1 The Illusion of Intelligence

Steve Rabin

1.1 Introduction

 e secret is out. Game AI is seldom about any deep intelligence but rather about the illusion of intelligence. O en we are trying to create believable human behavior, but the actual intelligence that we are able to program is fairly constrained and painfully brittle. Yet, we struggle to put up a good show and strive to keep up the illusion.

Well, maybe we should be a little more purposeful and directly work on propping up the illusion, instead of shoring up the crumbling intelligence. Maybe there are tricks that we can use to fool the players into believing there is real intelligence there. Instead of working toward patching up the actual intelligence, perhaps we should also consciously work toward promoting the appearance of intelligence.

In fact, if you do not work on the illusion side of the equation, you are likely failing to do your job. Expectations play a huge role in how humans perceive the world, and if expectations are not properly managed, then even truly human-level intelligent behavior might be perceived as incompetent and decidedly nonhuman.

 is chapter aims to accomplish two things. To start, we will explain why it is scienti cally possible to trick players into believing there is real intelligence. en, we will look at six concrete ways to perpetuate the illusion of intelligence in the eyes of the player.

1.2 Why the Illusion Works

 ere are three things that work to make players very susceptible to the illusion. First, players want to believe that there are glimmers of real human-level intelligence in their games. Second, humans have this desire to anthropomorphize nonhuman entities, seeing human traits and emotions where there are none. ird, expectations can easily become reality in the mind of the player.

1.2.1 Players Want to Believe

We have the perfect audience to make this work. e players want to believe—in fact, they are willing participants in the illusion. ey want to believe that the fake video game characters have human-like qualities. e players are incredibly forgiving as long as the virtual humans do not make any glaring mistakes. Players simply need the right clues and suggestions for them to share and fully participate in the deception.

1.2.2 Eagerly Ready to Anthropomorphize

When people talk about a thing or creature as if it were human, they are anthropomorphizing it. Anthropomorphism appears to happen naturally as we see things all around us that remind of human traits, emotions, or intentions. Your computer hates you, your car is temperamental, and your recently picked owers are starting to look sad.

One theory put forth by neuroscientists is that similar parts of the brain are involved when we think about both human and nonhuman entities (Gazzola et al. 2007). is suggests that anthropomorphism is the result of using similar processes as when we think about people. It is a sort of a misattribution e ect that is perhaps hardwired into our brains.

Another theory is that when people try to understand incomprehensible behavior, they o en apply familiar human traits to make sense of the situation (Waytz et al. 2010). So when a human-like entity in a game exhibits any kind of behavior, human-like traits are the rst template we try to apply. How can we understand this behavior? Well, let us try applying human behavior and see if that explains what we are seeing. When this happens, the confounding of actual human intelligence with AI is greatly enhanced.

And here we are, as video game developers, presenting human-looking avatars that animate, move, and speak similar to humans. Anthropomorphism is a welcome e ect that encourages the illusion.

1.2.3 The Power of Expectations

Expectations powerfully control how we experience the world. For example, if you believe a bottle of wine is very expensive, you will not only think the wine tastes better, but your actual enjoyment will be more. Researchers at Caltech and Stanford presented people with a 45 bottle of wine and a 5 bottle of wine. Using brain-imaging techniques, they found that the human brain actually experiences more pleasure when the participants believed they were drinking the expensive wine versus the cheap wine, even though both were the same (Plassmann et al. 2008). is is not people reporting that the wine was tastier—neurologically, the brain actually experienced more pleasure.

Similarly, the placebo e ect is a real phenomenon in humans that likely works on the same mechanism of expectations. A placebo is a medically ine ective treatment for a medical condition that is intended to deceive the patient. If we give a person this ine ective treatment, the person will o en have a perceived or actual improvement. is is called the placebo e ect or placebo response. Brain-imaging techniques have shown that placebo causes real physiological changes in the brain that are measurable (Lieberman et al. 2004). e e ect is attributed to the perceptions and expectations of the patient (Kirsch 1985).

Clearly, expectations can have a powerful e ect on what we experience. is further emphasizes that managing player expectations can have a signi cant e ect on promoting the illusion of intelligence.

1.3 Selling the Illusion

Now that we understand why the illusion works and how it is reinforced, our goal is to further encourage and nurture the illusion. is can be done through expectations and performance.

1.3.1 Promoting the Quality of the AI

One simple way to manage expectations is to simply tell the player about the strengths of the AI. Over the years, several games have chosen to tout the quality of their game's AI in press releases, interviews, and box art.

A positive example of this is from 2006 when the game e Elder Scrolls IV: Oblivion heavily promoted their Radiant AI system, which was subsequently used on e Elder Scrolls V: Skyrim, Fallout 3, Fallout: New Vegas, and Fallout 4. Similarly, the series Le 4 Dead let players know that an AI director was helping cra the tension and experience. When players have heard about or can even name the technology behind the game, then that is evidence it could be really good.

