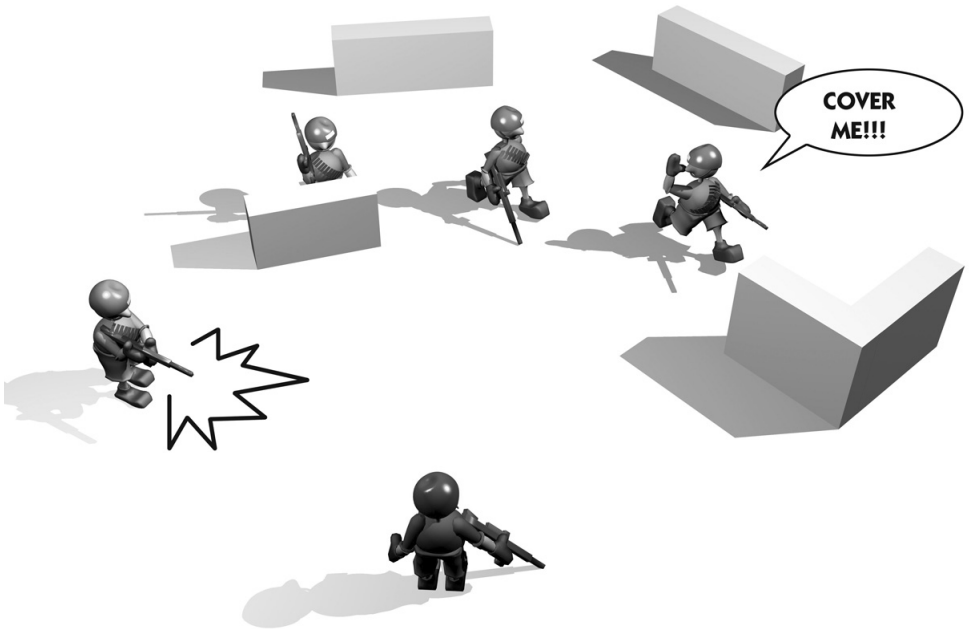


FIGURE 1.4 Emergent theory of mind in a loosely coordinated enemy squad.

as a different soldier pops up and starts firing. Here we have a system in which the soldiers are completely unaware of each other, the player's intentions, or the fact that they're performing a basic leapfrogging advance and cover military maneuver. But because the human player is naturally trying to form a ToM about the enemy, he is going to see this as very tightly coordinated, intelligent behavior. Therefore, the ruse has worked. We have created an intelligent system, at least for the entertainment world.

Bounded Optimality

In trying to achieve a level of rationality in our AI systems, the degree of rationality we are striving for must be declared and allowed to constrain the design of the system. If your goal is near perfect rationality, you had better be able to accept that your program is going to need vastly more time in which to run, unless the decision state space you are working with is very small indeed. For most entertainment games, perfect rationality is unwanted and unnecessary. As discussed earlier, the goal of game AI is to emulate human performance level, not perfect rationality. One of the reasons that humans make mistakes is the idea of bounded optimality (BO).

BO really just means that the system will make the best decision it can, given computation (as well as other resource) restrictions. The possibility of total rationality of the solution is directly linked to the number and amount of limitations. In other words, you get what you pay for. Given a limited view of the world, as well as time constraints, we can construct a decision-making paradigm that will make the best choice within its boundaries.

Like computers, the decision-making ability of people is limited by a number of factors, including the quality and depth of relevant knowledge, cognitive speed, and problem-solving ability. But that only covers the hardware and software. We also suffer from environmental limitations that might make it impossible to fully exploit our brains. We live in a "real-time" world, and must make decisions that could save our lives or make our careers in very short time frames. All these factors come together to flavor our decisions with a healthy dose of incorrectness. So, instead of trying to force our programs to find the ideal solution, we should be merely guiding our decision making in the right direction and working in that direction for as much time as we have. The decisions that come out will then, we hope, be somewhat more human (until, of course, computing power gets to the level that will dilute any time slice restriction to the point of zero) and work well with the limiting constraints of the platform and genre of game we are working on. In effect, we create optimal programs rather than achieve optimal actions.

The concept of BO is also starting to become somewhat prevalent in academic AI circles (as well as in game theory and even philosophy) because so-called optimal solutions to real-life problems are usually computationally intractable. Another reason is that very few real-life problems have no limitations. Given the realities of our world, we need a method of measuring success without requiring absolute rationality.

A problem with trying to use BO methods on many types of systems is that they require incremental solutions; that is, solutions get better by degrees as they are given more resources. Incremental solutions are definitely not universal to all problems, but the types of computationally challenging hurdles that require BO thinking can often be reduced in some way to an incremental level. Pathfinding, for example, can be given several levels of complexity. You might start by pathfinding between very large map sectors, then within those sectors, then locally, and then around dynamic objects. Each successive level is incrementally better than the last, but each level gets the player going in the right direction, at least at some gross level.