An example where this did not work out as well was Madden NFL 98 that bragged in a press release about their liquid AI system used to make their video football players move and ow like water. is partly back red due to the weak analogy, since water is not very intelligent. However, the worst blowback came from a competing football game, NFL GameDay 98, that snidely commented in a Next Generation magazine interview that "Liquid AI is the stu that ran down EA's leg when they saw GameDay."

A subtle use of managing expectations is to use hints during loading screens to highlight aspects of the AI. If the AI is considering several aspects to make a particular decision, perhaps mention this to the player as part of a tip. It will make the player more aware of how the AI is responding, and it might make both the AI and the game seem more interesting.

If your game is doing something truly remarkable, then there might be value in letting players know. However, you need to be con dent that you can deliver and not have it back re.

1.3.2 Perform with Animation and Dialog

 e AI must give the performance of its short lifetime if it wants to impress. e chief way this is done is through subtle animation and dialog. Unfortunately, this is frustrating for most game AI programmers because they do not directly create the animations or the dialog. e best they can do is make compelling arguments to management that these assets should be created. However, this is so important that it really does need to become a priority.

To understand how important this is, let us work through a very short thought experiment. Imagine all of the ways that the player comes to understand and experience an AI character. Let us call this the *vocabulary* of the AI. e vocabulary consists of every dialog clip, every grunt, every animation, every movement, and every interaction. Imagine that an AI character had only two sound clips (an attack grunt and a death cry) and had only four animations (idle, walk, attack, and die). e vocabulary of this particular AI is severely stunted, ironically, with the most interesting behavior happening when it dies. ere is virtually no way you can convey a deeply intelligent AI with such a limited vocabulary.

Fortunately, one way to programmatically add to an AI's vocabulary is with the head look. If the AI's head and gaze can be controlled directly, then you as an AI programmer can wield great power. With head control, you now have the ability for the AI to notice things, compare objects in the environment, anticipate actions, and truly seem aware. Let us illustrate this with a simple scenario. An AI has two enemies to fight: left bad guy and right bad guy. After running a complex evaluation, the AI decides it is best to fight the left bad guy. Fighting ensues, but the problem is that the subtleties of the decision were both instantaneous and hidden. However, what if the AI spends a second looking at each enemy and sizes them up before attacking the left bad guy. What if during the fight with the left bad guy, the AI occasionally looks back at the right bad guy to keep an eye on him. This can telegraph deep intelligence to the player for something that the AI instantaneously chose and is no longer concerned with. However, it is the showmanship of the situation that will convey a conscious and relatable AI character.

Another element of animation that programmers have control over is speed. Fast movements convey being upset, agitated, confused, and nervous. Slow movements convey being relaxed, calm, and in control over the situation. ese are all very human adjectives that we might want the player to liberally apply to our AI characters.

In the right situation, one of the best reactions to play might be one that is completely ambiguous. Consider this stroke of genius from the 2005 game *Façade*. In *Façade*, there were two AI characters who would respond to free-form text entered by the player. When the AI inevitably did not understand a player statement or knew that the statement was of questionable moral content, one of the AI characters might respond simply with a raised eyebrow. e genius of this choice is that the interpretation is le up to the player, because it is an ambiguous reaction. However, it is the perfect time for the player to project all kinds of human properties and thought processes onto the AI, furthering the illusion.

One of the truly great secrets in game AI is to use dialog between AI characters to emphasize and sell the intelligence. e rst prominent example of this was in F.E.A.R. where pursuing AI guards would converse with each other, remarking, "Where did the player go?" and responding with, "He is behind the boxes!" All of a sudden, not only were the guards hunting down the player, but they were working together! e surprising thing was that this was all smoke and mirrors. An AI module simply monitored what was happening and called for these dialog moments as they t the moment (Orkin 2015). It is a great technique that has been used many times since in games such as e Last of Us.

1.3.3 Stop Moving Like a Robot

Although it is crucial to have an adequate vocabulary, the quality of the movement is the other aspect that needs careful attention. If the AI movement is not smooth and lifelike, then the illusion will start to wear thin.

 is is where knowledge of animation techniques, such as anticipation, ease-in, and ease out, can really help. Your goal is to make the movement uid and credible. Work to identify jarring movements and eliminate them. Discontinuous movement is incredibly unnatural and draws attention to the inauthenticity of the situation. Be very sensitive to this when there are collisions. Sometimes, it is much better to brie y allow object penetration, thus avoiding hard collisions and discontinuous movement.

Reaction times are another key area that has the potential to destroy the illusion. Humans are incapable of instantaneous reaction. e fastest a hyperfocused human can react is 0.2 seconds with mental comparisons requiring a bare minimum of 0.4 seconds (Rabin 2015). Use these times as the baseline to always delay the results of a decision. However, realize that distracted or unfocused characters would have much longer reaction times.