Lessons from Robotics

Robotics is one of the few fields that share a vast amount of similar tasking with the world of game AI. Unlike academic endeavors, which deal with large-scale problems that can use exhaustive searches to find optimal results, robots have to deal

with real-time constraints like physics, computation speed, and perception of the environment. In effect, robots have to deal with the computational problems of solving problems intelligently and must master the skills necessary to house this technology into a physical construct that must deal with the real world at some level. This is truly an ambitious task and, as such, tends to take the academic theories and grind them against the stone of reality until finely honed. Because of this, many techniques coming from robotics end up in games because of the inherent optimizing and real-world use that robotics adds to the theoretical AI work done in research labs. The lion's share of the successful pathfinding methods we use in games, including the invaluable A* algorithm, came out of robotics research. Some of the prime lessons that robotics has given us include the following:

Simplicity of design and solution. Many robotics methodologies use the WW model, and games have fully embraced that paradigm. This is because robotics in general is a very hard problem, with an ambitious variety of challenges such as navigating undefined terrains, or recognizing general environmental objects. Every true sense that a researcher bestows on his robot translates to a tremendous amount of technology and study necessary to break down the system into workable parts. If the system can be made to work without the sense, then the solution is just as good, if not better, considering that the expense in both time and money was saved by not having to involve a complex perceptual system. Some of Rodney Brooks's robot designs illustrate this perfectly: instead of trying to navigate areas by recognizing obstacles, and either circumventing them, or calculating how to surmount them, some of his robot designs are largely mindless, insectile creations that blindly use general purpose methods to force their way over obstacles. The lesson here is that while others spend years trying tech-heavy methods for cleverly getting around obstacles and failing, Brooks's robot designs are being incorporated into robots that are headed to Mars.

Theory of Mind. ToM has also been advanced by robotics. Researchers have discovered that people deal better with robots if they can in some way attribute human attributes (if not human thought processes) to the robot. This natural human process, that of humanization, is a good thing for robotics researchers, in that it actually makes the robot seems more intelligent to people, and more agreeable in the eyes of the public. Imagine a robot built to simply move toward any bright light. Humans, when asked to describe this simple behavior, will usually report that the robot "likes lights," or even "is afraid of the dark." By continuing to give a robot the ability to show desires and intentions, instead of raw behaviors, researchers hope to make robots that people will not only toler-

ate but also enjoy working with in the real world. Robotic projects like Cog and Kismet [Brooks98] continue to push the realm of human-robot interaction, mostly through social cues that further people's ToM about the robot and enliven the interaction itself and the learning that the robot is engaging in.

Multiple layered decision architectures. Many modern robotics platforms use a system (sometimes called subsumption) whereupon the decision-making structure of the robot is broken down into layers, the array of which represents high-level to low-level decisions about the world [Brooks91]. This so-called bottom-up behavior design allows robots to achieve a level of autonomy in an environment because they will always have some fail-safe behavior to fall back on. So, a robot might have a very low-level layer whose only goal is to avoid obstacles or other nearby dangers. This layer would get fresh information from the world quite frequently. It would also override or modify behaviors coming from the top of the decision structure because it represents the highest priority of decision making. As you climb the layers, the priority lessens, the amount of interaction with the world lessens, and the overall goal complexity goes up. So, at the highest level, the robot formulates the high-level plan: "I need to leave the room." The layers within this system know nothing about each other (or as little as possible), they simply build on one another in such a way that the various tasks normally associated with the goal at large are specialized and concentrated into distinct layers. This layer independence also creates a much higher robustness to the system since it means that a layer getting confused (or receiving bad data) will not corrupt the entirety of the structure, and thus, the robot may still be able to perform while the rest of the system returns to normalcy.

A structure of this kind is very applicable to game genres that have to make decisions at many levels of complexity concurrently, like RTS games. By sticking to the formal conventions expressed (as well as experimentally tested) by robotics teams using subsumption techniques, we can also gain from the considerable benefits these systems have been found to exhibit, including automatic fault tolerance (between layers of the system), as well as the robustness to deal with any number of unknown or partially known pieces of information at each level. Subsumption architectures do not require an explicit, start-to-finish action plan, and a well-designed system will automatically perform the various parts of its intelligent plan in an order that represents the best way the environment will allow. This book will cover a general way of breaking down AI engine issues using a method something like this approach in Chapter 23.

SUMMARY

This chapter covered some basic AI terminology that we will use in later chapters, some general psychological theory, and some concepts from some other fields that are applicable to AI system design.

- This book will use the term game AI to mean character-based behavioral decision making, further refined by concentrating on tasks that require choosing among multiple good decisions, rather than finding the best possible decision.
- Older games used patterns or let the computer opponent cheat by giving it clandestine knowledge that the human player didn't have; both methods are being increasingly abandoned because of the increasing power of AI systems being used in games.
- AI is becoming increasingly important in today's games, as players demand better opponents to more complex games. This is true even though many games are going online because most people still play single-player modes exclusively.
- Game AI needs to be smart and fun because this is primarily a form of entertainment. Thus, game AI needs to exhibit human error and personality, be able to employ different difficulty levels, and make the human feel challenged, but not overly so.
- Brain organization shows us the use of object-oriented systems that build on each other, in complexity order.
- Like the brain, our AI systems can employ long- and short-term memories, which will lead us toward much more real AI behaviors.
- Learning in a game, like in real brains, can be conscious or unconscious. By using both types, we can model more realistic behavior modification over time, while still focusing our learning to things we want to pay attention to.
- Cognition studies lead us to think of AI reasoning systems as "filters" that take our inputs and lead us toward sensible outputs. Thinking of the nature of the state space that a given game has, and contrasting that with the types of AI techniques available, the right "filter" can be found for your game.
- By making it our goal to give someone playing our game a Theory of Mind about the AI-controlled agents, we extend the attributes of the agent to basic needs and desires, and therefore extend the realism of his decision making to the player.
- Bounded rationality is a formal concept that we can use to visualize our game AI goals, in that we are not searching for optimal actions but, rather, optimal programs that give good solutions while working under many constraints.
- Robotics gives us the notions of design and implementation simplicity, extends our desire for giving our creations a ToM, and provides us with a generic subsumption architecture for designing and implementing autonomous agents from the bottom up.