A nal aspect of movement that should be mentioned is that your AI should stop pursuing the player relentlessly, similar to a terminator. Intelligent creatures sometimes stop, they re ect, they hesitate, they reconsider, they second-guess themselves, they size up their opponent, and they pause. Movement is an indication of deeper thought processes, and variations in movement can convey all of these thoughts and more. In addition, enemies that temporarily back o are much more enjoyable adversaries. is is an old advice that was well known even during the early 1980s, as shown by the wave-patterned attack/retreat behavior of ghosts in Pac-Man.

1.3.4 Have a Reason to Exist

AI characters need to stop standing around waiting for the player to approach. Characters with nothing to do are a clear signal that the AI is fake and has no real purpose. AI characters should have a reason to exist beyond the player.

For each AI character, this can be as simple as guring out their backstory, and why they are in their current situation. What is their agenda for today? By giving each AI its own motivations (beyond its interactions with the player), it can make each character feel more connected to the game world. A er all, the game world is their home and reality. If it makes sense, it will be much more natural and realistic to the player.

1.3.5 Project a Strong Personality

Personality is the culmination of all the properties of an intelligent character. It implies the entire existence of the character up until the point you interact with it: where it was born, how it grew up, and how it interacts with its reality. It exudes emotions, motives, purpose, and competence. Personality implies incredible depth and authenticity.

Because personality has such power and in uence, a carefully cra ed personality can convincingly convey there is something beneath the surface of your characters, whether there is or not. Personality can be used as a shell around your character to imply humanistic qualities that are simply an illusion. How you leverage this tool can completely change how your players feel about the game.

In addition, a strong personality goes a long way to covering up any inconsistencies in the behavior or logic of a character. Strong personalities can be irrational and unpredictable, allowing incredible leeway in how players might critique their actions.

1.3.6 React Emotionally on Demand

Some programmers have this weird obsession with trying to get game AI to simulate emotions. is seems to stem from the belief that if an AI was truly sad, angry, or happy, then maybe it might nally convince players that some deep kind of intelligence was actually there. is can be equated to the practice of method acting, where an actor will immerse themselves in the character, and through this process, it is hoped that authenticity will emanate out of their performance. It seems to be an unfounded belief that if an AI truly feels emotions, perhaps it will pervade the AI's behavior, and maybe the player will notice.

Without simulating everything that makes up human-level intelligence, this approach for the purposes of games appears misguided. e more straightforward approach would be to directly convey emotions as directly demanded by the situation and the environment. For example, if surrounded by overwhelming forces, fear would be a good emotion to directly convey. Fear does not need to emanate from a simulation within the character; it can be directly shown through dialog and animation when the situation calls for it. If a creature calculates that it is doomed, it should give a performance that matches the situation, conveying a fear of death.

Players can only see an AI's behavior, not what is being simulated. If you want to make an AI appear emotional, then directly show that speci c emotion in the correct situations. is can have a dramatic e ect on how the player feels toward the AI.

1.4 Conclusion

In this chapter, we looked at the importance of promoting the illusion of intelligence. It is not enough for game AI characters to actually have intelligence, but there is a need and obligation to actively sell the illusion. Luckily, there are many things helping us out, such as players who are willing participants, unconscious anthropomorphism, and the power of setting expectations.

Fortunately, there are many levers that we have in order to promote the illusion of intelligence. We covered six main areas: promoting the quality of the AI, perform with animation and dialog, stop moving like a robot, have a reason to exist, project a strong personality, and react emotionally on demand. With many of these tricks up your sleeve, you should not only be able to sell the illusion, but master it.

References

Gazzola, V., Rizzolatti, G., Wicker, B., and Keysers, C. 2007. e anthropomorphic brain: e mirror neuron system responds to human and robotic actions. *NeuroImage*, 35, 1674–1684.

Kirsch, I. 1985. Response expectancy as a determinant of experience and behavior. *American Psychologist*, 40 (11), 1189–1202.

Lieberman, M. D., Jarcho, J. M., Berman, S., Nalibo , B. D., Suyenobu, B. Y., Mandelkern, M., and Mayer, E. A. 2004. e neural correlates of placebo e ects: A disruption account. Neuroimage, 22, 447–455.

Orkin, J. 2015. Combat dialog in FEAR: e illusion of communication. In Game AI Pro 2, ed. S. Rabin. Boca Raton, FL: CRC Press.

Plassmann, H., O'Doherty, J., Shiv, B., and Rangel, A. 2008. Marketing actions can modulate neural representations of experienced pleasantness. Proceedings of the National Academy of Sciences USA, 105, 1050–1054.

Rabin, S. 2015. Agent reaction time: How fast should an AI react? In Game AI Pro 2, ed. S. Rabin. Boca Raton, FL: CRC Press.

Waytz, A., Morewedge, C. K., Epley, N., Monteleone, G., Gao, J. H., and Cacioppo, J. T. 2010. Making sense by making sentient: E ectance motivation increases anthropomorphism. Journal of Personality and Social Psychology, 99(3), 410–435.

2 Creating the Past, Present, and Future with Random Walks

John Manslow

2.1 Introduction

Randomness plays an important role in many games by adding replay value and forc ing the player to adapt to unpredictable events. It o en takes the form of variables with values that change gradually but randomly with time and hence perform what are tech nically known as random walks. Such variables might affect visibility or cloud cover in a weather simulation, the mood of an NPC, the loyalty of a political faction, or the price of a commodity. is chapter will describe the statistical properties of normally distributed random walks and will show how they can be shaped and manipulated so that they remain unpredictable while also being subject to scripted constraints and responsive to player interaction.

We begin by describing how to generate a simple random walk and discuss some of its limitations. We then show how to overcome those limitations using the walk's statistical properties to e ciently sample from it at arbitrary points in its past and future. Next, we discuss how to x the values of the walk at speci c points in time, how to control the general shape of the movement of the walk between them, and how to limit the walk to a speci c range of values. Finally, we describe how to generate random walks with arbitrary probability distributions and to allow for player interaction. e book's web site contains spreadsheets and C source code for all the techniques that are described in this chapter.

2.2 Problems with a Basic Random Walk

One simple way to generate a random walk is to initialize a variable, say x, to some desired starting value x_0, and then, on each step of the simulation of the game world, add a sample from a normal distribution (a bell curve). is process is fast and e cient and produces values of x that start at x_0 and then randomly wander around, perhaps ending up a long way from the starting point or perhaps returning to a point close to it.

is approach is not without its problems, however. For example, how can we work out what the value of x will be two days from now? Equivalently, what value should x be at right now, given that the player last observed it having a particular value two days ago? Perhaps we are simulating the prices of commodities in a large procedurally generated universe, and the player has just returned to a planet that they last visited two days ago. is is the problem of extrapolating a random walk.

2.3 Solving the Extrapolation Problem Using Statistical Methods

One way to solve the extrapolation problem is to quickly simulate the missing part of the random walk. However, this might be computationally infeasible or undesirable, particularly if we are modeling a large number of random walks simultaneously, as might be the case if they represent prices in a virtual economy. Fortunately, we can use statistical methods to work out exactly how x will be distributed two days a er it was last observed based only on the last observed value. Speci cally, the central limit theorem tells us that if x had the value x_0 at time t_0, then at time t, x will be distributed as

$$p \; x \quad N \; x_0, \; t \quad t_0 \quad {}^{2}_{xx} \tag{2.1}$$

is is a normal distribution with the following two characteristics: it has mean x_0, which is the last observed value; and variance $(t \quad t_0) \; {}^{2}_{xx}$, where $t \quad t_0$ is the time since the last observation; and ${}^{2}_{xx}$ is a parameter that controls how quickly the walk can wander away from the starting point. In fact, since the distribution is normal, we know that x will lie in a range of approximately $x_0 \quad 1.96\sqrt{(t \quad t_0) \; {}^{2}_{xx}}$ about 95 of the time. Figure 2.1a shows a random walk generated using Equation 2.1 with x_0 equal to 90 and ${}^{2}_{xx}$ equal to one.

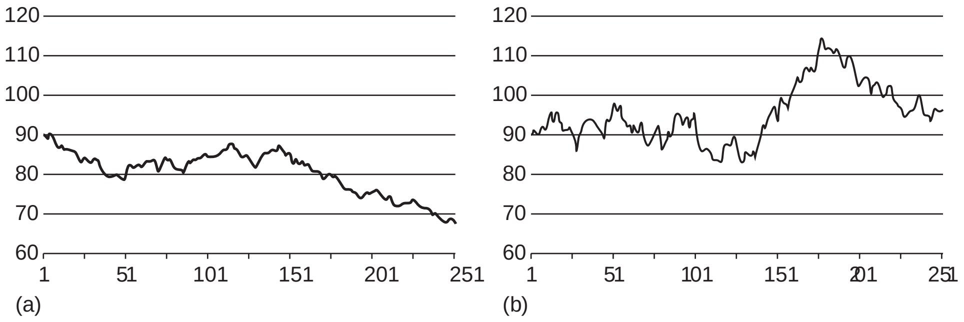

Figure 2.1

(a) shows 250 steps of a random walk generated using the extrapolation equation and (b) shows the effect of increasing $^{2}_{xx}$.

Figure 2.1b shows how increasing the value of $^{2}_{xx}$ fo ve makes the walk wander about much more rapidly.

Of course, we need to select a speci c value for x, and that can be done by sampling from p(x). Although this produces a value of x that is technically not the same as the one that would have been produced by simulating two days' worth of random walk, it is impossible for the player to tell the di erence because the true value and the sampled value have identical statistical properties given the player's limited state of knowledge about how the numbers are generated.

Equation 2.1 actually provides us with a much better way of generating random walks than the naive approach that was mentioned earlier. In particular, because t_0 represents the time between successive samples, it allows us to update the random walk in a way that is invariant to changes in the real times of steps in the simulation of the game world; the statistical properties of the walk will remain the same even if the times between updates are irregular or di erent on di erent platforms.

To generate a random walk using Equation 2.1, rst pick an initial value for x_0 for use in p(x), and then sample from p(x) to get the next value, x_1. e value x_1 can then be used in place of x_0 in p(x), and x_2 can be sampled from p(x). Next, use x_2 in place of x_1 in p(x) and sample x_3, and so on. In each case, the time interval should be the time between the observations of the random walk and hence might be the times between updates to the game world but might also be arbitrary and irregular intervals if the player does not observe the value of the walk on every tick of the game engine.

An interesting feature of Equation 2.1 is that it is time reversible and hence can be used to generate values of x in the past just as easily as ones in the future. Consider if a player visits a planet for the rst time and needs to see a history of the price of a commodity. Equation 2.1 can be used to generate a sequence of samples that represent the price history. is is done in exactly the same way as when samples are generated forward in time except that x_1 would be interpreted as preceding x_0, x_2 would be interpreted as preceding x_1, and so on.

Although having the ability to extrapolate random walks forward and backward in time is extremely useful, we o en need to do more. What happens, for example, if the player has observed the value of a variable but we need to make sure that it has some other specific value in one hour from now? This situation might arise if a prescripted event is due to occur—perhaps a war will affect commodity prices, or there will be a thunderstorm for which we need thick cloud cover. Since we now have two fixed points on the walk—the most recently observed value and a specified future value—it is no longer good enough to be able to extrapolate—we must interpolate.

2.4 Using Interpolation to Walk toward a Fixed Point

Now that we can generate samples from a random walk that are consistent with a single fixed value, we have everything we need to interpolate between two fixed values—we just need to generate samples that are consistent with both. We do this by calculating the probability distributions for x with respect to each of the fixed values and then multiply them together. As Equation 2.1 represents a normal distribution, we can write down the probability distribution for interpolated points quite easily:

$$p(x) \sim N\left(\left(\frac{x_0}{(t-t_0)\sigma_{xx}^2}+\frac{x_n}{(t_n-t)\sigma_{xx}^2}\right)\Big/\left(\frac{1}{(t-t_0)\sigma_{xx}^2}+\frac{1}{(t_n-t)\sigma_{xx}^2}\right),\; 1\Big/\left(\frac{1}{(t-t_0)\sigma_{xx}^2}+\frac{1}{(t_n-t)\sigma_{xx}^2}\right)\right) \tag{2.2}$$

Here x_0 is the first specified value, which occurs at time t_0; x_n is the second, which occurs at t_n; and x is the interpolated value of the walk at any time t. As before, in order to obtain a specific value for x, we need to sample from this distribution. Interpolated values of x are guaranteed to start at x_0 at time t_0, to randomly wander around between t_0 and t_n, and to converge to x_n at time t_n. Interpolation therefore makes it possible to precisely determine the value of the walk at specific points in time while leaving it free to wander about in between.

To generate a walk using Equation 2.2, use x_0 and x_n to sample from $p(x)$ to generate the next value of x and x_1. Next, use x_1 in place of x_0 and sample again from $p(x)$ to generate x_2, and so on—this can be done either forward or backward in time. The interpolation equation has fractal properties and will always reveal more detail no matter how small the interpolated interval. This means that it can be applied recursively to solve problems like allowing the player to see a 25-year history of the price of a particular commodity while also allowing him to zoom in on any part of the history to reveal submillisecond price movements.

2.5 Restricting the Walk to a Fixed Range of Values

One potentially undesirable feature of the random walk that has been described so far is that, given enough time, it might wander arbitrarily far from its starting point. In practice, however, we usually want it to take on some range of reasonable values, and this can easily be done by adding a statistical constraint that specifies that the values of x must, over an infinite amount of time, follow a particular probability distribution. If we choose a normal distribution with mean x and variance ^{2} to keep the math simple, the equation for extrapolating becomes

$$p\ x\quad N\quad \frac{x_0}{t\quad t_0\quad {}^{2}_{xx}}\quad \frac{x}{{}^{2}}\ \Big/\ \frac{1}{t\quad t_0\quad {}^{2}_{xx}}\quad \frac{1}{{}^{2}}\ ,1\Big/\ \frac{1}{t\quad t_0\quad {}^{2}_{xx}}\quad \frac{1}{} \qquad (2.3)$$

and the equation for interpolating becomes

$$p\ x \sim N\ \frac{\dfrac{x_0}{t\quad t_0\quad {}^{2}_{xx}}\quad \dfrac{x}{{}^{2}}\quad \dfrac{x_n}{t_n\quad t\quad {}^{2}_{xx}}}{\dfrac{1}{t\quad t_0\quad {}^{2}_{xx}}\quad \dfrac{1}{{}^{2}}\quad \dfrac{1}{t_n\quad t\quad {}^{2}_{xx}}}\ ,\ 1\Big/\ \frac{1}{t\quad t_0\quad {}^{2}_{xx}}\quad \frac{1}{{}^{2}}\quad \frac{1}{t_n\quad t\quad {}^{2}_{xx}} \qquad (2.4)$$

A walk generated according to these equations is subject to a so bound in the sense that it will lie in the range $x\quad 1.96\sqrt{\quad}$ about 95 of the time but will occasionally wander a little further, exceeding $x\quad 6.11\sqrt{\quad}$ with a probability of less than one in a billion. Figure 2.2a shows a random walk generated according to Equation 2.3 with x equal to 90 and ^{2} equal to 100.

If it is necessary to be absolutely certain that the walk will stay between xed bounds, then it is better to use the unconstrained extrapolation and interpolation equations and postprocess the values they generate. If this is done by re ecting the walk o the bounds whenever it encounters them, the walk will retain the important statistical property that its behavior is invariant to the time steps that are used to generate it. To see how this works in practice, consider generating a random walk that must be constrained to lie between zero and one. is can be done by generating a dummy variable x using the unconstrained extrapolation and interpolation equations but presenting a value x to the player that is derived according to the following rules:

```
if floor(x*) is even then x=x*  floor(x*)
otherwise x=1 x*+floor(x*).
```

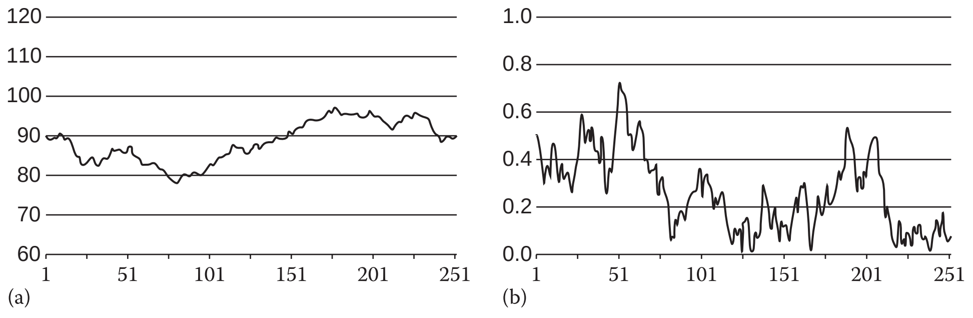

Figure 2.2

(a) shows a random walk that is soft bounded to have a normal distribution with $x^{*} = 90$ and $^{*2} = 100$ and (b) shows a random walk that is hard bounded to lie between zero and one.

x will perform the required random walk between the bounds zero and one, and in the long term will tend toward a uniform distribution. Figure 2.2b shows a random walk between the bounds zero and one that was generated using this technique. If we simply require x to be nonnegative, it is su cient to take the absolute value of x; doing so will produce a random walk that is always nonnegative that will also, in the long term, tend toward a uniform distribution.

2.6 Manipulating and Shaping the Walk with Additive Functions

We have so far described how a random walk can be manipulated by using the interpolation equation with one or more xed points. Interpolation guarantees that the walk will pass through the required points but it gives us no control over what it does in between—whether it follows roughly a straight line, roughly a curve, or makes multiple sudden jumps. Sometimes we want exactly that kind of control, and one way to achieve it is simply to add the random walk to another function that provides the basic form of the walk that we want. In this way, the random walk is just adding random variability to disguise the simple underlying form.

Consider a game where we have a commodity with a price of 90 credits that must rise to 110 credits in one hour from now. We could simply do this by using the interpolation equation and letting the walk nd its own way between the two xed price points, as shown in Figure 2.3a. Alternatively, we could choose a function that starts at 90 credits and rises to

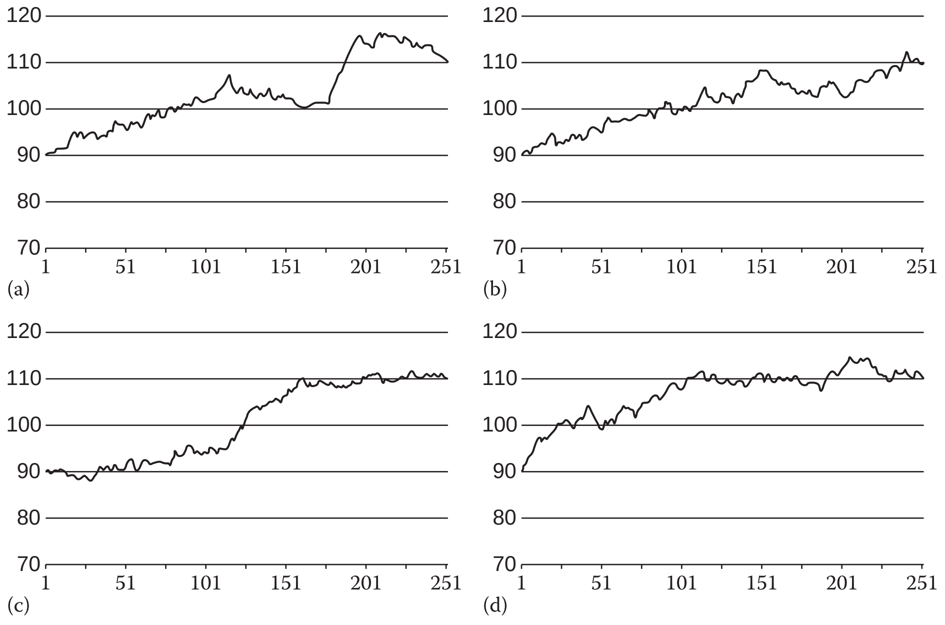

Figure 2.3

(a) shows the result of using the basic interpolation equation to generate a random walk from 90 to 110 in 250 steps, (b) shows the result of using a linear function to shape the walk, (c) shows the result of using the smoothstep function, and (d) shows the result of using the quarter circle function.

110 credits and provides us with the basic shape of the movement we want and add to it a random walk that is bounded by the interpolation equation to start and end at zero.

For example, we might want the price to move roughly linearly between the fixed points so we would use the linear function

$$x = 90(1 - t) + 110t \qquad (2.5)$$

to provide the basic shape. Here, for convenience, t has been scaled so that it has the value zero at the start of the hour and one at the end. By adding the random walk to values of x generated by this formula and using the interpolation formula to generate the random walk in such a way that it starts and ends at zero, the sum of x and the random walk will be 90 at the start of the hour, 110 at the end, and move roughly linearly in between but with some random variation, as shown in Figure 2.3b.

Of course, we do not always have to use a linear function to provide the basic form. Other useful functions are the step function, which produces a sudden jump, the smooth-step function

$$x = 90 + 20(3t^2 - 2t^3) \qquad (2.6)$$

which provides a smoothly curving transition, and the quarter circle function

$$x = 90 + 20\sqrt{1 - (1 - t)^2} \qquad (2.7)$$

which rises rapidly at first and then levels out. Random walks based on the smoothstep and quarter circle functions are shown in Figure 2.3c and d respectively.

2.7 Using Additive Functions to Allow for Player Interaction

We now know how to extrapolate and interpolate random walks and bend and manipulate them in interesting ways, but how can we make them interactive so that the player can influence where they go? Fortunately, player interaction is just another way in which random walks are manipulated and hence all of the techniques that have already been described can be used to produce player interaction.

For example, the player might start a research program that produces a 25% reduction in the basic cost of a particular weapon class over the course of 15 minutes. This effect could be produced by simulating the price using a random walk with zero mean added to a function that represents the average price, which declines by 25% during the course of the research program. This will produce a price that randomly wanders around but is typically 25% lower once the research program has completed than it was before it was started.

Similarly, a player might sell a large quantity of a particular commodity, and we might want to simulate the effect of a temporary excess of supply over demand by showing a temporary reduction in its price. This could be done either by recording an observation of an artificially reduced price immediately after the sale and generating future prices by extrapolation or by subtracting an exponentially decaying function from the commodity's price and allowing the randomness of the walk to hide the function's simple form.

In the case of the research program, we must permanently record the action of the player, and its effect on price because the effect was permanent. In the case of the excess of supply over demand, the e ect is essentially temporary because the exponential decay will ensure that it will eventually become so small that it can be ignored, at which point the game can forget about it unless it might need to create a price history at some point in the future.

2.8 Combining Walks to Simulate Dependent Variables

We have so far discussed how to generate independent random walks and provided a simple set of tools for controlling and manipulating them. In practice, we might need to generate random walks that are in some way related: perhaps we need two random walks that tend to go up and down at the same time, or one that tends to go up when another goes down or vice versa. ese e ects can easily be achieved by adding and multiplying random walks together and through the use of dummy variables.

Imagine that we need to simulate the prices of electronics and robotics products. Since electronics products are a core component of robotics products, we would expect the price of robotics products to increase if the price of electronics products increased—but not for either price to track the other exactly. is e ect can be achieved by modeling the price of electronics products using a random walk and then using another random walk to model a dummy variable that represents either the di erence in the prices of electronics and robotics products or their ratio.

If we decide to use a dummy variable to represent the di erence between the prices, we could use a random walk with an x of 100 and ^{2} of 100 to model the price of electronics products and a walk with x of 25 and ^{2} of 100 to model the di erence in the price of electronics products and robotics products. Since the price of robotics products would be the sum of the values of these two random walks, it would itself be a random walk, and it would have an x of 125 and an ^{2} of 200 and would tend to increase and decrease with the prices of electronics products, as shown in Figure 2.4.

Combinations of random walks can be made arbitrarily complex. e price of a space cra , for example, could be a weighted sum of the prices of its various components plus the value of a dummy variable that represents the deviation of the actual sale price from the total cost of its components. In general, if a random walk is formed by a sum of N

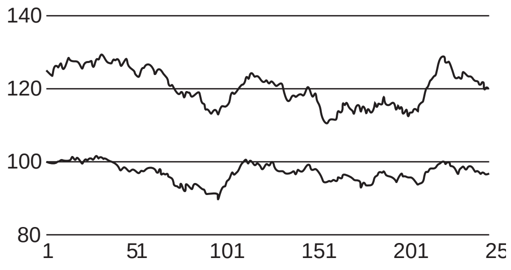

Figure 2.4

The lower random walk has x^* = 100 and ^{*2} = 100 and represents the price of electronics products. The upper random walk is the sum of the lower random walk and another with x^* = 25 and ^{*2} = 100 and represents the price of robotics products.

component random walks $x_1 \ldots x_N$ with means $x_1 \ldots x_N$ and variances $_1^2 \ldots \ _N^2$, which have weights $w_1 \ldots w_N$ in the sum, it will have mean

$$x \qquad \overset{N}{\underset{n\ 1}{\ }} w_n x_n \tag{2.8}$$

and variance

$$^2 \qquad \overset{N}{\underset{n\ 1}{\ }} w_n^2 \ _n^2 \tag{2.9}$$

It could be the case that the player can observe the values of all, some, or none of the components in the sum. We might, for example, want to make the prices of a commodity in a space simulation more similar in star systems that are closer together, and one way to do that is to use a dummy variable in each system to represent a dummy price that the player cannot observe. e price that the player would see in any particular system would then be the weighted sum of the dummy prices of neighboring systems with larger weights being assigned to closer systems. It is interesting to note that if one of the components in Equation 2.8 has a negative weight, then it produces a negative correlation; that is, when its value increases, it will tend to reduce the value of the sum. is can be useful when walks represent mutually competing interests such as the strengths of warring empires.

2.9 Generating Walks with Different Probability Distributions

If a random walk is generated according to the so -bounded extrapolation equation, the values it takes will, over a long period of time, have a normal distribution with mean x and variance ^{2}. is is perfect for most applications, but we will occasionally want to generate a walk with a di erent distribution. For example, share prices have been modeled using log-normal distributions, and we can easily generate a log-normally-distributed random walk by interpreting the extrapolation equation as modeling the natural logarithm of the share price and producing the actual price by applying the exponential function.

More generally, the inverse transformation method is o en used to convert a random variable that is uniformly distributed between zero and one to another random variable with a particular target distribution by applying a nonlinear transformation. For example, taking the natural logarithm of a random variable that is uniformly distributed between zero and one produces a random variable that has an exponential distribution that can be used to realistically model the times between random events.

Although the random walks that have been described in this chapter have normal distributions, they can be made uniformly distributed by hard bounding them to a xed interval, as was described earlier, or by transforming them using the cumulative normal distribution function. Speci cally, if a sample x from a random walk has mean x and variance ^{2}, we can compute the variable

$$y \quad F\ x, x\ ,\ \ ^2 \tag{2.10}$$

whereF is the cumulative normal distribution function. e variable y will be uniformly distributed between zero and one and hence can be used with the inverse transformation method to generate random walks with a wide range of distributions. For example, a linear transformation of y can be used to produce a random walk that is uniformly distributed over an arbitrary range of values, while taking the natural logarithm produces a walk with an exponential distribution. It should be noted that, when the inverse transformation method is used to change a random walk's distribution, the sizes of the steps taken by the walk will not be independent of its value. A walk that is bounded by the inverse transformation method to lie in the range zero to one, for example, will take smaller steps when its value is close to its bounds than when it is far from them.

2.10 Solving the Persistence Problem with Procedural Generation

At the heart of a computer-generated random walk lies a random number generator that can be made to produce and reproduce a speci c sequence of numbers by applying a seed. By recording the seed values that were used in constructing parts of a random walk, those parts can be exactly reconstructed if and when required. For example, if a player visited a planet for the rst time and looked at the history of the price of a commodity over the last year, that history could be created by seeding the random number generator with the time of the player's visit and then applying the extrapolation equation backward in time. If the player returned to the planet a couple of months later and looked at the same history, it could easily be reconstructed based only on the original seed, and hence the history itself would not need to be stored. is is a major bene t, particularly in large open world games that contain many random walks with detailed and observable histories.

2.11 Conclusion

is chapter has provided a selection of simple but powerful techniques for generating and manipulating random walks. e techniques make it possible to exploit the unpredictability and replay value that randomness adds while also providing the control that is necessary to allow it to be both scripted and interactive when necessary. Spreadsheets and C classes that demonstrate the key concepts that are described in this chapter have been provided to make it as easy as possible to apply them in practice.